Coccinelle For Rust

a Tutorial!

Tathagata Roy
Julia Lawall

What is Coccinelle For Rust

Coccinelle is a tool for automatic program matching and transformation that was originally developed for making large scale changes to the Linux kernel source code (ie, C code). Coccinelle for Rust is Coccinelle, but for Rust.

Uses. How does it make our life easier

1. Search for semantic patterns
2. Perform repeated changes
3. Can be used to enforce patterns and restrictions
4. Run analysis using scripting

How to install CFR

1. Clone the repository at https://gitlab.inria.fr/coccinelle/coccinelleforrust/-/tree/kangrejos25
2. Build and install using `cargo install –path ./`
3. If you have trouble please let us know!

https://tinyurl.com/2vyh7hpa

a.extend(b.into_iter()) → syntax mansplaining

commit 66adc8e17940fcd71a2435ca108960abc698cc40 (HEAD)
Author: Bryan Gurney <bgurney@redhat.com>
Date: Thu Aug 24 15:28:47 2023 -0400

    Remove useless .into_inter() calls

    Signed-off-by: Bryan Gurney <bgurney@redhat.com>

a.extend(b.into_iter()) → syntax mansplaining

```
diff --git a/src/dbus_api/api/shared.rs b/src/dbus_api/api/shared.rs
index 290d29db..825e1b3d 100644
--- a/src/dbus_api/api/shared.rs
+++ b/src/dbus_api/api/shared.rs
@@ -117,7 +117,7 @@ pub fn get_managed_objects_method(
                 })
             })
             .fold(HashMap::new(), |mut props, prop| {
-                props.extend(prop.into_iter());
+                props.extend(prop);
                 props
             });
```

a.extend(b.into_iter()) → syntax mansplaining

```
-               props.extend(prop.into_iter());
+               props.extend(prop);
```

a.extend(b.into_iter()) → syntax mansplaining

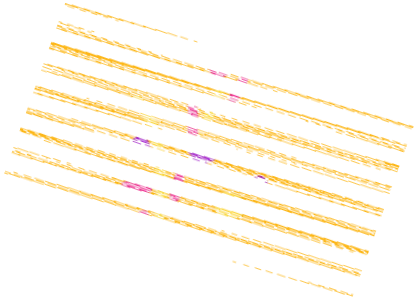
```
-               expr1.extend(expr2.into_iter());
+               expr1.extend(expr2);
```

Exactly how a semantic patch looks!

a.extend(b.into_iter()) → syntax mansplaining

```
@@
expression x, y;
@@

-x.extend(y.into_iter())
+x.extend(y)
```



Congrats!

Our first semantic patch!

a.extend(b.into_iter()) → syntax mansplaining

Try it yourself!

```
cd examples
make eg1   # checks out the specific commit

<edit eg1.cocci>

cfr -c eg1.cocci stratisd

# you can also cheat and look at eg1_solution.cocci!
```

a.extend(b.into_iter()) → syntax mansplaining

We found a few cases that the commit missed!

If you do a

`git show`

And compare it to the output of cfr you see a few differences.

a.extend(b.into_iter()) → syntax mansplaining

BUT HEY! THIS CAN BE DONE WITH A REGEX!!!

a.extend(b.into_iter()) → syntax mansplaining

BUT HEY! THIS CAN BE DONE WITH A REGEX!!!

(\w+)\.extend\((\w+)\.into_iter\(\)\);

Putting it in a box causes certainty. RIP Schrödinger.

commit 04ca6c70026d1fa16f0a29d7fea0a5bf63e76f2a (HEAD)
Author: Bryan Gurney <bgurney@redhat.com>
Date: Thu Jan 9 12:08:15 2025 -0500

    Stop using local array for test_grow_physical write_block

    Signed-off-by: Bryan Gurney <bgurney@redhat.com>

Putting it in a box causes certainty. RIP Schrödinger.

```
--- a/src/engine/strat_engine/pool/v1.rs
+++ b/src/engine/strat_engine/pool/v1.rs
@@ -1802,7 +1802,7 @@ mod tests {
             .tempdir()
             .unwrap();
         let new_file = tmp_dir.path().join("stratis_test.txt");
-        let write_block = &[0; 512_000];
+        let write_block = vec![0; 512_000].into_boxed_slice();

         {
             let (_, fs) = pool.get_filesystem(fs_uuid).unwrap();
```

Putting it in a box causes certainty. RIP Schrödinger.

```
@@ -1823,7 +1823,7 @@ mod tests {
             .open(new_file)
             .unwrap();
         while !pool.out_of_alloc_space() {
-            f.write_all(write_block).unwrap();
+            f.write_all(&write_block).unwrap();
             f.sync_all().unwrap();
             match pool {
                 AnyPool::V1(p) => p.event_on(pool_uuid, &pool_name).unwrap(),
```

Putting it in a box causes certainty. RIP Schrödinger.

```
-       let write_block = &[0; 512_000];
+       let write_block = vec![0; 512_000].into_boxed_slice();


==========


-           f.write_all(write_block).unwrap();
+           f.write_all(&write_block).unwrap();
```

Putting it in a box causes certainty. RIP Schrödinger.

```
@find@
expression y1, y2;
identifier i;
@@

- let i = &[y1; y2];
+let i = vec![y1; y2].into_boxed_slice();
```

```
@replace@
identifier find.i;
@@

- f.write_all(i).unwrap()
+f.write_all(&i).unwrap()
```

```
cfr -c eg2.cocci stratisd
```

Putting it in a box causes certainty. RIP Schrödinger.

Try it yourself!

```
cd examples
make eg2   # checks out the specific commit

<edit eg2.cocci>

cfr -c eg2.cocci stratisd
```

Putting it in a box causes certainty. RIP Schrödinger.

```
-           let buf = &[1u8; SECTOR_SIZE];

+           let buf = vec![y1; y2].into_boxed_slice();
```

Putting it in a box causes certainty. RIP Schrödinger.

```
-           let buf = &[1u8; SECTOR_SIZE];

+           let buf = vec![y1; y2].into_boxed_slice();
```

```
let buf = vec![1u8; SECTOR_SIZE].into_boxed_slice();
```

Putting it in a box causes certainty. RIP Schrödinger.

MACROS!!!

Putting it in a box causes certainty. RIP Schrödinger.

We again find a few examples
that was missed in the commit!

Came for the safety, stayed for the shiny format strings

rustc branch master

```
debug!(
    "report_borrow_conflicts_with_destructor(\
        {:?}, {:?}, ({:?}, {:?}), {:?}\
        )",
        location, borrow, place, drop_span, kind,
    );
```

- Hard to read.
- Looks ugly.
- Adding a new parameter in the right position is cumbersome

Came for the safety, stayed for the shiny format strings

```
debug ! ( "report_borrow_conflicts_with_destructor({location:?}, {borrow:?},({place:?}, {drop_span:?}), {kind:?} )" ) ;
```

- Prettier
- Head hurts less trying to figure out the position of an argument
- Life has more meaning
- Can add/modify arguments easily

Came for the safety, stayed for the shiny format strings

```
@r@
expression e, fname;
expression_list es1;
@@

fname!(e, es1);
```

```
@script:python r2@
assimilated;
e << r.e;
x << r.es1;
@@
global to_eprintln

parts = [e]+x

try:
        assimilated = to_eprintln(parts,
unchanged=None)
        if assimilated == None:
                include_match=False
except Exception as e:
        print("EXCEPTION:", e)
        include_match=False
```

External python file

```
@r3@
expression r2.assimilated;
expression r.e, r.es1,
r.fname;
@@

-fname!(e, es1)
+fname!(assimilated)
```

On *rust/compiler* this semantic patch changes **332** files.

Came for the safety, stayed for the shiny format strings

Run the code on your own database! (please make a commit before hand)

Command -

```
cfr -c eg3.cocci yourproject/src –import-pyfile fmt.py
```

To make the changes, just add –apply at the end of the command.

Note: You would probably like to run `cargo fmt` because cfr isn't the best at handling macro formatting yet.

Work in progress and the Future

1. BETTER MACRO SUPPORT
    a. Try to work with rustc-expand crate
2. Better Type Inference
    a. Parallel execution is blocked when type inference is used due to a non-thread safe type in RA interface
3. Rule dependencies
4. Better script communication
    a. Being able to pass ASTs back and from python
    b. Currently we can only pass strings and expressions

LINKS!

Website - https://rust-for-linux.com/coccinelle-for-rust

Gitlab - https://gitlab.inria.fr/coccinelle/coccinelleforrust (Main branch)

These slides - coccinelleforrust/talks/kangrejos25.pdf

Contact - t.roy@student.vu.nl

julia.lawall@inria.fr

Thank you!
Questions?