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

```
693     """
694     if string_network.count('/') == 1:
695         try:
696             mask = int(string_network.split('/')[1])
697         except ValueError:
698             return False
699
```

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

```
118       } else {
119         NSRange range = [component rangeOfString:@"max-age="];
120         if (range.location != NSNotFound) {
121           NSInteger seconds = [[component substringFromIndex:range.location + range.length] integerValue];
122           staleTime = [originalDate dateByAddingTimeInterval:(NSTimeInterval)seconds];
123         }
124       }
```

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

```
169   * @param {string} path
170   */
171  function extractNameFromPath (path) {
172    return path && path !== 'root' ? path.split('/').slice(-2, -1)[0] : 'Root'
173  }
174
175  /**
```

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

```
21
22  export default function parseUserAgent(ua = navigator.userAgent) {
23    ua = ua.toLowerCase()
24    const osRe = OS_REGEXPS.find((re) => re.test(ua))
25    let [, os = 'other', os_version = '0'] = (osRe && ua.match(osRe)) || []
26    if (os === 'iphone os' || os === 'ipad os') os = 'ios'
27    const browserRe = BROWSER_REGEXPS.find((re) => re.test(ua))
```

Parsers are everywhere!

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

```
480                 while(str[i] == '\033') {
481                         do {
482                                 i++;
483                         } while(i < str.length() && str[i] != 'm');
484                         i++;
485                         if(i >= str.length()) break;
486                 }
```

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

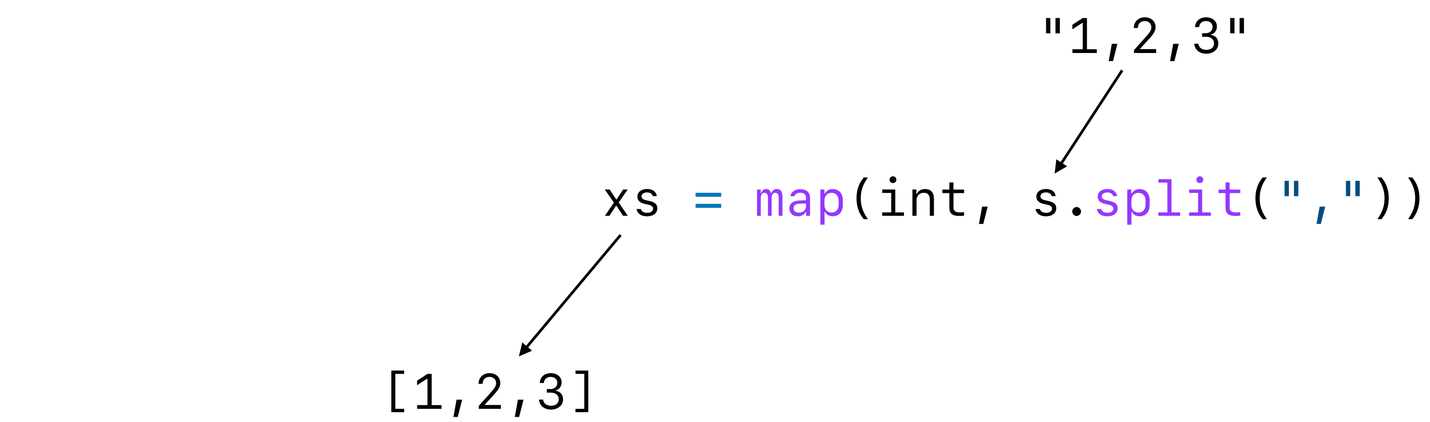
```
372             if "@" not in value or value.startswith("@") or value.endswith("@"):
373                 return False
374             try:
375                 p1, p2 = value.split("@")
376             except ValueError:
377                 # value contains more than one @.
378                 return False
```

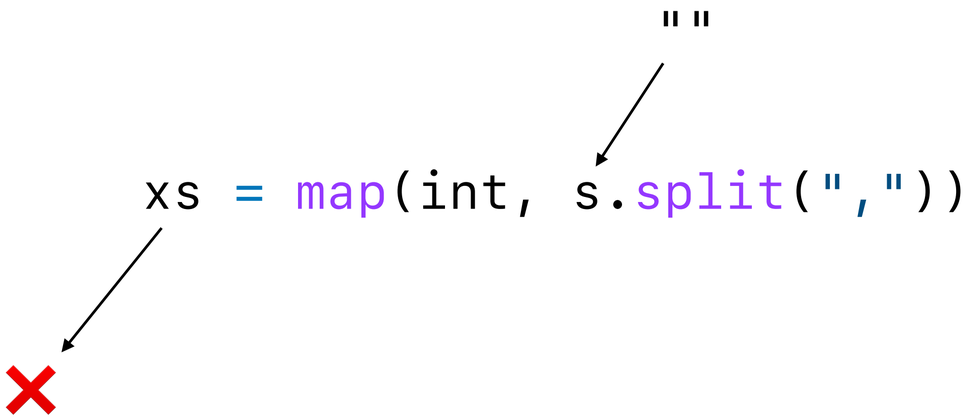
packages/GitPython/lib/git/actor.py — 3 matches | Python

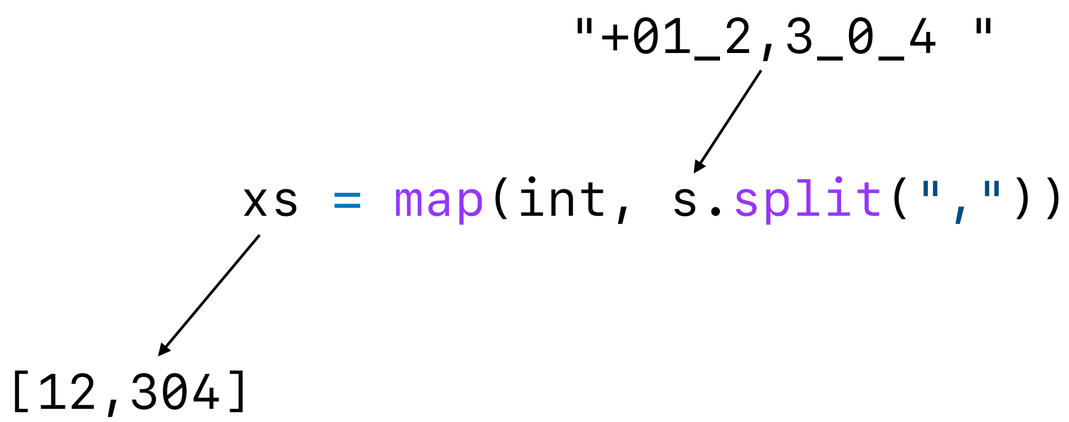
```
35              Actor
36          """
37          if re.search(r'<.+>', string):
38              m = re.search(r'(.*) <(.+?)>', string)
39              name, email = m.groups()
40              return Actor(name, email)
41          else:
```

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

```
1  const React = window.React;
2
3  function csv(string) {
4    return string.split(/\s*,\s*/);
5  }
6
7  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 argumer 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 base is 2, 8, 10, or 16, and so that `int('010', 0)` is not legal, while `int('010')` is, as w

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).

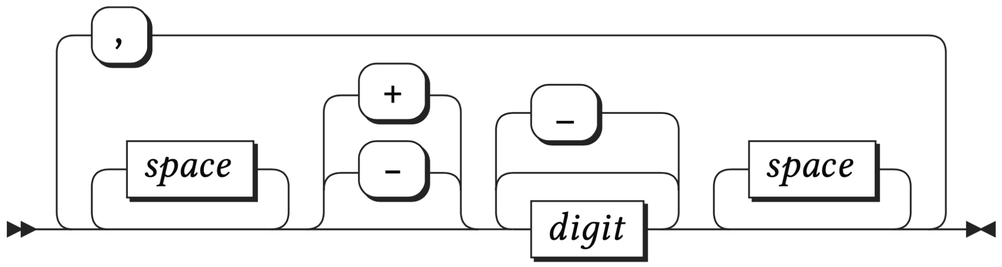
If *sep* is given, consecutive delimiters are not grouped together and are deemed to delimit empty strings (for example, `'1,,2'.split(',')` returns `['1', '', '2']`). The *sep* argument may consist of multiple characters (for example, `'1<>2<>3'.split('<>')` returns `['1', '2', '3']`). Splitting an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



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

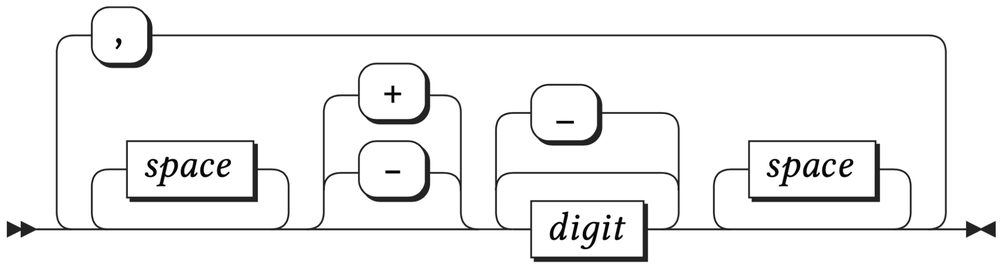
```
xs = map(int, s.split(","))
```

```
"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 […]

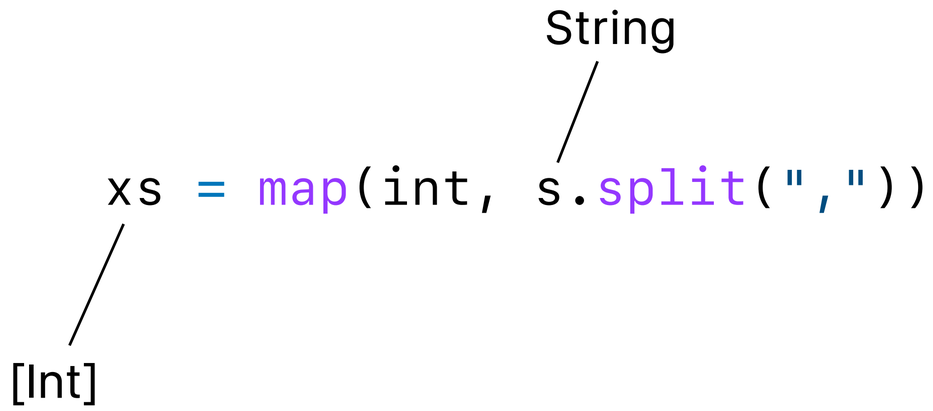


$$
\begin{aligned}
s &\rightarrow int \mid int\ \texttt{,}\ s \\
int &\rightarrow space^{*}\ sign^{?}\ digit\ (\texttt{\_}^{?}\ digit)^{*}\ space^{*} \\
digit &\rightarrow \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8}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

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

Parser : Grammar ≈ Function : Type

Type Inference



✨ Grammar Inference ✨

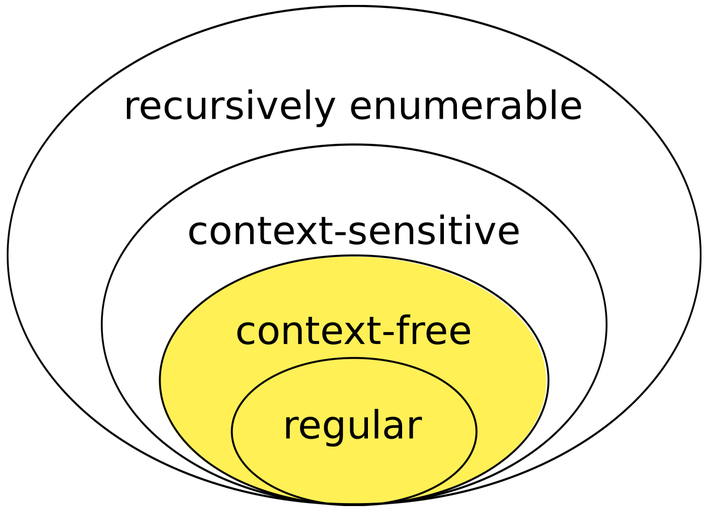
$$
\begin{array}{rcl}
s & \rightarrow & int \mid int\ \texttt{,}\ s \\
int & \rightarrow & space^{*}\ sign^{?}\ digit\ (\_^{?}\ digit)^{*}\ space^{*} \\
digit & \rightarrow & \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid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

String {}

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

[Int]

Toward Grammar Inference



The Chomsky hierarchy

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

Language

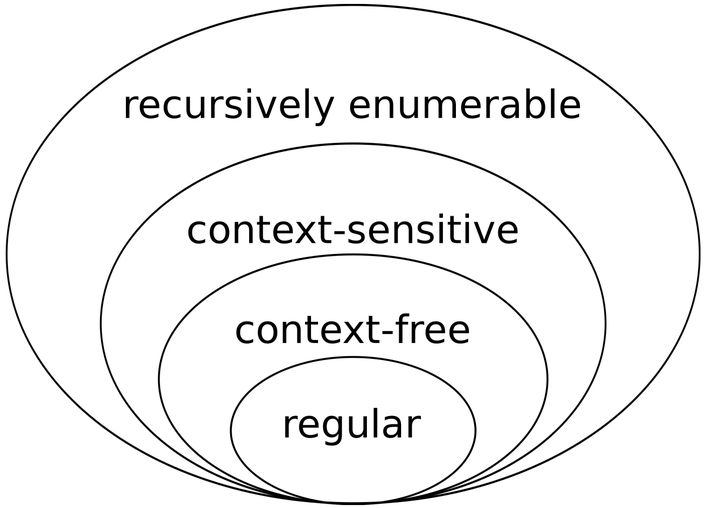
Grammar

$S \to aSb$

$S \to \varepsilon$

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
```

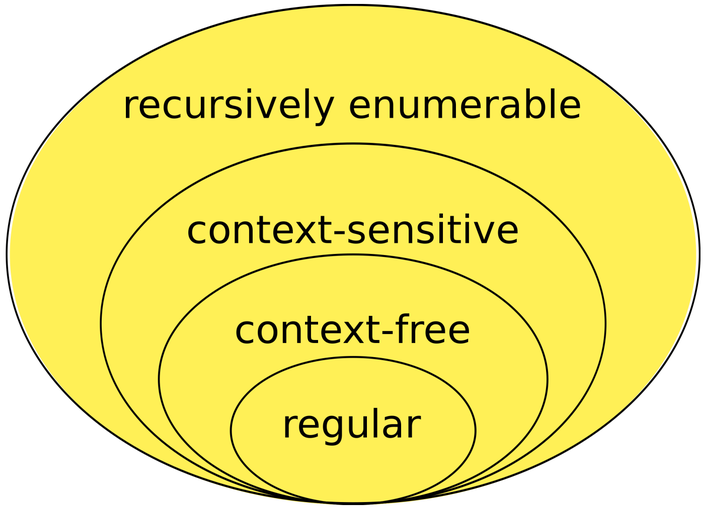


The Chomsky hierarchy

```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}")
```



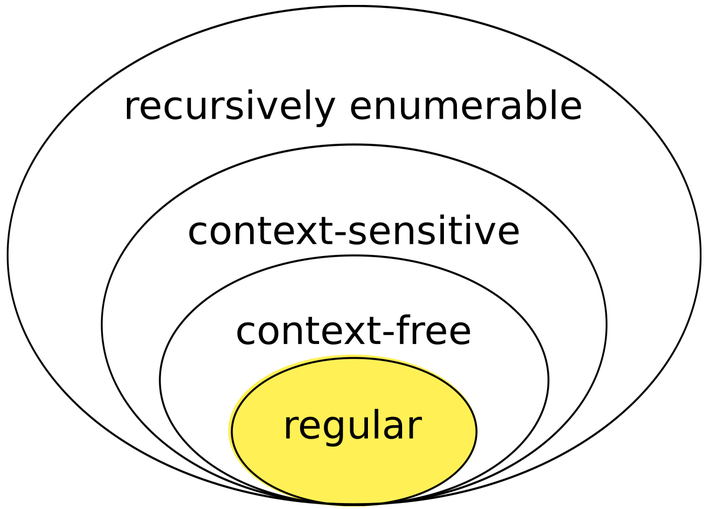
The Chomsky hierarchy

```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}")
```

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

Source Function Specifications

$$\texttt{split}_{\texttt{py}} : \{d : \mathbb{S} \mid len(d) \geq 0\} \rightarrow s : \mathbb{S} \rightarrow \{l : [\mathbb{S}] \mid len(l) \geq 1 \wedge s = l_0 \cdot \Pi_{i=1}^{len(l)-1}(d \cdot l_i)\}$$

$$\texttt{int}_{\texttt{py}} : \{s : \mathbb{S} \mid s = W^* \cdot (+ \mid -)^? \cdot D \cdot (\_^? \cdot D)^* \cdot W^*\} \rightarrow \mathbb{Z}$$

Refinement Type System

$$\frac{}{\{v_1 : b \mid p_1\} <: \{v_2 : b \mid p_2\} \dashv \forall v_1 : b.\ p_1 \Rightarrow p_2[v_1/v_2]}\ \text{SUB-BASE}$$

$$\frac{s_2 <: s_1 \dashv C_I \quad t_1[x_2/x_1] <: t_2 \dashv C_O}{x_1 : s_1 \rightarrow t_1 <: x_2 : s_2 \rightarrow t_2 \dashv C_I \wedge (\underline{x_2} :: s_2 \Rightarrow C_O)}\ \text{SUB-FUN}$$

$$\frac{\Gamma \vdash e \nearrow s \dashv C_1 \quad \Gamma \vdash s <: t \dashv C_2}{\Gamma \vdash e \swarrow t \dashv C_1 \wedge C_2}\ \text{CHK-SYN} \qquad \frac{\Gamma(x) = t}{\Gamma \vdash x \nearrow t \dashv \top}\ \text{SY}$$

```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}")
```

Ad Hoc Parser Source Code

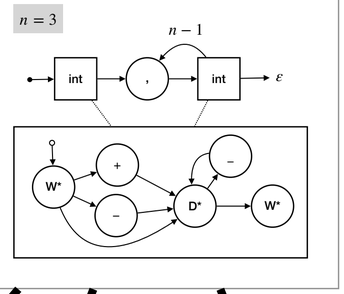
Simplification

```
λs.
  let v1 = split_py "," s in
  let xs = map int_py v1 in
  let v2 = length xs in
  let v3 = equals 3 v2 in
  assert v3
```

Intermediate Representation

Inference

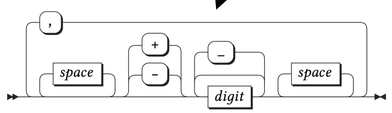
Language Model



Synthesis

```
s      → int | int , s
int    → space* sign? digit (_? digit)* space*
digit  → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
sign   → + | -
space  → ␣ | \t | \n | \v | \f | \r
```

Grammar



Visualization

🚧 ***work in progress*** 🚧

New Possibilities

Interactive Documentation

NEW

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

Inferred grammar:

$$\text{pred} \rightarrow a = v_1 \mid a\ o\ v_2$$
$$a \rightarrow (\Sigma \backslash o)^*$$
$$v_1 \rightarrow \Sigma^*$$
$$v_2 \rightarrow \text{eval}$$
$$o \rightarrow \text{<} \mid \text{<=} \mid \text{>} \mid \text{>=}$$

```
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:

$$\text{pred} \rightarrow a = v_1 \mid a\ o\ v_2$$
$$a \rightarrow (\Sigma \backslash o)^*$$
$$v_1 \rightarrow \Sigma^*$$
$$v_2 \rightarrow \text{eval}$$
$$o \rightarrow \text{<} \mid \text{<=} \mid \text{>} \mid \text{>=}$$

```
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:

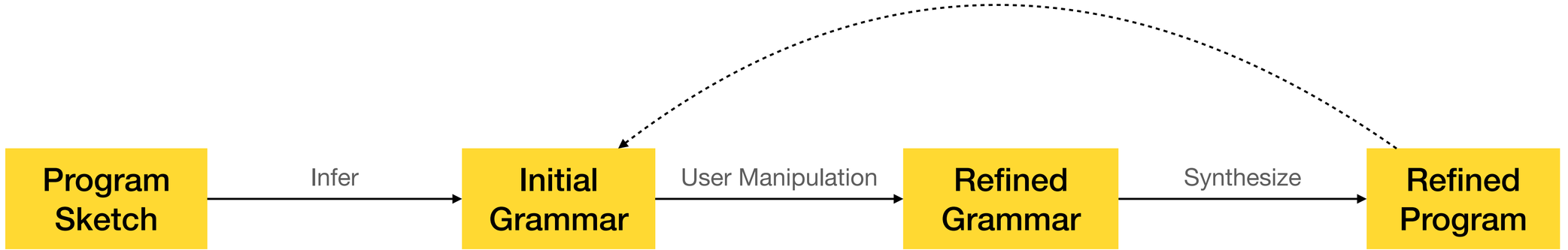
$$\text{pred} \rightarrow a = v_1 \mid a\ o\ v_2$$
$$a \rightarrow (\Sigma \backslash o)^*$$
$$v_1 \rightarrow \Sigma^*$$
$$v_2 \rightarrow \text{eval}$$
$$o \rightarrow \text{<} \mid \text{<=} \mid \text{>} \mid \text{>=}$$

```
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

- semantic program transformation via inferred grammar

  → program sketching




```
01  def foo(s):
02    i = 0
03    while s[i] == "a"
04      i += 1
```

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

foo $\rightarrow a^*$

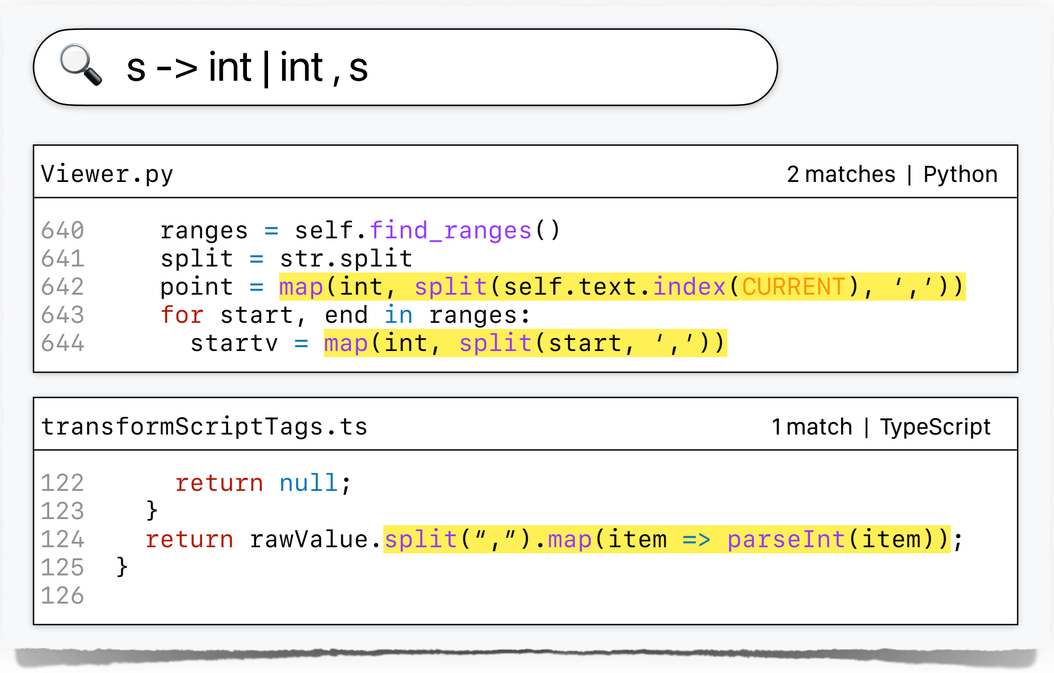
```
01  def foo(s):
02    i = 0
03    while i < len(s):
04      assert s[i] == "a"
```

Mining & Learning

- grammar as *equivalence class* over concrete implementations

  → detecting code clones of ad hoc parsers

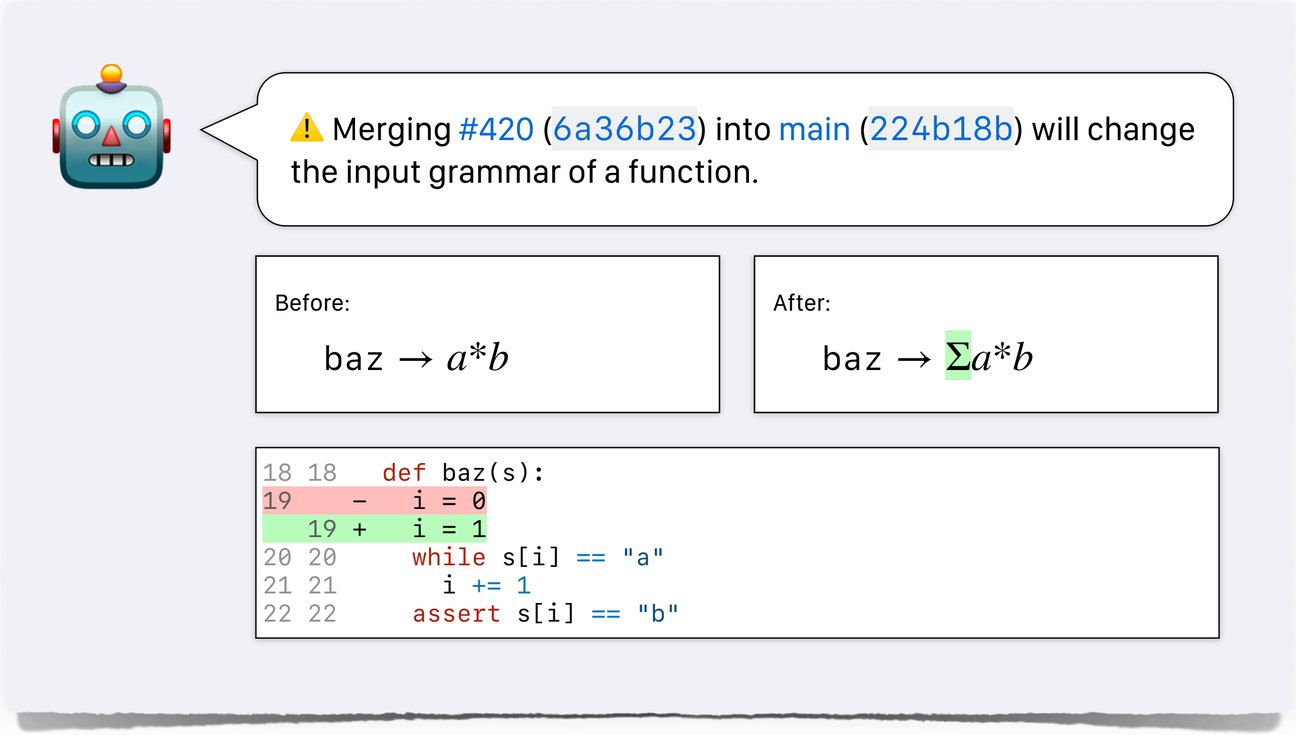
  → grammar-enhanced semantic code search



Mining & Learning

- 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