An Exploratory Study of Ad Hoc Parsers in Python

https://arxiv.org/abs/2304.09733

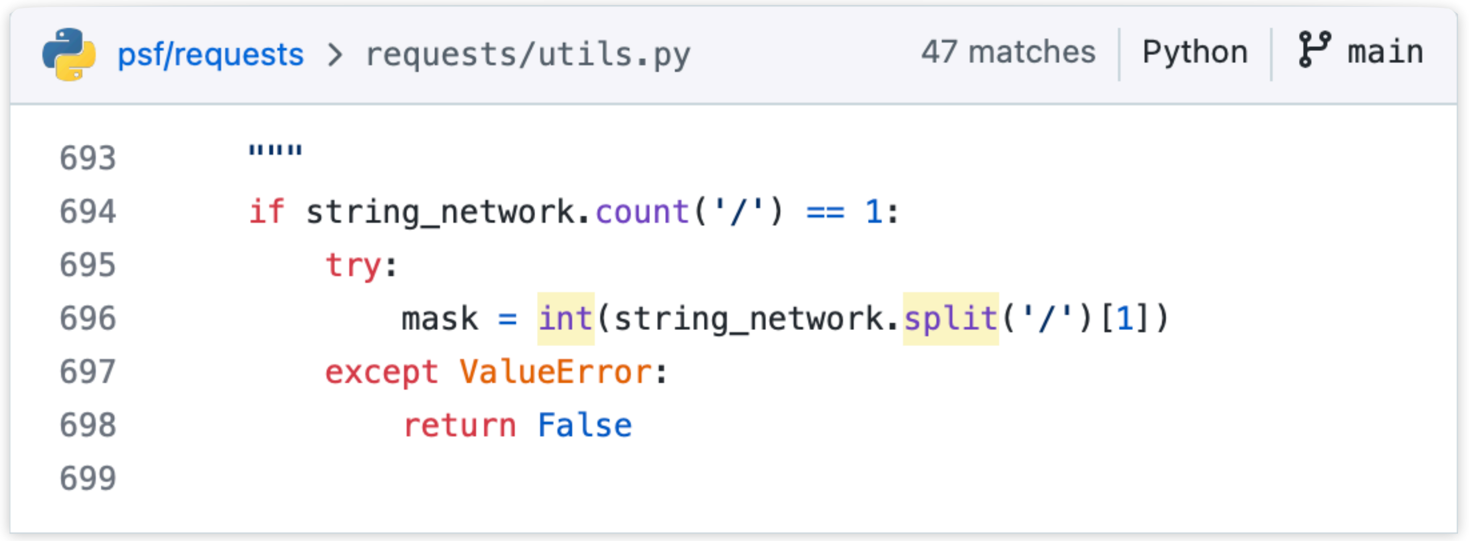
Michael Schröder
TU Wien
Vienna, Austria
michael.schroeder@tuwien.ac.at

Marc Goritschnig
TU Wien
Vienna, Austria

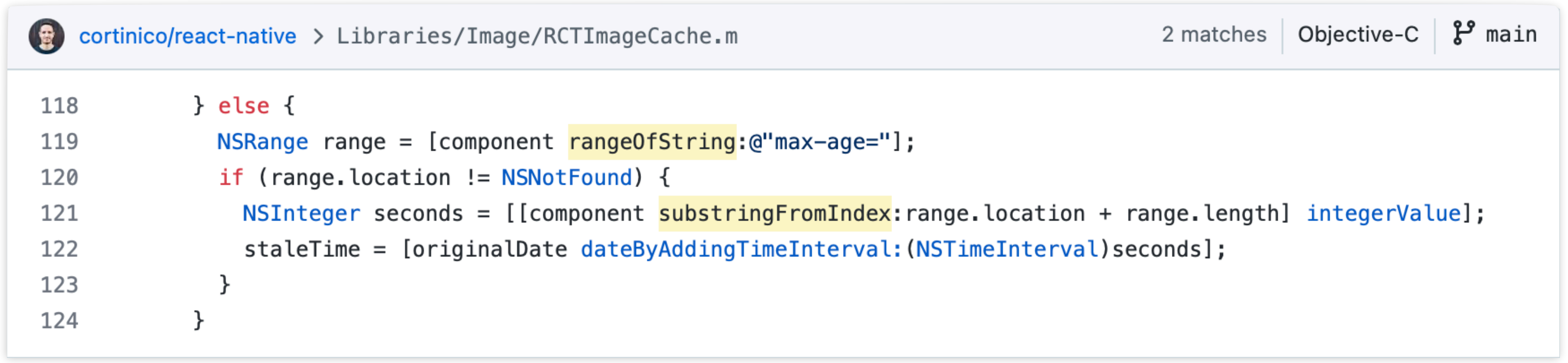
Jürgen Cito
TU Wien
Vienna, Austria
juergen.cito@tuwien.ac.at

MSR 2023 Registered Report
Melbourne, Australia

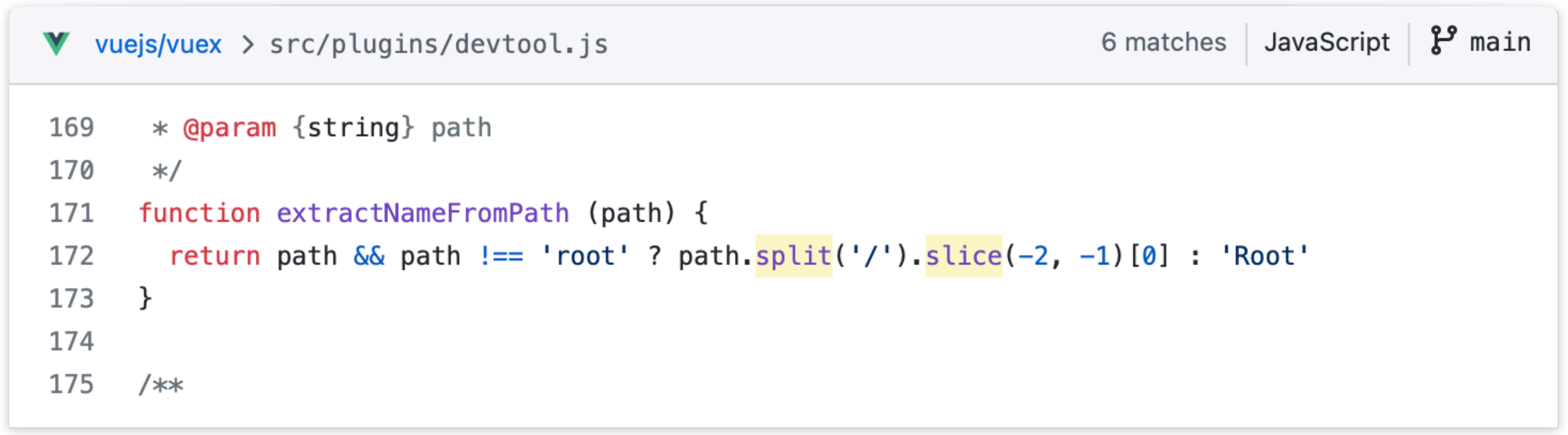
TU WIEN Informatics


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

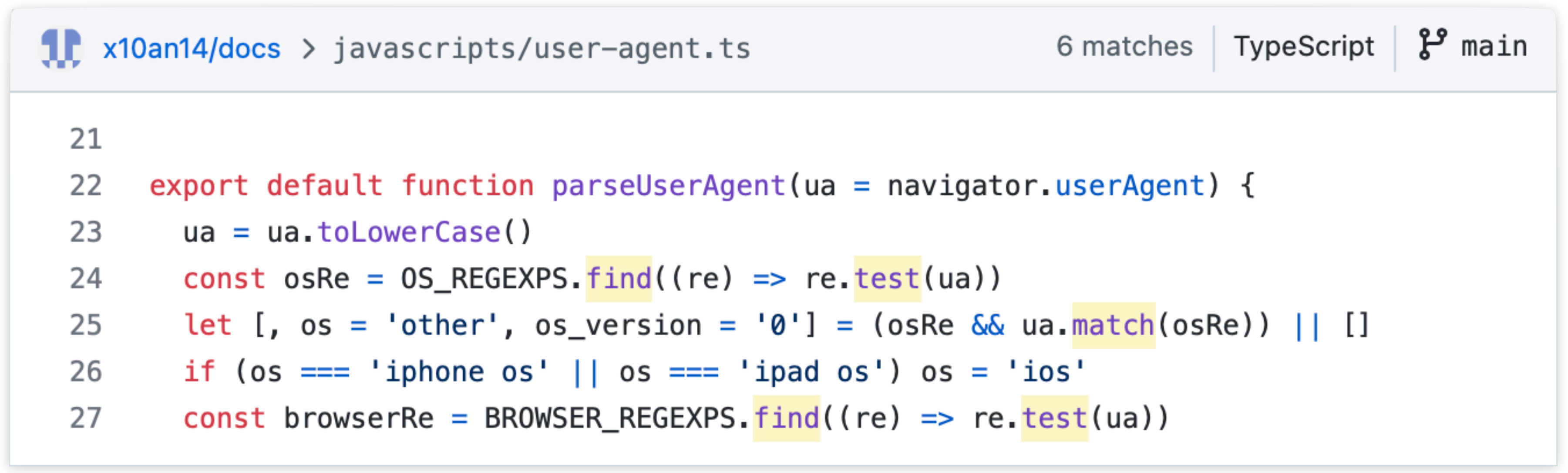
```python
    """
    if string_network.count('/') == 1:
        try:
            mask = int(string_network.split('/')[1])
        except ValueError:
            return False

```


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];
        }
      }
```

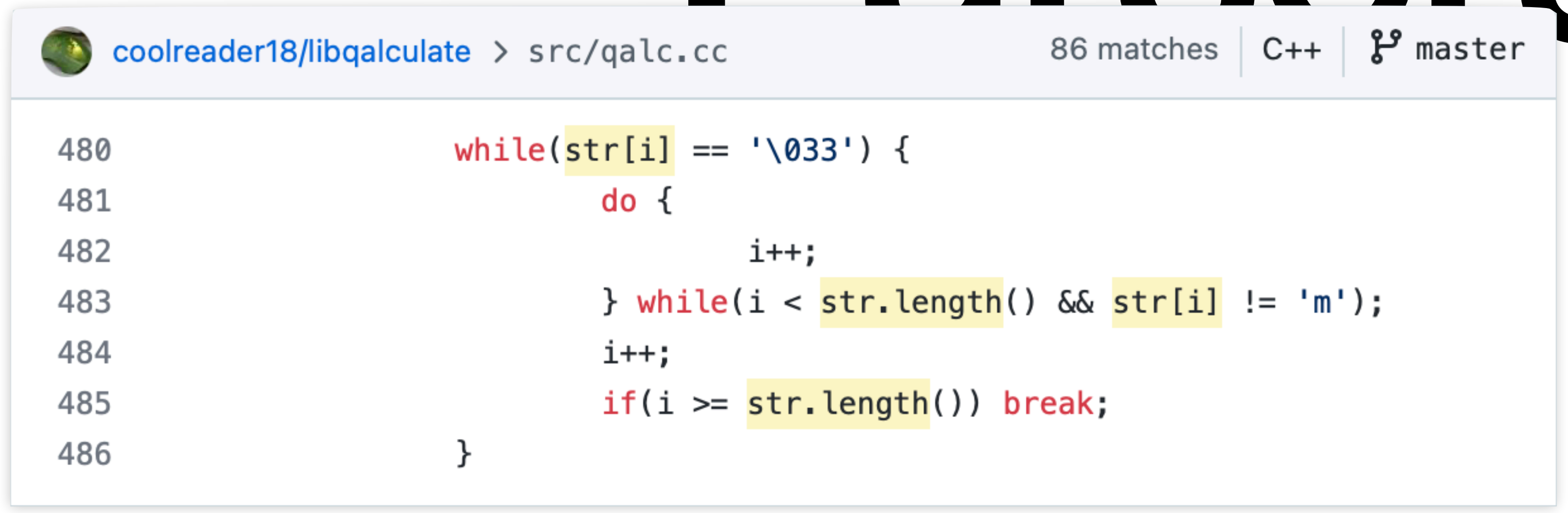

vuejs/vuex > src/plugins/devtool.js 6 matches JavaScript main

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

/**
```

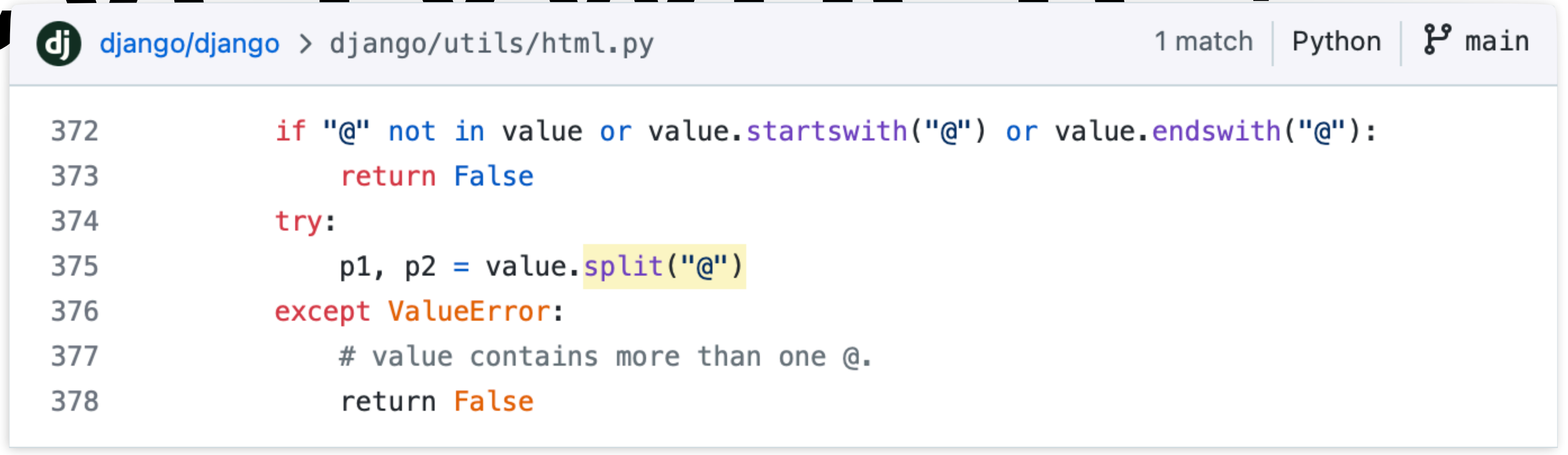

x10an14/docs > javascripts/user-agent.ts 6 matches TypeScript main

```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))
```

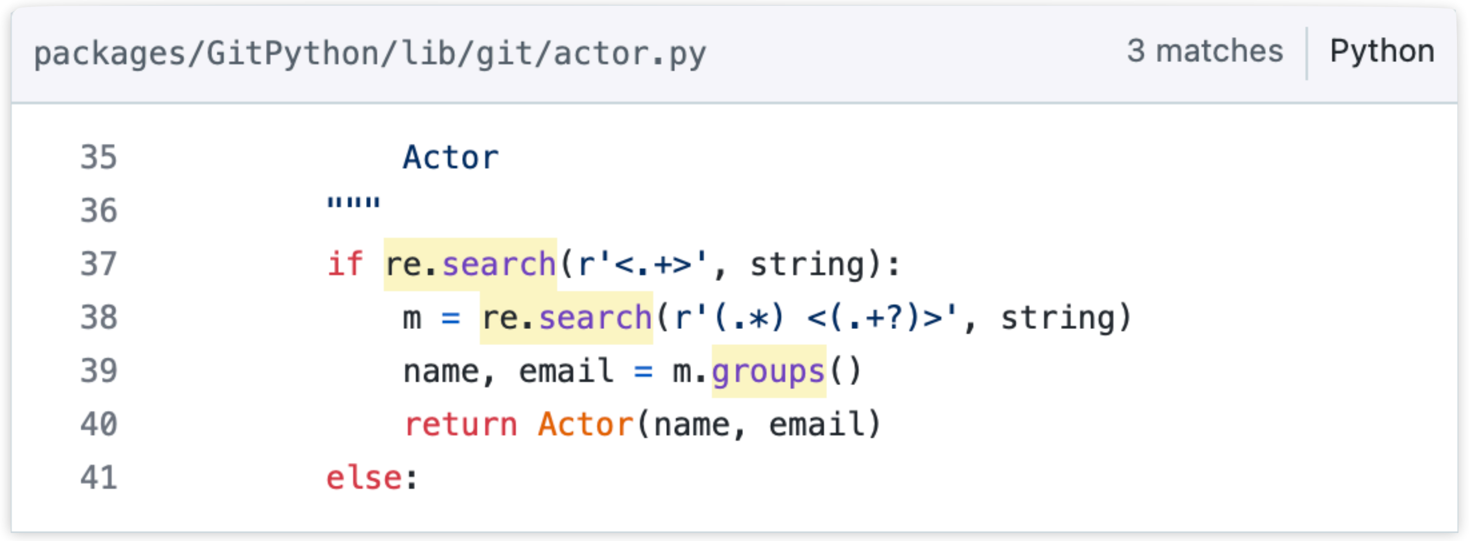
Parsers are everywhere!


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

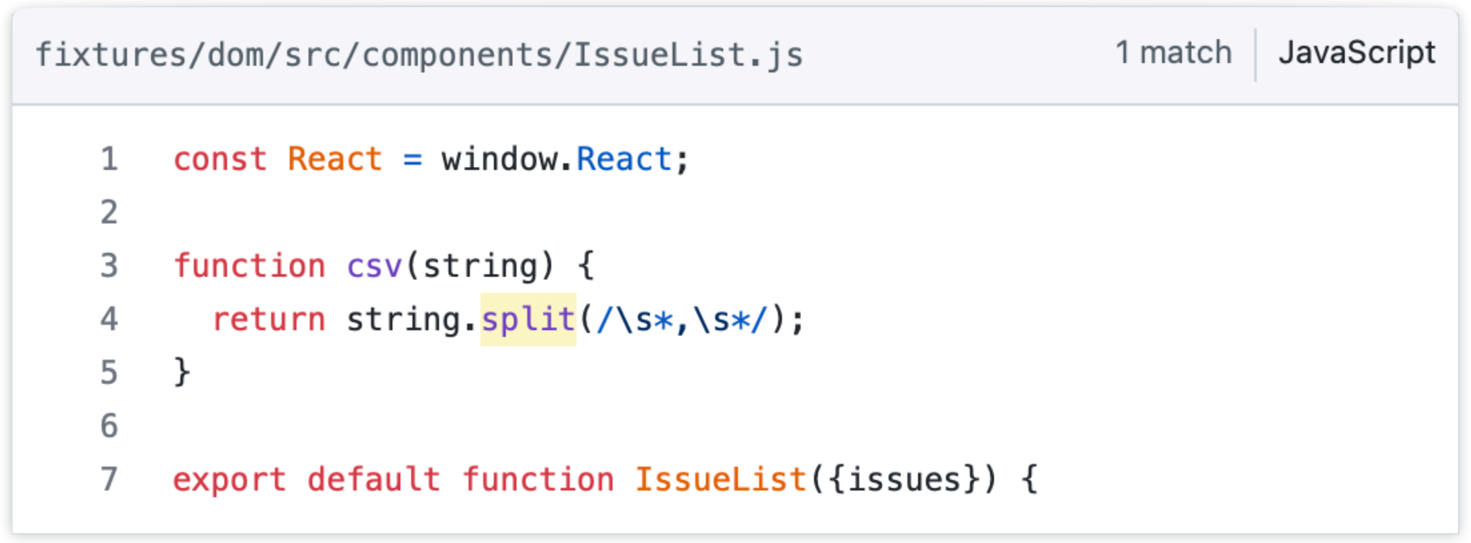
```cpp
        while(str[i] == '\033') {
            do {
                i++;
            } while(i < str.length() && str[i] != 'm');
            i++;
            if(i >= str.length()) break;
        }
```


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:
```

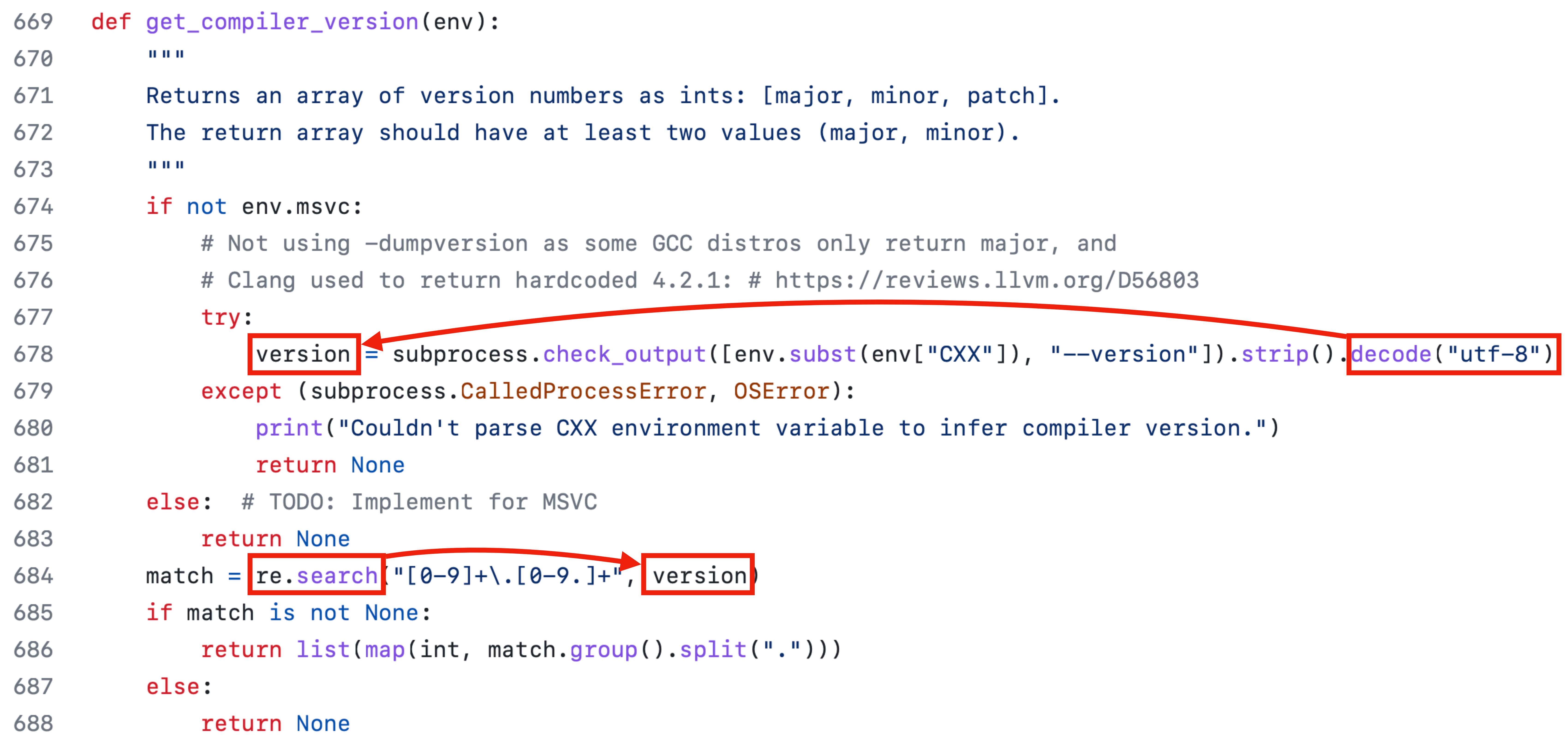

fixtures/dom/src/components/IssueList.js 1 match JavaScript

```javascript
const React = window.React;

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

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

Step 1: Program Slicing

https://boa.cs.iastate.edu

- identify string variables via known functions



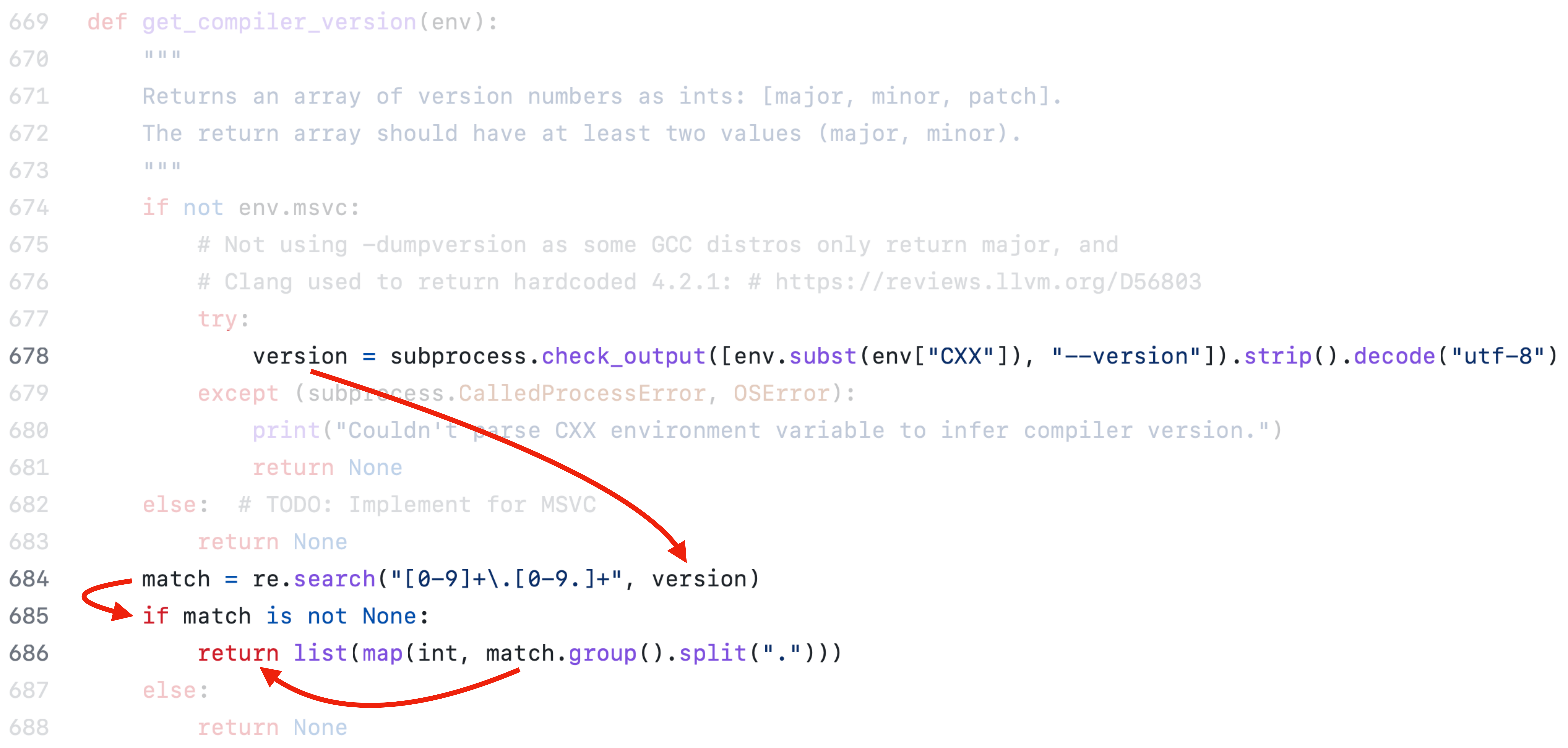
```python
def get_compiler_version(env):
    """
    Returns an array of version numbers as ints: [major, minor, patch].
    The return array should have at least two values (major, minor).
    """
    if not env.msvc:
        # Not using -dumpversion as some GCC distros only return major, and
        # Clang used to return hardcoded 4.2.1: # https://reviews.llvm.org/D56803
        try:
            version = subprocess.check_output([env.subst(env["CXX"]), "--version"]).strip().decode("utf-8")
        except (subprocess.CalledProcessError, OSError):
            print("Couldn't parse CXX environment variable to infer compiler version.")
            return None
    else:  # TODO: Implement for MSVC
        return None
    match = re.search("[0-9]+\.[0-9.]+", version)
    if match is not None:
        return list(map(int, match.group().split(".")))
    else:
        return None
```

https://github.com/neshume/godot/blob/e43c867/methods.py

Step 1: Program Slicing

https://boa.cs.iastate.edu

- identify string variables via known functions
- construct forward slice based on data dependencies

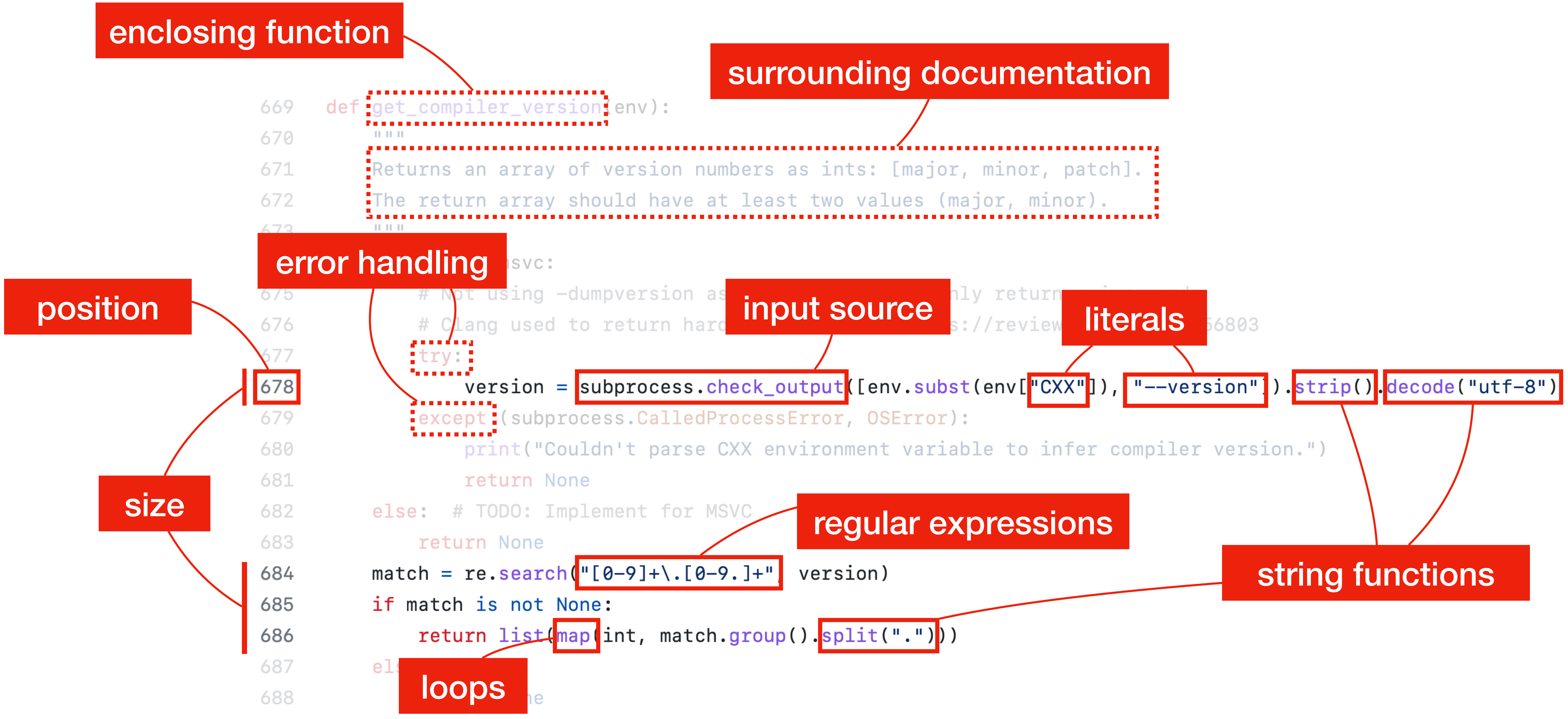


```
def get_compiler_version(env):
    """
    Returns an array of version numbers as ints: [major, minor, patch].
    The return array should have at least two values (major, minor).
    """
    if not env.msvc:
        # Not using -dumpversion as some GCC distros only return major, and
        # Clang used to return hardcoded 4.2.1: # https://reviews.llvm.org/D56803
        try:
            version = subprocess.check_output([env.subst(env["CXX"]), "--version"]).strip().decode("utf-8")
        except (subprocess.CalledProcessError, OSError):
            print("Couldn't parse CXX environment variable to infer compiler version.")
            return None
    else:  # TODO: Implement for MSVC
        return None
    match = re.search("[0-9]+\.[0-9.]+", version)
    if match is not None:
        return list(map(int, match.group().split(".")))
    else:
        return None
```

https://github.com/neshume/godot/blob/e43c867/methods.py

Step 2: Feature Extraction

- 25 metrics across 9 research questions (initial list)



```
def get_compiler_version(env):
    """
    Returns an array of version numbers as ints: [major, minor, patch].
    The return array should have at least two values (major, minor).
    """
                  msvc:
        # Not using -dumpversion as                 nly return
        # Clang used to return har                 s://review                56803

        try:
            version = subprocess.check_output([env.subst(env["CXX"]), "--version"]).strip().decode("utf-8")
        except (subprocess.CalledProcessError, OSError):
            print("Couldn't parse CXX environment variable to infer compiler version.")
            return None
    else:  # TODO: Implement for MSVC
        return None
    match = re.search("[0-9]+\.[0-9.]+", version)
    if match is not None:
        return list(map(int, match.group().split(".")))
    el
                 e
```

https://github.com/neshume/godot/blob/e43c867/methods.py

Step 2: Feature Extraction

- 25 metrics across 9 research questions (initial list)

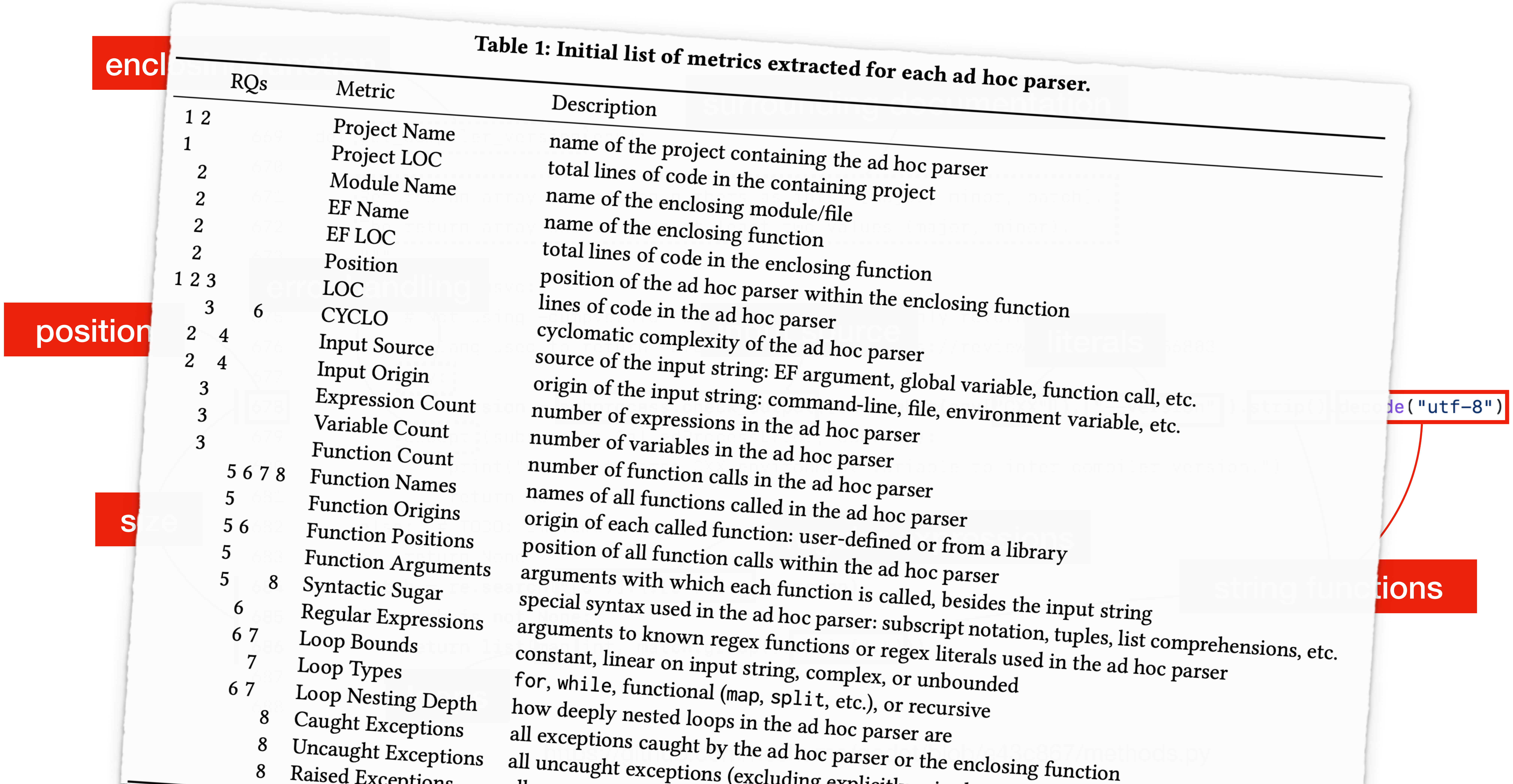

encl
position
s
ions
de("utf-8")

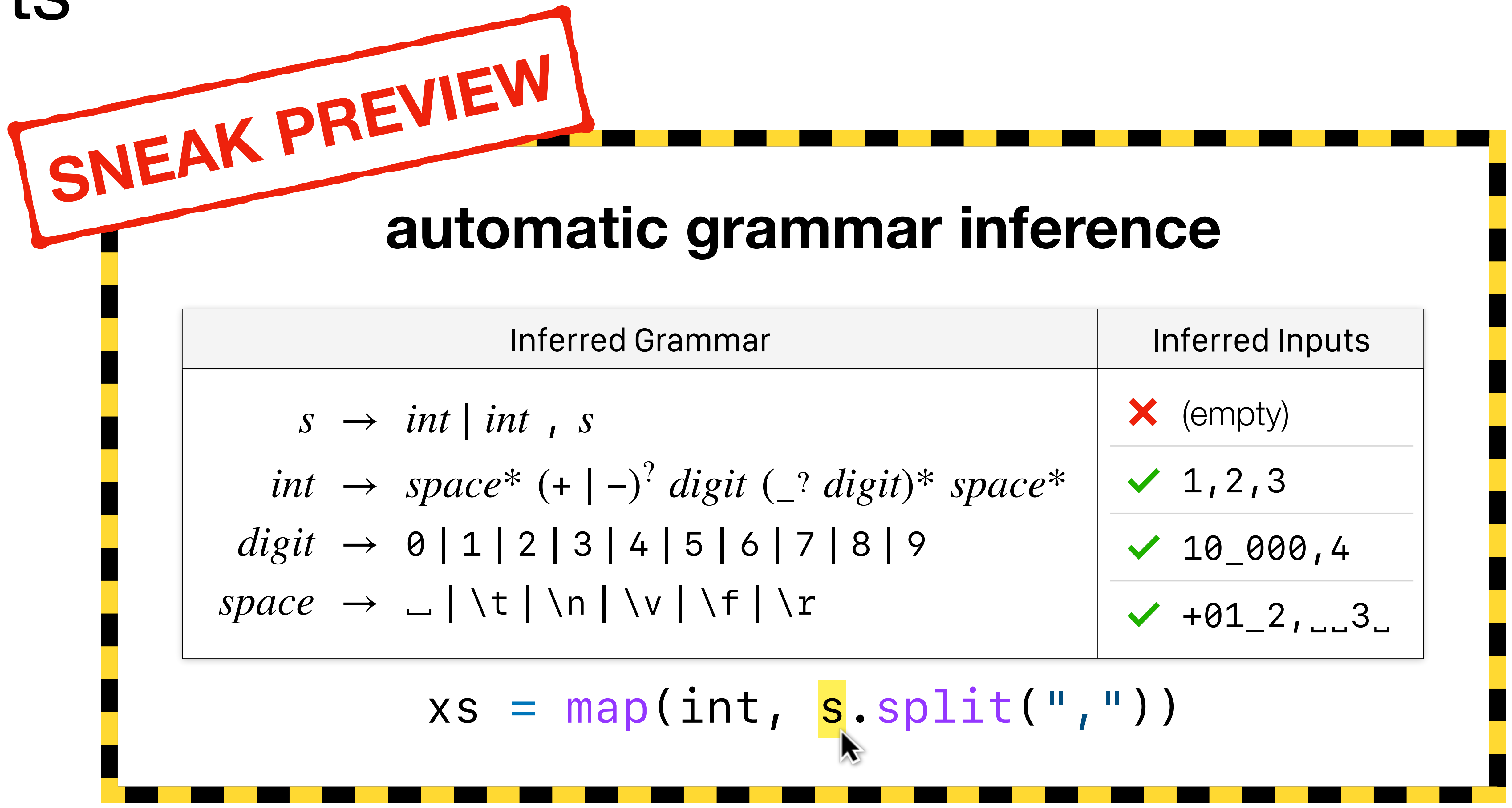
Table 1: Initial list of metrics extracted for each ad hoc parser.

RQs	Metric	Description
1 2	Project Name	name of the project containing the ad hoc parser
1	Project LOC	total lines of code in the containing project
2	Module Name	name of the enclosing module/file
2	EF Name	name of the enclosing function
2	EF LOC	total lines of code in the enclosing function
2	Position	position of the ad hoc parser within the enclosing function
1 2 3	LOC	lines of code in the ad hoc parser
3 6	CYCLO	cyclomatic complexity of the ad hoc parser
2 4	Input Source	source of the input string: EF argument, global variable, function call, etc.
2 4	Input Origin	origin of the input string: command-line, file, environment variable, etc.
3	Expression Count	number of expressions in the ad hoc parser
3	Variable Count	number of variables in the ad hoc parser
3	Function Count	number of function calls in the ad hoc parser
5 6 7 8	Function Names	names of all functions called in the ad hoc parser
5	Function Origins	origin of each called function: user-defined or from a library
5 6	Function Positions	position of all function calls within the ad hoc parser
5	Function Arguments	arguments with which each function is called, besides the input string
5 8	Syntactic Sugar	special syntax used in the ad hoc parser: subscript notation, tuples, list comprehensions, etc.
6	Regular Expressions	arguments to known regex functions or regex literals used in the ad hoc parser
6 7	Loop Bounds	constant, linear on input string, complex, or unbounded
7	Loop Types	`for`, `while`, functional (`map`, `split`, etc.), or recursive
6 7	Loop Nesting Depth	how deeply nested loops in the ad hoc parser are
8	Caught Exceptions	all exceptions caught by the ad hoc parser or the enclosing function
8	Uncaught Exceptions	all uncaught exceptions (excluding
8	Raised Exceptions	

An Exploratory Study of Ad Hoc Parsers in Python

https://arxiv.org/abs/2304.09733

- reveal common syntactic and semantic characterstics of ad hoc parsing code
- understand the nature of ad hoc parsers
- inform future program analysis efforts



automatic grammar inference

Inferred Grammar	Inferred Inputs
$s \rightarrow int \mid int \texttt{,} s$	✗ (empty)
$int \rightarrow space^* (+ \mid -)^? \, digit \, (\_^? \, digit)^* \, space^*$	✓ `1,2,3`
$digit \rightarrow \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9}$	✓ `10_000,4`
$space \rightarrow$ ␣ $\mid$ `\t` $\mid$ `\n` $\mid$ `\v` $\mid$ `\f` $\mid$ `\r`	✓ `+01_2,␣␣3␣`

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