RUFF

An extremely fast Python linter, written in Rust.

RUFF

An extremely fast Python linter, written in Rust.

RUFF

A static analysis toolchain for Python

Linter

Identify ‘problems’ in Python source code: unused imports, deprecated APIs, etc.

Similar to: Flake8, Pylint.

Formatter

Reformat Python source code to follow a consistent style.

Similar to Black, yapf.

Code transformer

Fix issues identified by the linter automatically: remove unused imports, upgrade to modern APIs, etc.

Similar to: isort, pyupgrade.


⌥⌘1
-zsh — 88×22
packse (c76c120) [$+] is 📦 v0.0.0 via 🐍 v3.12.2
❯ ruff check src/packse/fetch.py
src/packse/fetch.py:2:8: F401 [*] `os` imported but unused
|
1 | import logging
2 | import os
| ^^ F401
3 | import shutil
4 | import subprocess
|
= help: Remove unused import: `os`
src/packse/fetch.py:14:11: UP007 [*] Use `X | Y` for type annotations
|
13 | def fetch(
14 | dest: Optional[Path] = None,
| ^^^^^^^^^^^^^^ UP007
15 | ref: Optional[str] = None,
16 | repo_url: str = f'https://github.com/astral-sh/packse',
|
= help: Convert to `X | Y`


⌥⌘2
less
@@ -1,26 +1,22 @@
import logging
-import os
import shutil
import subprocess
import tempfile
import time
from pathlib import Path
-from typing import Optional
logger = logging.getLogger(__name__)
def fetch(
-    dest: Optional[Path] = None,
-    ref: Optional[str] = None,
-    repo_url: str = f'https://github.com/astral-sh/packse',
+   dest: Path | None = None,
+   ref: str | None = None,
+   repo_url: str = "https://github.com/astral-sh/packse",
repo_subdir: str = "scenarios",
:

RUFF

Empowering impactful projects across open source

Over the past year, Ruff has grown to over **18M monthly downloads**, over **500,000 VS Code installs**, over **27,000 GitHub stars**, and over **400 open-source contributors**.

DOWNLOADS PER MONTH

18.0M

GITHUB STARS

27,011

VS CODE INSTALLS

582,301

CONTRIBUTORS

431


scale


dbt


OpenAI


MISTRAL
AI_


Microsoft


ASTRONOMER


MongoDB


dagster


Hugging Face


snowflake


Apache
Airflow


SciPy

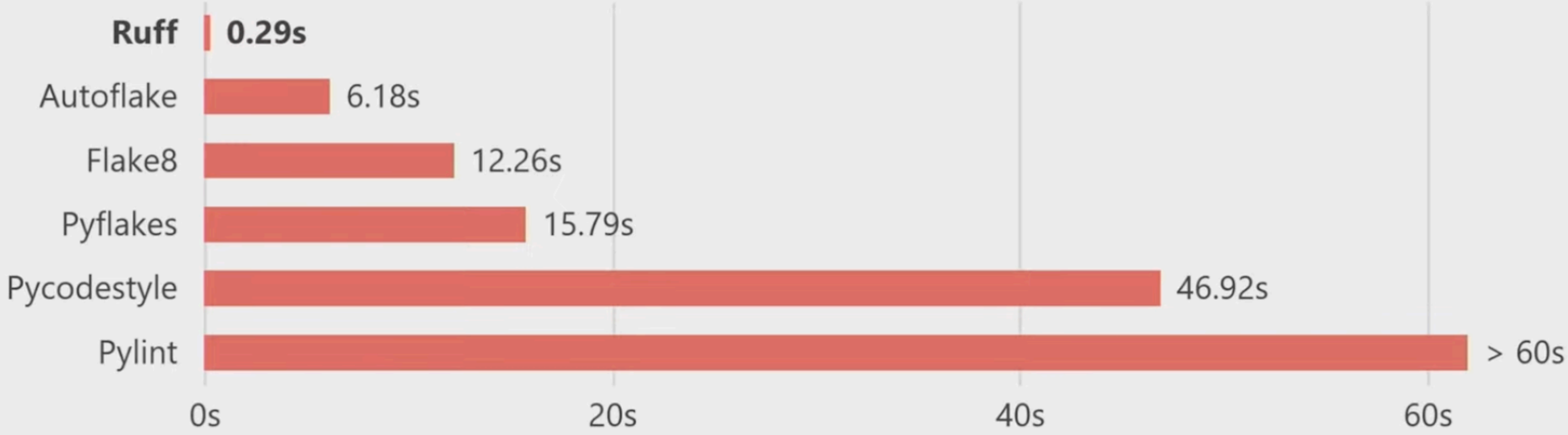

pandas

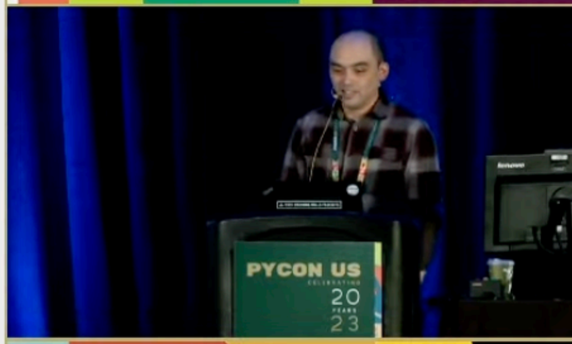

FastAPI


replit

BENCHMARKS

An extremely fast toolchain for Python





20
YEARS CELEBRATION
PYCON US
23

20 years of PyCon and more!

24:40 / 26:57

ASTRAL

WHAT MAKES RUFF FAST?

ASTRAL

HOW DOES RUFF WORK?

From source code to diagnostics

1. Tokenization
2. Parsing
3. Semantic analysis
4. Diagnostic analysis

From source code to diagnostics

1. **Tokenization**
2. Parsing
3. Semantic analysis
4. Diagnostic analysis

From source code to diagnostics

1. Tokenization

2. Parsing

3. Semantic analysis

4. Diagnostic analysis

```
y = x + 12
```

From source code to diagnostics

1. Tokenization

2. Parsing

3. Semantic analysis

4. Diagnostic analysis

```
y = x + 12
```

```
Name(y)
Equal
Name(x)
Plus
Int(12)
```

From source code to diagnostics

1. Tokenization

2. Parsing

3. Semantic analysis

4. Diagnostic analysis

```
y = x + 12
```

```
Name(y)
Equal
Name(x)
Plus
Int(12)
```

From source code to diagnostics

1. Tokenization

2. Parsing

3. Semantic analysis

4. Diagnostic analysis

```
y = x + 12
```

```
Name(y)
Equal
Name(x)
Plus
Int(12)
```

From source code to diagnostics

1. Tokenization

2. Parsing

3. Semantic analysis

4. Diagnostic analysis

```
y = x + 12
```

```
Name(y)
Equal
Name(x)
Plus
Int(12)
```

From source code to diagnostics

1. Tokenization

2. Parsing

3. Semantic analysis

4. Diagnostic analysis

```
y = x + 12
```

```
Name(y)
Equal
Name(x)
Plus
Int(12)
```

From source code to diagnostics

1. Tokenization

2. Parsing

3. Semantic analysis

4. Diagnostic analysis

```
y = x + 12
```

```
Name(y)
Equal
Name(x)
Plus
Int(12)
```

From source code to diagnostics

1. Tokenization

2. Parsing

3. Semantic analysis

4. Diagnostic analysis

```
y = x + 12
```

```
Name(y)
Equal
Name(x)
Plus
Int(12)
```

From source code to diagnostics

1. Tokenization

2. Parsing

3. Semantic analysis

4. Diagnostic analysis

```
Name(y)
Equal
Name(x)
Plus
Int(12)
```

From source code to diagnostics

1. Tokenization

2. Parsing

3. Semantic analysis

4. Diagnostic analysis

TOKEN STREAM

```
Name(y)
Equal
Name(x)
Plus
Int(12)
```

From source code to diagnostics

1. Tokenization

2. Parsing

3. Semantic analysis

4. Diagnostic analysis

From source code to diagnostics

1. Tokenization
2. **Parsing**
3. Semantic analysis
4. Diagnostic analysis

```
y = x + 12
```

From source code to diagnostics

1. Tokenization
2. **Parsing**
3. Semantic analysis
4. Diagnostic analysis

From source code to diagnostics

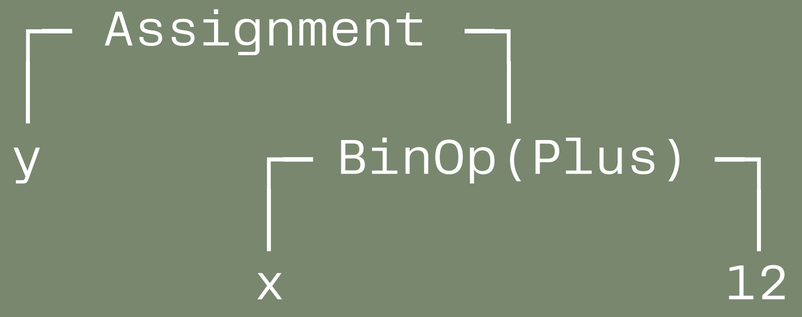
1. Tokenization
2. **Parsing**
3. Semantic analysis
4. Diagnostic analysis

From source code to diagnostics

1. Tokenization
2. **Parsing**
3. Semantic analysis
4. Diagnostic analysis

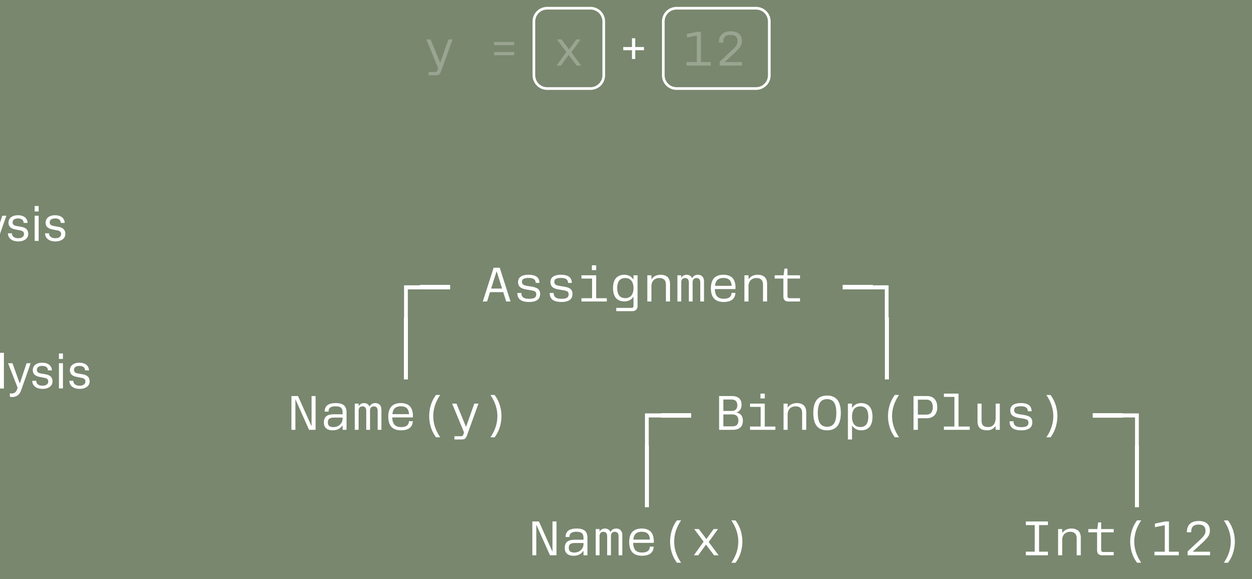
From source code to diagnostics

1. Tokenization
2. **Parsing**
3. Semantic analysis
4. Diagnostic analysis



From source code to diagnostics

1. Tokenization
2. **Parsing**
3. Semantic analysis
4. Diagnostic analysis



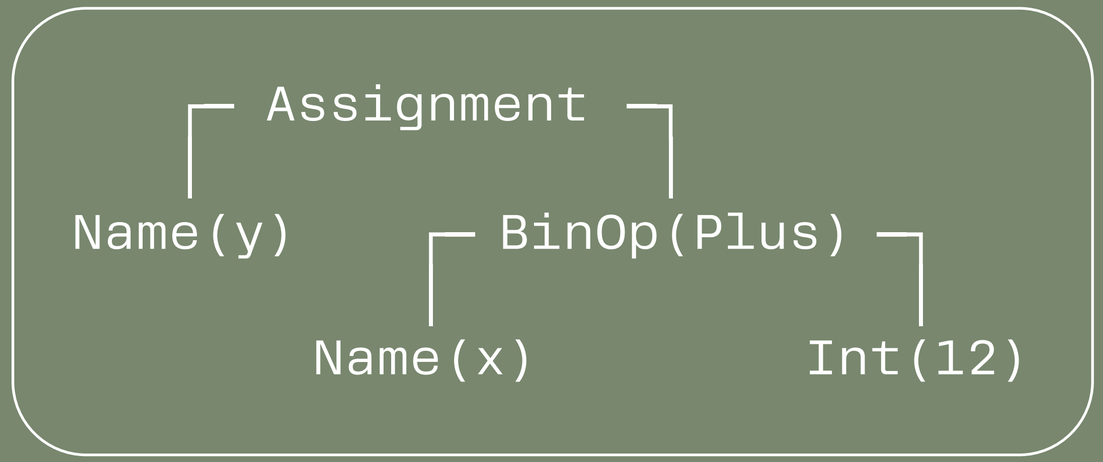
From source code to diagnostics

1. Tokenization

2. Parsing

3. Semantic analysis

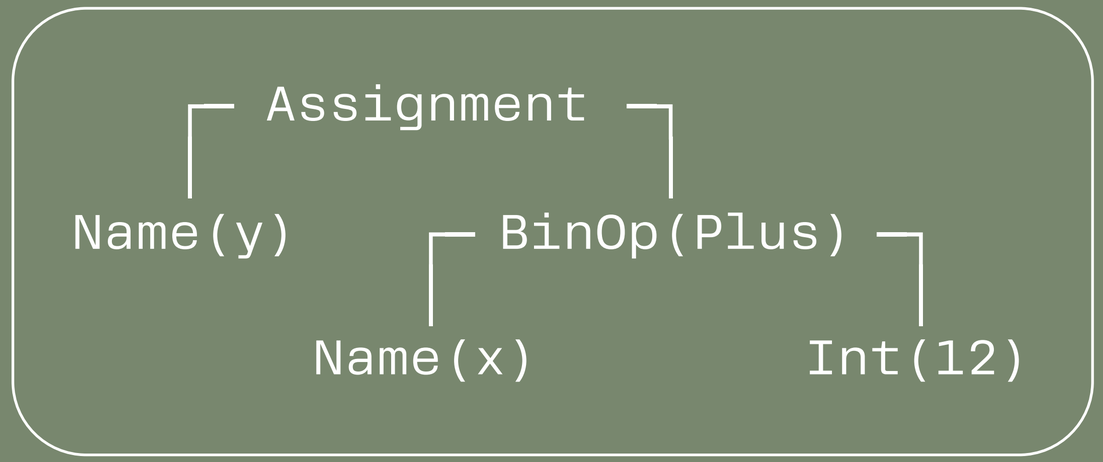
4. Diagnostic analysis



From source code to diagnostics

1. Tokenization
2. **Parsing**
3. Semantic analysis
4. Diagnostic analysis

ABSTRACT SYNTAX TREE



From source code to diagnostics

1. Tokenization
2. Parsing
3. **Semantic analysis**
4. Diagnostic analysis

From source code to diagnostics

1. Tokenization
2. Parsing
3. **Semantic analysis**
4. Diagnostic analysis

```
y = x + 12
```

From source code to diagnostics

1. Tokenization
2. Parsing
3. **Semantic analysis**
4. Diagnostic analysis

```
y = x + 12

print(y)
```

From source code to diagnostics

1. Tokenization
2. Parsing
3. **Semantic analysis**
4. Diagnostic analysis

```
y = x + 12

print(y)
```

From source code to diagnostics

1. Tokenization
2. Parsing
3. **Semantic analysis**
4. Diagnostic analysis

```
y = x + 12

print(y)
```

From source code to diagnostics

1. Tokenization
2. Parsing
3. **Semantic analysis**
4. Diagnostic analysis

```
y = x + 12

print(y)
```

From source code to diagnostics

1. Tokenization
2. Parsing
3. **Semantic analysis**
4. Diagnostic analysis

SEMANTIC MODEL

```
y = x + 12

print(y)
```

From source code to diagnostics

1. Tokenization
2. Parsing
3. Semantic analysis
4. **Diagnostic analysis**

From source code to diagnostics

1. Tokenization
2. Parsing
3. Semantic analysis
4. **Diagnostic analysis**

```
y = x + 12

print(y)
```

From source code to diagnostics

1. Tokenization
2. Parsing
3. Semantic analysis
4. **Diagnostic analysis**

```
y = x + 12

print(y)
```

From source code to diagnostics

1. Tokenization
2. Parsing
3. Semantic analysis
4. **Diagnostic analysis**

```
y = x + 12

print(y)
```

Print statements are disallowed in production.

From source code to diagnostics

1. Tokenization
2. Parsing
3. Semantic analysis
4. **Diagnostic analysis**

```
y = x + 12
print(y)
```

From source code to diagnostics

1. Tokenization
2. Parsing
3. Semantic analysis
4. **Diagnostic analysis**

```
y = x + 12
```

From source code to diagnostics

1. Tokenization
2. Parsing
3. Semantic analysis
4. Diagnostic analysis

From source code to diagnostics

1. Tokenization → **Token stream**

2. Parsing

3. Semantic analysis

4. Diagnostic analysis

From source code to diagnostics

1. Tokenization → Token stream
2. **Parsing** → **Abstract syntax tree**
3. Semantic analysis
4. Diagnostic analysis

From source code to diagnostics

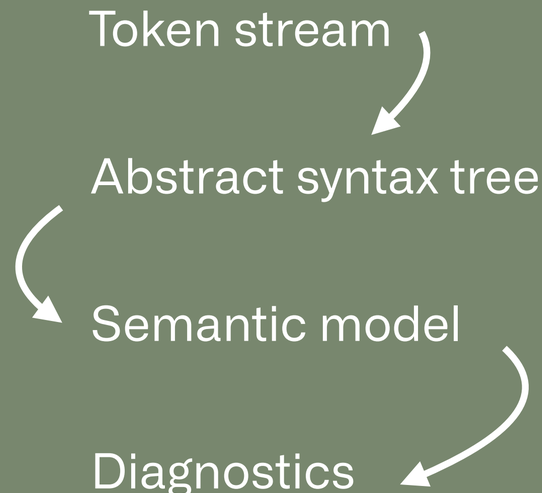
1. Tokenization	Token stream
2. Parsing	Abstract syntax tree
3. Semantic analysis →	**Semantic model**
4. Diagnostic analysis	

From source code to diagnostics

1. Tokenization	Token stream
2. Parsing	Abstract syntax tree
3. Semantic analysis	Semantic model
4. Diagnostic analysis →	**Diagnostics**

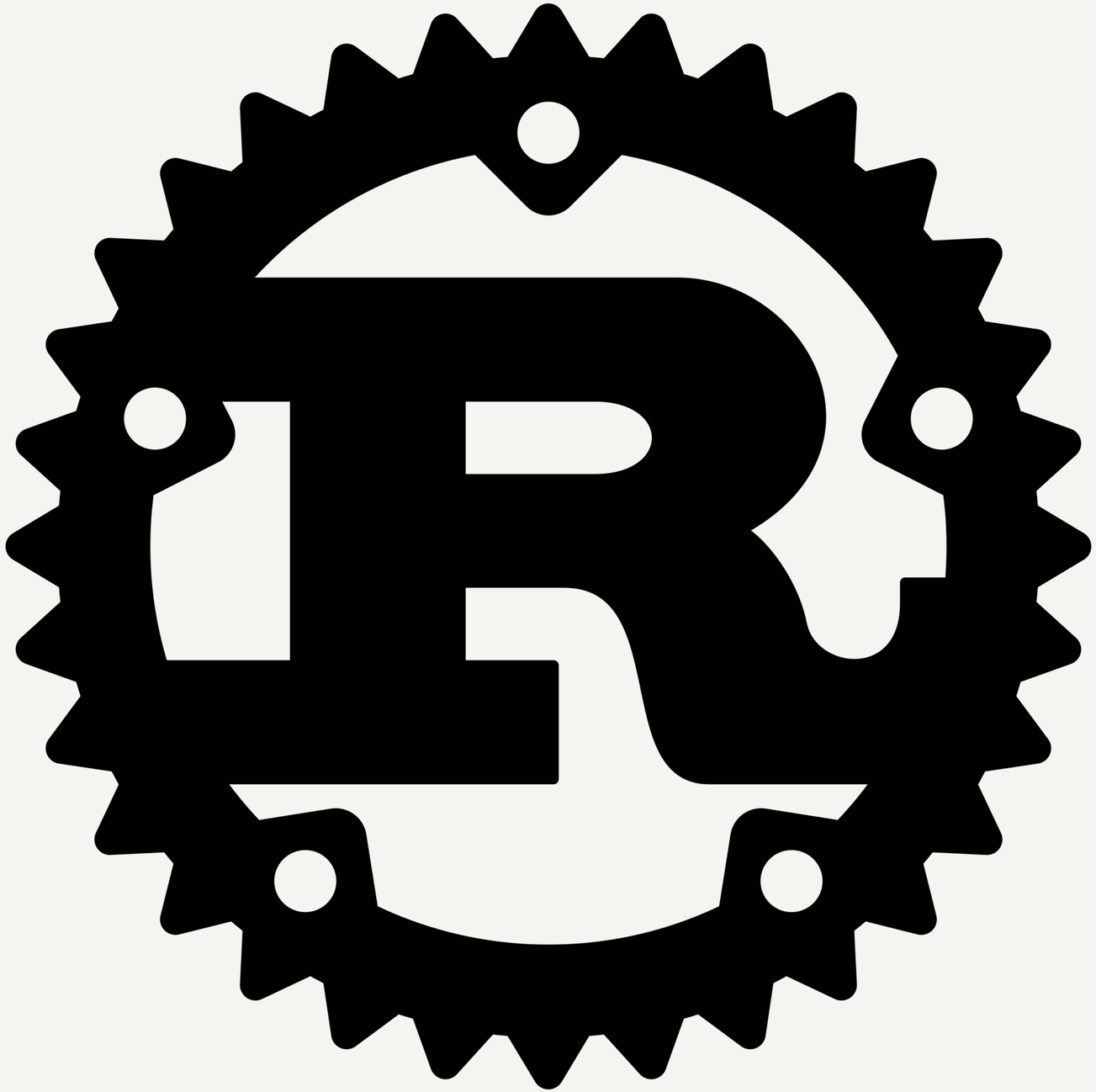
From source code to diagnostics

1. Tokenization
2. Parsing
3. Semantic analysis
4. Diagnostic analysis
5. Fix application



ASTRAL

WHAT MAKES RUFF FAST?



Rust

- Ruff is written in pure Rust (~300,000 LoC)
- But… Ruff has gotten significantly faster over time
- Rust gives you a fast baseline
- Rust gives you the tools you need to write really, really fast programs
- Rust forces you to care about the details (e.g., memory)

Deduplication

- Pyflakes, pycodestyle, pydocstyle:
  - Each tool tokenizes the source code
  - Each tool creates its own intermediate representation
- Ruff:
  - Tokenize once, parse once, analyze once
  - All rules operate off of the same intermediate representations

Concurrency and parallelism

- “Do more work at the same time”
- Rust enables ‘fearless concurrency’
- Ruff’s data model allows every file to be analyzed in parallel
- Leverage data parallelism (e.g., SIMD) where we can

Culture

- Performance is everyone’s problem
- Small wins add up
- Follow your curiosity
- Constant learning
- Ground decisions in data

SMALL-INTEGER OPTIMIZATION

Avoiding allocations in integer parsing

- Python: arbitrary-length integers
- Rust: integers at fixed sizes
- When we see a huge Python constant, what do we do?
- Stack vs. heap
- In the real-world, most integers are small!

Python

```
x = 18_446_744_073_709_551_616
```

Rust

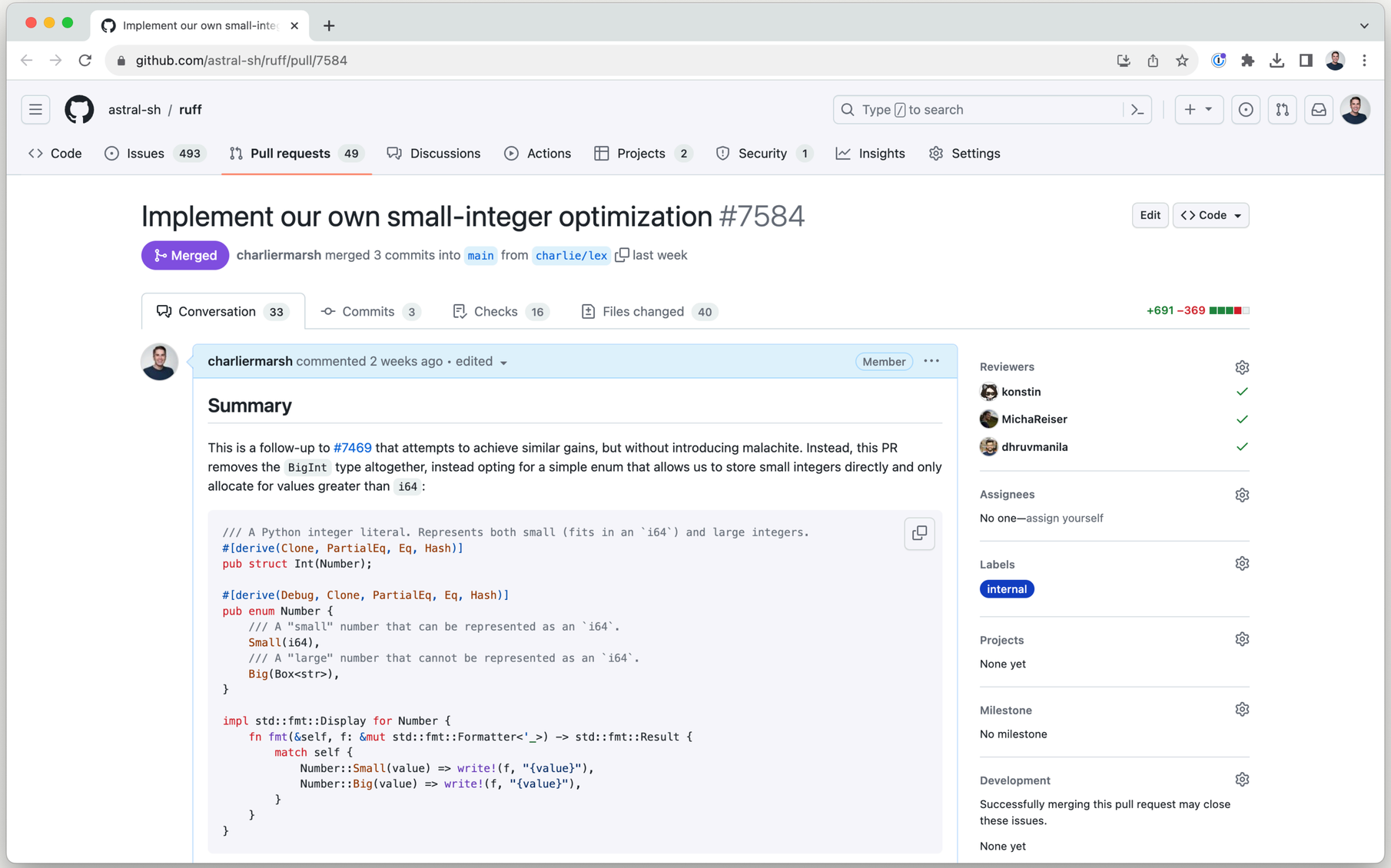
```
u16 -> [0, 65_535]

u64 -> [0, 18_446_744_073_709_551_615]
```

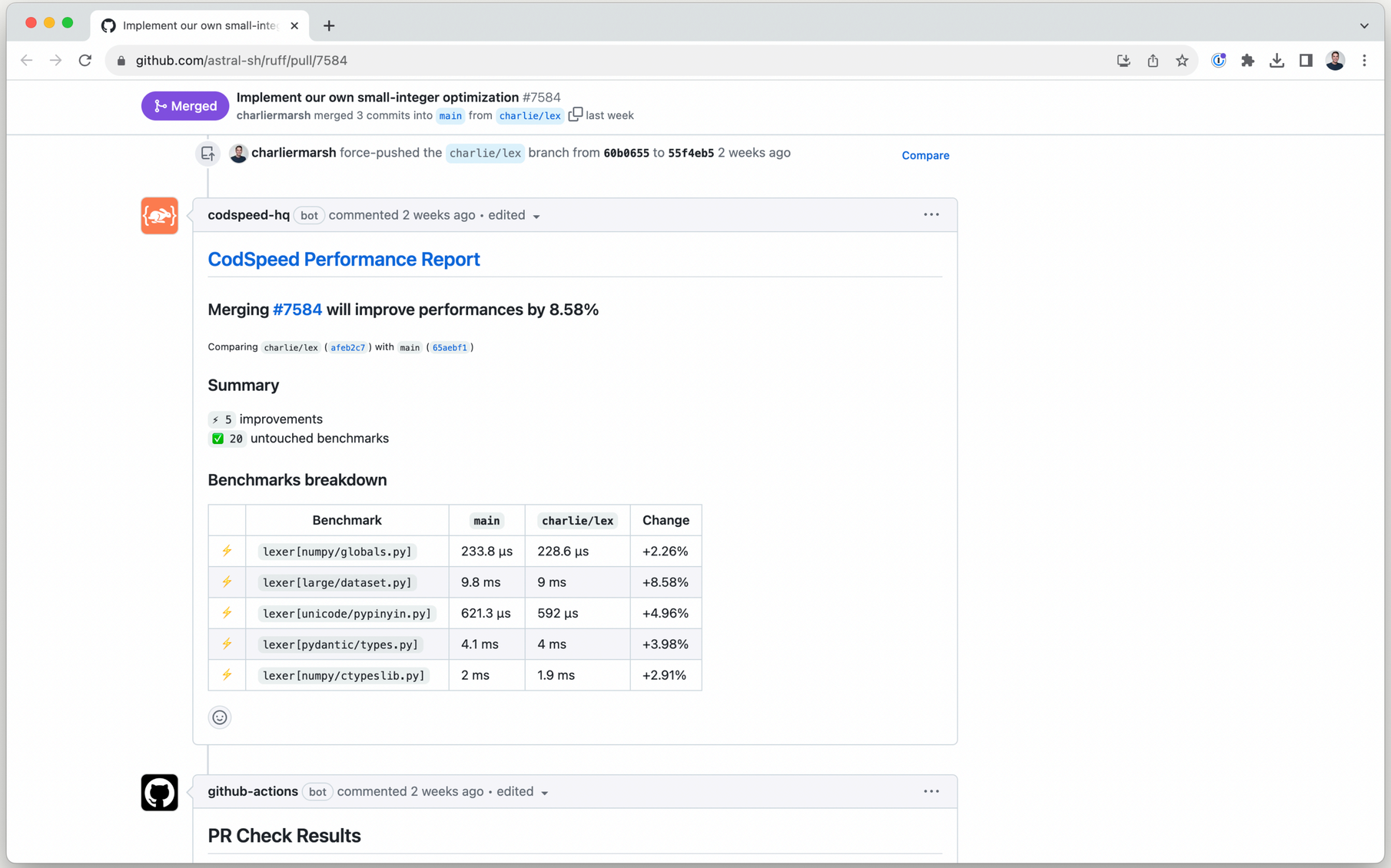
Before

```
struct Int {
    digits: Vec<u32>,
}
```

After

```
enum Int {
    Small(u64),
    Big(Vec<u32>),
}
```


Implement our own small-inte
github.com/astral-sh/ruff/pull/7584
astral-sh / ruff
Type / to search
Code
Issues 493
Pull requests 49
Discussions
Actions
Projects 2
Security 1
Insights
Settings
Implement our own small-integer optimization #7584
Edit
Code
Merged
charliermarsh merged 3 commits into main from charlie/lex last week
Conversation 33
Commits 3
Checks 16
Files changed 40
+691 −369
charliermarsh commented 2 weeks ago • edited
Member
Summary
This is a follow-up to #7469 that attempts to achieve similar gains, but without introducing malachite. Instead, this PR removes the BigInt type altogether, instead opting for a simple enum that allows us to store small integers directly and only allocate for values greater than i64:
/// A Python integer literal. Represents both small (fits in an `i64`) and large integers.
#[derive(Clone, PartialEq, Eq, Hash)]
pub struct Int(Number);
#[derive(Debug, Clone, PartialEq, Eq, Hash)]
pub enum Number {
/// A "small" number that can be represented as an `i64`.
Small(i64),
/// A "large" number that cannot be represented as an `i64`.
Big(Box<str>),
}
impl std::fmt::Display for Number {
fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
match self {
Number::Small(value) => write!(f, "{value}"),
Number::Big(value) => write!(f, "{value}"),
}
}
}
Reviewers
konstin
MichaReiser
dhruvmanila
Assignees
No one—assign yourself
Labels
internal
Projects
None yet
Milestone
No milestone
Development
Successfully merging this pull request may close these issues.
None yet


Implement our own small-inte
github.com/astral-sh/ruff/pull/7584
Merged
Implement our own small-integer optimization #7584
charliermarsh merged 3 commits into main from charlie/lex last week
charliermarsh force-pushed the charlie/lex branch from 60b0655 to 55f4eb5 2 weeks ago
Compare
codspeed-hq bot commented 2 weeks ago • edited
CodSpeed Performance Report
Merging #7584 will improve performances by 8.58%
Comparing charlie/lex (afeb2c7) with main (65aebf1)
Summary
⚡ 5 improvements
✅ 20 untouched benchmarks
Benchmarks breakdown
Benchmark | main | charlie/lex | Change
⚡ lexer[numpy/globals.py] | 233.8 µs | 228.6 µs | +2.26%
⚡ lexer[large/dataset.py] | 9.8 ms | 9 ms | +8.58%
⚡ lexer[unicode/pypinyin.py] | 621.3 µs | 592 µs | +4.96%
⚡ lexer[pydantic/types.py] | 4.1 ms | 4 ms | +3.98%
⚡ lexer[numpy/ctypeslib.py] | 2 ms | 1.9 ms | +2.91%
github-actions bot commented 2 weeks ago • edited
PR Check Results

Writing a lexer for suppression comments

- Regular expressions: powerful, but harder to optimize
- Parsing gets 3-10x faster
- Ruff itself gets 1-2% faster
- “Does this even matter?”

Python

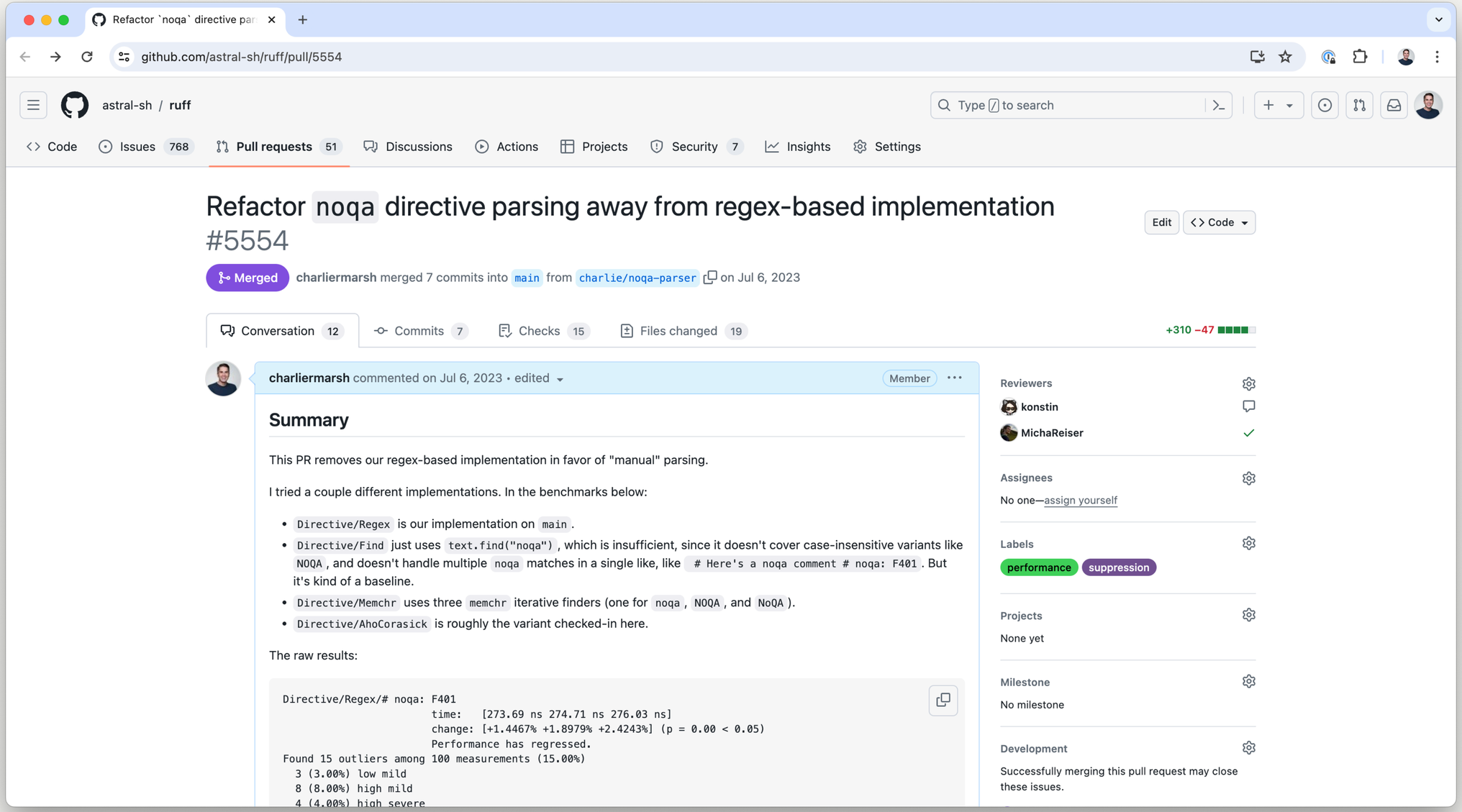
```
import os  # noqa: F401
```

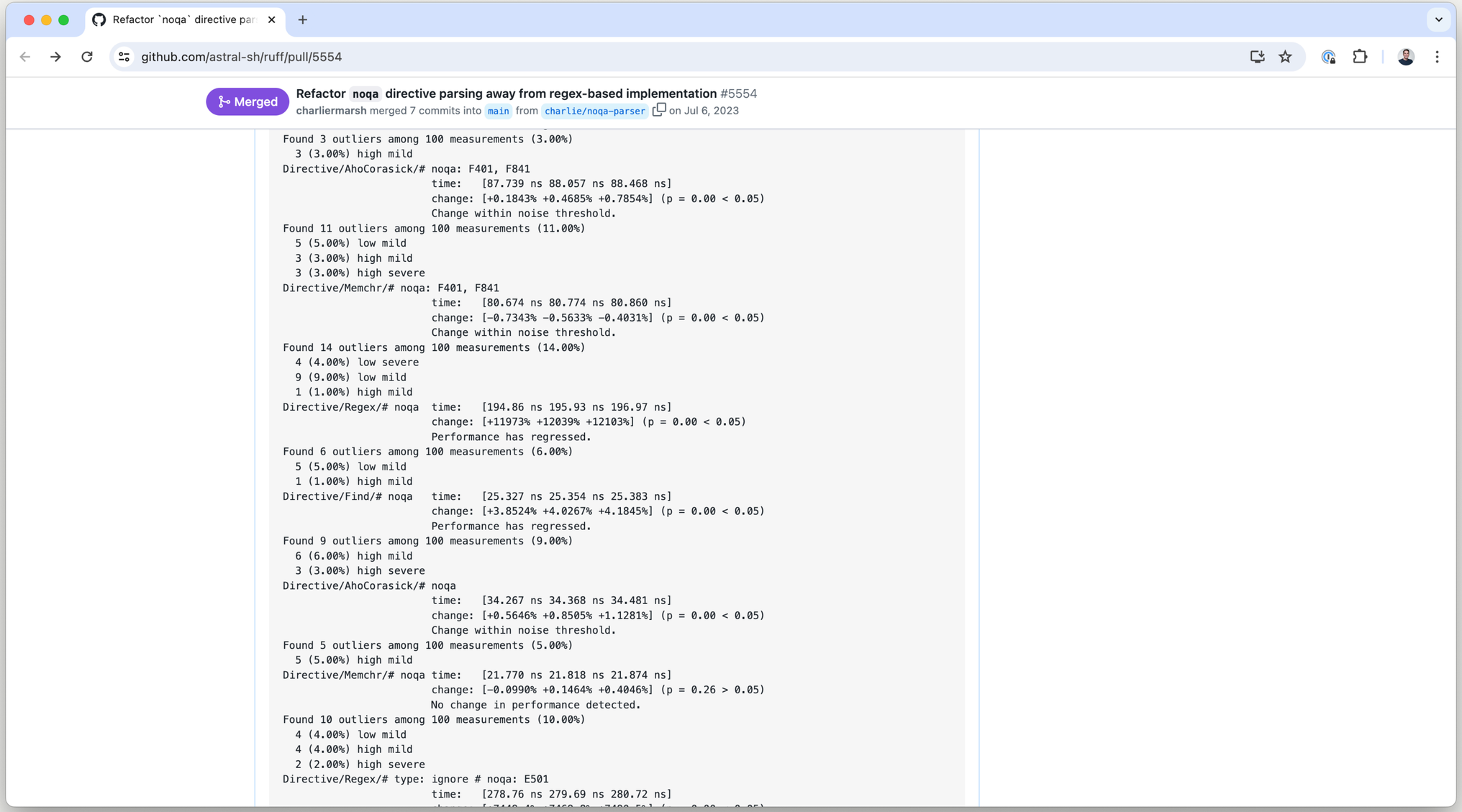
Regex

```
(?P<noqa>(?i:# noqa)(?::\s?(?P<codes>[A-Z]+[0-9]+(?:[,\s]+[A-Z]+[0-9]+)*))?)
```

Lexer

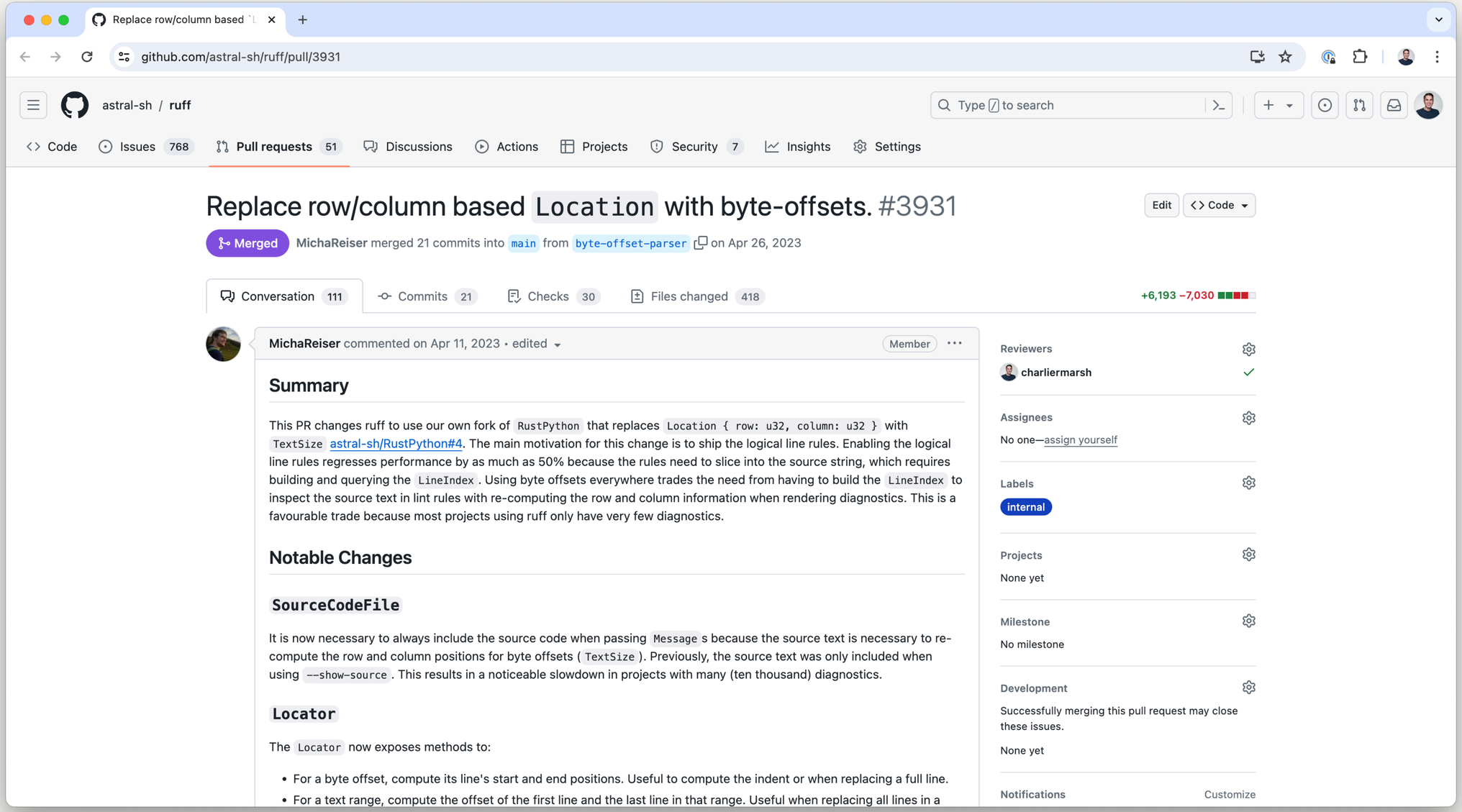
```
- Eat the hash character.
- Eat whitespace.
- Eat the `noqa` literal.
- ...
```

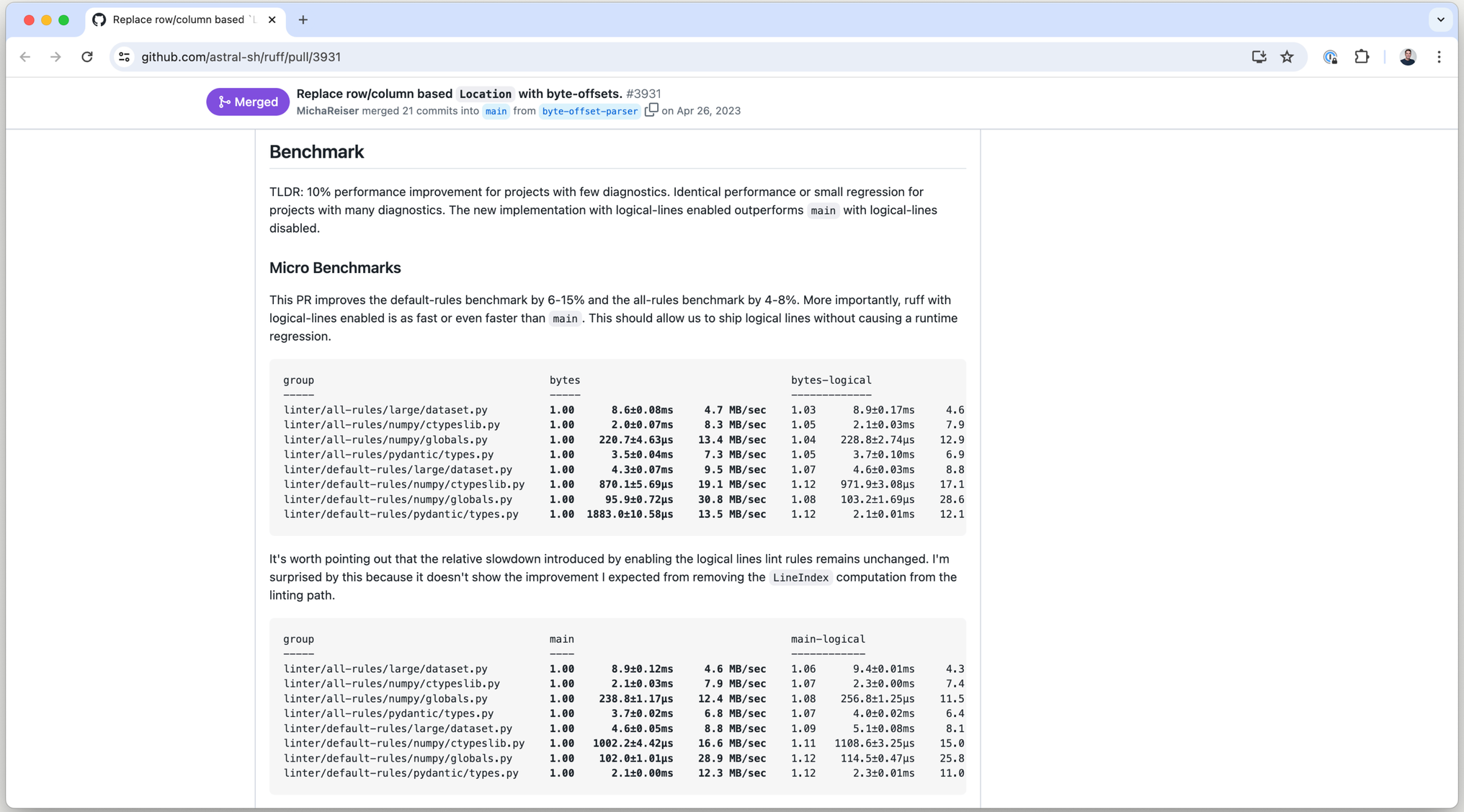

Refactor `noqa` directive par
github.com/astral-sh/ruff/pull/5554
astral-sh / ruff
Type / to search
Code
Issues 768
Pull requests 51
Discussions
Actions
Projects
Security 7
Insights
Settings
Refactor noqa directive parsing away from regex-based implementation #5554
Edit
Code
Merged
charliermarsh merged 7 commits into main from charlie/noqa-parser on Jul 6, 2023
Conversation 12
Commits 7
Checks 15
Files changed 19
+310 −47
charliermarsh commented on Jul 6, 2023 • edited
Member
Summary
This PR removes our regex-based implementation in favor of "manual" parsing.
I tried a couple different implementations. In the benchmarks below:
Directive/Regex is our implementation on main.
Directive/Find just uses text.find("noqa"), which is insufficient, since it doesn't cover case-insensitive variants like NOQA, and doesn't handle multiple noqa matches in a single like, like # Here's a noqa comment # noqa: F401. But it's kind of a baseline.
Directive/Memchr uses three memchr iterative finders (one for noqa, NOQA, and NoQA).
Directive/AhoCorasick is roughly the variant checked-in here.
The raw results:
Directive/Regex/# noqa: F401
time: [273.69 ns 274.71 ns 276.03 ns]
change: [+1.4467% +1.8979% +2.4243%] (p = 0.00 < 0.05)
Performance has regressed.
Found 15 outliers among 100 measurements (15.00%)
3 (3.00%) low mild
8 (8.00%) high mild
4 (4.00%) high severe
Reviewers
konstin
MichaReiser
Assignees
No one—assign yourself
Labels
performance
suppression
Projects
None yet
Milestone
No milestone
Development
Successfully merging this pull request may close these issues.


Refactor `noqa` directive par
github.com/astral-sh/ruff/pull/5554
Merged
Refactor noqa directive parsing away from regex-based implementation #5554
charliermarsh merged 7 commits into main from charlie/noqa-parser on Jul 6, 2023
Found 3 outliers among 100 measurements (3.00%)
3 (3.00%) high mild
Directive/AhoCorasick/# noqa: F401, F841
time: [87.739 ns 88.057 ns 88.468 ns]
change: [+0.1843% +0.4685% +0.7854%] (p = 0.00 < 0.05)
Change within noise threshold.
Found 11 outliers among 100 measurements (11.00%)
5 (5.00%) low mild
3 (3.00%) high mild
3 (3.00%) high severe
Directive/Memchr/# noqa: F401, F841
time: [80.674 ns 80.774 ns 80.860 ns]
change: [−0.7343% −0.5633% −0.4031%] (p = 0.00 < 0.05)
Change within noise threshold.
Found 14 outliers among 100 measurements (14.00%)
4 (4.00%) low severe
9 (9.00%) low mild
1 (1.00%) high mild
Directive/Regex/# noqa time: [194.86 ns 195.93 ns 196.97 ns]
change: [+11973% +12039% +12103%] (p = 0.00 < 0.05)
Performance has regressed.
Found 6 outliers among 100 measurements (6.00%)
5 (5.00%) low mild
1 (1.00%) high mild
Directive/Find/# noqa time: [25.327 ns 25.354 ns 25.383 ns]
change: [+3.8524% +4.0267% +4.1845%] (p = 0.00 < 0.05)
Performance has regressed.
Found 9 outliers among 100 measurements (9.00%)
6 (6.00%) high mild
3 (3.00%) high severe
Directive/AhoCorasick/# noqa
time: [34.267 ns 34.368 ns 34.481 ns]
change: [+0.5646% +0.8505% +1.1281%] (p = 0.00 < 0.05)
Change within noise threshold.
Found 5 outliers among 100 measurements (5.00%)
5 (5.00%) high mild
Directive/Memchr/# noqa time: [21.770 ns 21.818 ns 21.874 ns]
change: [−0.0990% +0.1464% +0.4046%] (p = 0.26 > 0.05)
No change in performance detected.
Found 10 outliers among 100 measurements (10.00%)
4 (4.00%) low mild
4 (4.00%) high mild
2 (2.00%) high severe
Directive/Regex/# type: ignore # noqa: E501
time: [278.76 ns 279.69 ns 280.72 ns]

Storing offsets instead of rows and columns

- Two ways to think about a “location” in a program: **row, column** vs. **offset**
- Initial strategy: index by row and column
- Optimized strategy: index by offset, translate to row-and-column at reporting time


Replace row/column based `L
github.com/astral-sh/ruff/pull/3931
astral-sh / ruff
Type / to search
Code
Issues 768
Pull requests 51
Discussions
Actions
Projects
Security 7
Insights
Settings
Replace row/column based Location with byte-offsets. #3931
Edit
Code
Merged
MichaReiser merged 21 commits into main from byte-offset-parser on Apr 26, 2023
Conversation 111
Commits 21
Checks 30
Files changed 418
+6,193 −7,030
MichaReiser commented on Apr 11, 2023 • edited
Member
Summary
This PR changes ruff to use our own fork of RustPython that replaces Location { row: u32, column: u32 } with TextSize astral-sh/RustPython#4. The main motivation for this change is to ship the logical line rules. Enabling the logical line rules regresses performance by as much as 50% because the rules need to slice into the source string, which requires building and querying the LineIndex. Using byte offsets everywhere trades the need from having to build the LineIndex to inspect the source text in lint rules with re-computing the row and column information when rendering diagnostics. This is a favourable trade because most projects using ruff only have very few diagnostics.
Notable Changes
SourceCodeFile
It is now necessary to always include the source code when passing Message s because the source text is necessary to re-compute the row and column positions for byte offsets (TextSize). Previously, the source text was only included when using --show-source . This results in a noticeable slowdown in projects with many (ten thousand) diagnostics.
Locator
The Locator now exposes methods to:
For a byte offset, compute its line's start and end positions. Useful to compute the indent or when replacing a full line.
For a text range, compute the offset of the first line and the last line in that range. Useful when replacing all lines in a
Reviewers
charliermarsh
Assignees
No one—assign yourself
Labels
internal
Projects
None yet
Milestone
No milestone
Development
Successfully merging this pull request may close these issues.
None yet
Notifications
Customize


Replace row/column based `L
github.com/astral-sh/ruff/pull/3931
Merged
Replace row/column based Location with byte-offsets. #3931
MichaReiser merged 21 commits into main from byte-offset-parser on Apr 26, 2023
Benchmark
TLDR: 10% performance improvement for projects with few diagnostics. Identical performance or small regression for projects with many diagnostics. The new implementation with logical-lines enabled outperforms main with logical-lines disabled.
Micro Benchmarks
This PR improves the default-rules benchmark by 6-15% and the all-rules benchmark by 4-8%. More importantly, ruff with logical-lines enabled is as fast or even faster than main. This should allow us to ship logical lines without causing a runtime regression.
group bytes bytes-logical
linter/all-rules/large/dataset.py 1.00 8.6±0.08ms 4.7 MB/sec 1.03 8.9±0.17ms 4.6
linter/all-rules/numpy/ctypeslib.py 1.00 2.0±0.07ms 8.3 MB/sec 1.05 2.1±0.03ms 7.9
linter/all-rules/numpy/globals.py 1.00 220.7±4.63µs 13.4 MB/sec 1.04 228.8±2.74µs 12.9
linter/all-rules/pydantic/types.py 1.00 3.5±0.04ms 7.3 MB/sec 1.05 3.7±0.10ms 6.9
linter/default-rules/large/dataset.py 1.00 4.3±0.07ms 9.5 MB/sec 1.07 4.6±0.03ms 8.8
linter/default-rules/numpy/ctypeslib.py 1.00 870.1±5.69µs 19.1 MB/sec 1.12 971.9±3.08µs 17.1
linter/default-rules/numpy/globals.py 1.00 95.9±0.72µs 30.8 MB/sec 1.08 103.2±1.69µs 28.6
linter/default-rules/pydantic/types.py 1.00 1883.0±10.58µs 13.5 MB/sec 1.12 2.1±0.01ms 12.1
It's worth pointing out that the relative slowdown introduced by enabling the logical lines lint rules remains unchanged. I'm surprised by this because it doesn't show the improvement I expected from removing the LineIndex computation from the linting path.
group main main-logical
linter/all-rules/large/dataset.py 1.00 8.9±0.12ms 4.6 MB/sec 1.06 9.4±0.01ms 4.3
linter/all-rules/numpy/ctypeslib.py 1.00 2.1±0.03ms 7.9 MB/sec 1.07 2.3±0.00ms 7.4
linter/all-rules/numpy/globals.py 1.00 238.8±1.17µs 12.4 MB/sec 1.08 256.8±1.25µs 11.5
linter/all-rules/pydantic/types.py 1.00 3.7±0.02ms 6.8 MB/sec 1.07 4.0±0.02ms 6.4
linter/default-rules/large/dataset.py 1.00 4.6±0.05ms 8.8 MB/sec 1.09 5.1±0.08ms 8.1
linter/default-rules/numpy/ctypeslib.py 1.00 1002.2±4.42µs 16.6 MB/sec 1.11 1108.6±3.25µs 15.0
linter/default-rules/numpy/globals.py 1.00 102.0±1.01µs 28.9 MB/sec 1.12 114.5±0.47µs 25.8
linter/default-rules/pydantic/types.py 1.00 2.1±0.00ms 12.3 MB/sec 1.12 2.3±0.01ms 11.0

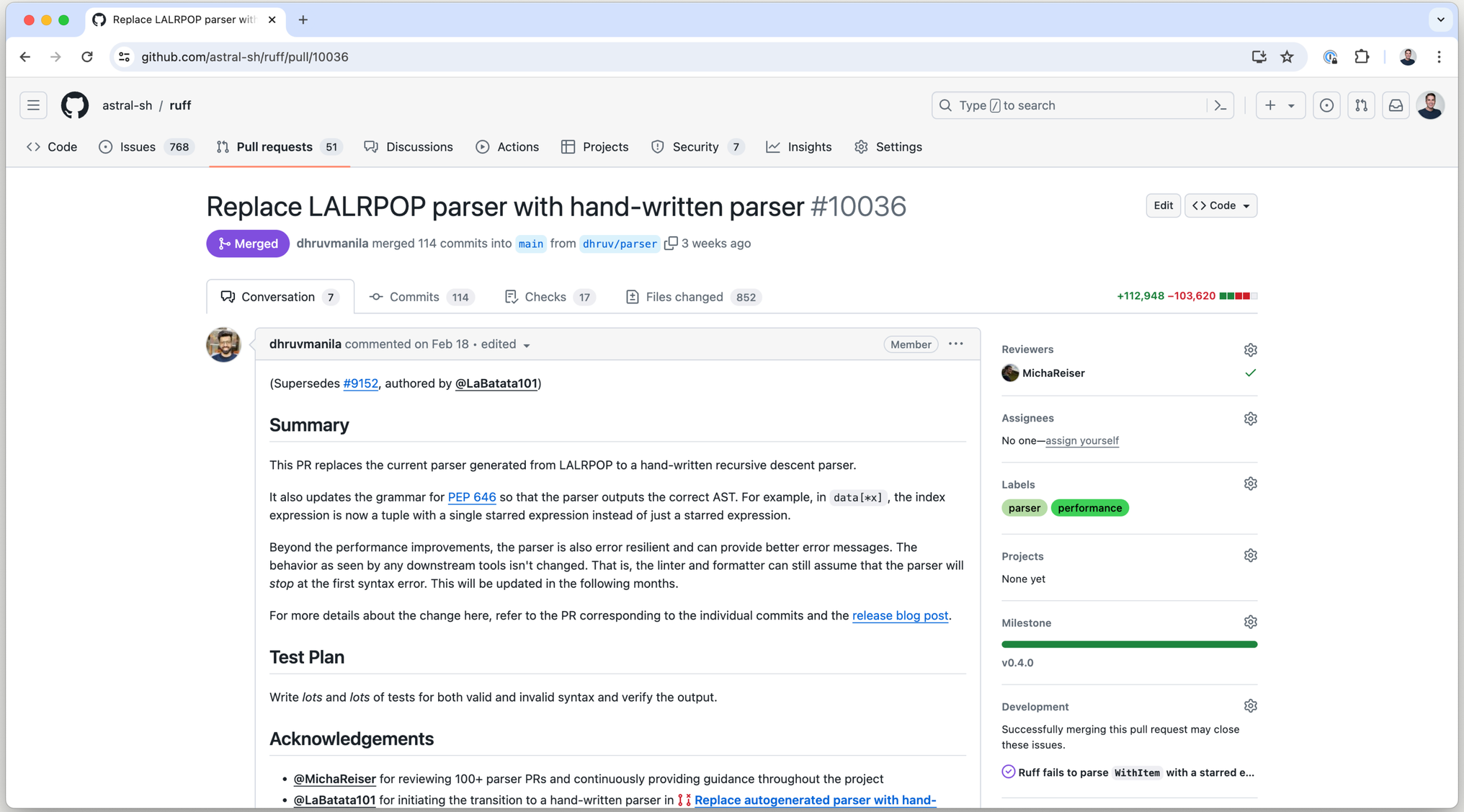
Migrating away from a parser generator

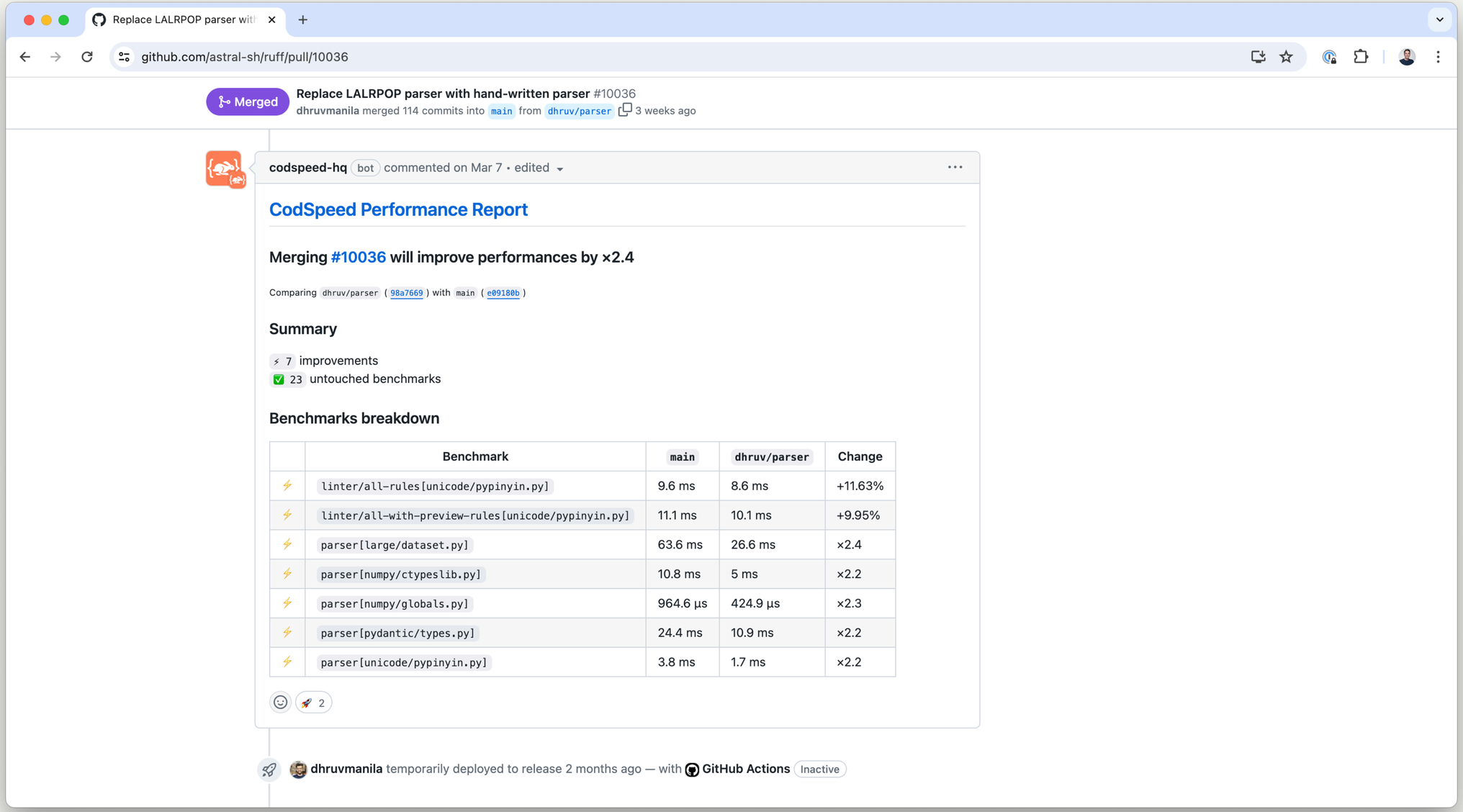
```
GlobalStatement: ast::Stmt = {
  "global" <names:OneOrMore<Identifier>> => {
    ast::Stmt::Global(
      ast::StmtGlobal { names }
    )
  },
};
```

```rust
fn parse_global_statement(&mut self) -> ast::Global {
    let start = self.node_start();
    self.bump(TokenKind::Global);

    let names = self.parse_comma_separated(
        RecoveryContextKind::Identifiers,
        Parser::parse_identifier,
    );

    ast::Global {
        range: self.node_range(start),
        names,
    }
}
```

Migrating away from a parser generator

- Why?
  1. Parsing and representing invalid programs
  2. Ironically, easier to maintain as Python expands its grammar
  3. Significant performance boost
- Performance doesn't have to be a tradeoff
- Pick your battles (for Ruff, a parser is "worth it")


Replace LALRPOP parser wit
github.com/astral-sh/ruff/pull/10036
astral-sh / ruff
Type / to search
Code
Issues 768
Pull requests 51
Discussions
Actions
Projects
Security 7
Insights
Settings
Replace LALRPOP parser with hand-written parser #10036
Edit
Code
Merged
dhruvmanila merged 114 commits into main from dhruv/parser 3 weeks ago
Conversation 7
Commits 114
Checks 17
Files changed 852
+112,948 −103,620
dhruvmanila commented on Feb 18 • edited
Member
(Supersedes #9152, authored by @LaBatata101)
Summary
This PR replaces the current parser generated from LALRPOP to a hand-written recursive descent parser.
It also updates the grammar for PEP 646 so that the parser outputs the correct AST. For example, in data[*x], the index expression is now a tuple with a single starred expression instead of just a starred expression.
Beyond the performance improvements, the parser is also error resilient and can provide better error messages. The behavior as seen by any downstream tools isn't changed. That is, the linter and formatter can still assume that the parser will stop at the first syntax error. This will be updated in the following months.
For more details about the change here, refer to the PR corresponding to the individual commits and the release blog post.
Test Plan
Write lots and lots of tests for both valid and invalid syntax and verify the output.
Acknowledgements
@MichaReiser for reviewing 100+ parser PRs and continuously providing guidance throughout the project
@LaBatata101 for initiating the transition to a hand-written parser in Replace autogenerated parser with hand-
Reviewers
MichaReiser
Assignees
No one—assign yourself
Labels
parser
performance
Projects
None yet
Milestone
v0.4.0
Development
Successfully merging this pull request may close these issues.
Ruff fails to parse WithItem with a starred e...


Replace LALRPOP parser wit
github.com/astral-sh/ruff/pull/10036
Merged
Replace LALRPOP parser with hand-written parser #10036
dhruvmanila merged 114 commits into main from dhruv/parser 3 weeks ago
codspeed-hq bot commented on Mar 7 • edited
CodSpeed Performance Report
Merging #10036 will improve performances by ×2.4
Comparing dhruv/parser (98a7669) with main (e09180b)
Summary
⚡ 7 improvements
✅ 23 untouched benchmarks
Benchmarks breakdown
Benchmark | main | dhruv/parser | Change
⚡ linter/all-rules[unicode/pypinyin.py] | 9.6 ms | 8.6 ms | +11.63%
⚡ linter/all-with-preview-rules[unicode/pypinyin.py] | 11.1 ms | 10.1 ms | +9.95%
⚡ parser[large/dataset.py] | 63.6 ms | 26.6 ms | ×2.4
⚡ parser[numpy/ctypeslib.py] | 10.8 ms | 5 ms | ×2.2
⚡ parser[numpy/globals.py] | 964.6 µs | 424.9 µs | ×2.3
⚡ parser[pydantic/types.py] | 24.4 ms | 10.9 ms | ×2.2
⚡ parser[unicode/pypinyin.py] | 3.8 ms | 1.7 ms | ×2.2
2
dhruvmanila temporarily deployed to release 2 months ago — with GitHub Actions Inactive

Building fast tools

- Rust enables us to build different kinds of programs
- But many of the principles behind “building fast programs” can apply anywhere (tooling, culture, thinking in data, etc.)
- Sometimes, performance doesn’t matter
- Sometimes, it’s worth caring about and working on performance, even when it “doesn’t matter”
- You can learn this stuff

WHAT MAKES RUFF FAST?

- Fast-path heuristics for comment detection (9808)
- Small-integer optimization (7584)
- Byte offsets (3931)
- Trie-based settings router (11111)
- SIMD string parsing (8227)
- Reduce enum size (9900)
- Use matches for static comparisons (5099)
- Set cold attributes (10121)
- Pre-allocate buffer based on estimated size (6612)
- Cache semantic model lookups (6047)
- Box string slices (9855)
- Migrate to hand-written parser (10036)
- SIMD string flexing (9888)
- Flatten chained iterators (9867)
- Lazily compute line length offset (5811)
- Compile regex once (77)
- Avoid allocations in line-length check (95)
- Use custom allocator (2768)
- Skip string normalization (10116)
- Avoid large stack allocations in rule table (10971)

Not all fast software is world-class, but all world-class software is fast.

TOBI LUTKE

Shopify CEO

ASTRAL

NEXT-GEN PYTHON TOOLING