Coccinelle for Rust
https://gitlab.inria.fr/coccinelle/coccinelleforrust.git

Julia Lawall, Tathagata Roy

September 17, 2023

Goals

- Perform repetitive transformations at a large scale.

Goals

- Perform repetitive transformations at a large scale.
  - Rust is 1.6 MLOC.
  - The Linux kernel is 23 MLOC.

Goals

- Perform repetitive transformations at a large scale.
  - Rust is 1.6 MLOC.
  - The Linux kernel is 23 MLOC.
  - Collateral evolutions: a change in an API requires changes in all clients.

Goals

- Perform repetitive transformations at a large scale.
  - Rust is 1.6 MLOC.
  - The Linux kernel is 23 MLOC.
  - Collateral evolutions: a change in an API requires changes in all clients.
- Provide a transformation language that builds on developer expertise.

Goals

- Perform repetitive transformations at a large scale.
  - Rust is 1.6 MLOC.
  - The Linux kernel is 23 MLOC.
  - Collateral evolutions: a change in an API requires changes in all clients.
- Provide a transformation language that builds on developer expertise.
- Changes + developer familiarity = (semantic) patches

An example change (Rust repository)

```
commit d822b97a27e50f5a091d2918f6ff0ffd2d2827f5
Author: Kyle Matsuda <kyle.yoshio.matsuda@gmail.com>
Date:   Mon Feb 6 17:48:12 2023 -0700

    change usages of type_of to bound_type_of

diff --git a/compiler/rustc_borrowck/src/diagnostics/conflict_errors.rs b/compiler/.../conflict_errors.rs
@@ -2592,4 +2592,4 @@ fn annotate_argument_and_return_for_borrow(
             } else {
-                let ty = self.infcx.tcx.type_of(self.mir_def_id());
+                let ty = self.infcx.tcx.bound_type_of(self.mir_def_id()).subst_identity();
                 match ty.kind() {
                     ty::FnDef(_, _) | ty::FnPtr(_) => self.annotate_fn_sig(
diff --git a/compiler/rustc_borrowck/src/diagnostics/mod.rs b/compiler/.../mod.rs
@@ -1185,4 +1185,4 @@ fn explain_captures(
                         matches!(tcx.def_kind(parent_did), rustc_hir::def::DefKind::Impl { .. })
                             .then_some(parent_did)
-                            .and_then(|did| match tcx.type_of(did).kind() {
+                            .and_then(|did| match tcx.bound_type_of(did).subst_identity().kind() {
                                 ty::Adt(def, ..) => Some(def.did()),
...
```

136 files changed, 385 insertions(+), 262 deletions(-)

An example change (Rust repository)

```
commit d822b97a27e50f5a091d2918f6ff0ffd2d2827f5
Author: Kyle Matsuda <kyle.yoshio.matsuda@gmail.com>
Date:   Mon Feb 6 17:48:12 2023 -0700

    change usages of type_of to bound_type_of

diff --git a/compiler/rustc_borrowck/src/diagnostics/conflict_errors.rs b/compiler/.../conflict_errors.rs
@@ -2592,4 +2592,4 @@ fn annotate_argument_and_return_for_borrow(
             } else {
-                let ty = self.infcx.tcx.type_of(self.mir_def_id());
+                let ty = self.infcx.tcx.bound_type_of(self.mir_def_id()).subst_identity();
                 match ty.kind() {
                     ty::FnDef(_, _) | ty::FnPtr(_) => self.annotate_fn_sig(
diff --git a/compiler/rustc_borrowck/src/diagnostics/mod.rs b/compiler/.../mod.rs
@@ -1185,4 +1185,4 @@ fn explain_captures(
                         matches!(tcx.def_kind(parent_did), rustc_hir::def::DefKind::Impl { .. })
                             .then_some(parent_did)
-                            .and_then(|did| match tcx.type_of(did).kind() {
+                            .and_then(|did| match tcx.bound_type_of(did).subst_identity().kind() {
                                 ty::Adt(def, ..) => Some(def.did()),
...
```

136 files changed, 385 insertions(+), 262 deletions(-)

Creating a semantic patch: Step 1: remove irrelevant code

```
-                 let ty = self.infcx.tcx.type_of(self.mir_def_id())
+                 let ty = self.infcx.tcx.bound_type_of(self.mir_def_id()).subst_identity()
```

```
-                     .and_then(|did| match tcx.type_of(did).kind() {
+                     .and_then(|did| match tcx.bound_type_of(did).subst_identity().kind() {
```

Creating a semantic patch: Step 2: pick a typical example

```
@@

@@

- self.infcx.tcx.type_of(self.mir_def_id())
+ self.infcx.tcx.bound_type_of(self.mir_def_id()).subst_identity()
```

Creating a semantic patch: Step 3: abstract over subterms using metavariables

```
@@
expression tcx, arg;
@@

- tcx.type_of(arg)
+ tcx.bound_type_of(arg).subst_identity()
```

Creating a semantic patch: Step 3: abstract over subterms using metavariables

```
@@
expression tcx, arg;
@@

- tcx.type_of(arg)
+ tcx.bound_type_of(arg).subst_identity()
```

Updates over 200 call sites.

An outlier

```
            let (shim_size, shim_align, _kind) = ecx.get_alloc_info(alloc_id);
+           let def_ty = ecx.tcx.bound_type_of(def_id).subst_identity();
            let extern_decl_layout =
-               ecx.tcx.layout_of(ty::ParamEnv::empty().and(ecx.tcx.type_of(def_id))).unwrap();
+               ecx.tcx.layout_of(ty::ParamEnv::empty().and(def_ty)).unwrap();
            if extern_decl_layout.size != shim_size || extern_decl_layout.align.abi != shim_align {
                throw_unsup_format!(
                    "`extern` static `{name}` from crate `{krate}` has been declared \
```

An outlier

```
            let (shim_size, shim_align, _kind) = ecx.get_alloc_info(alloc_id);
+           let def_ty = ecx.tcx.bound_type_of(def_id).subst_identity();
            let extern_decl_layout =
-               ecx.tcx.layout_of(ty::ParamEnv::empty().and(ecx.tcx.type_of(def_id))).unwrap();
+               ecx.tcx.layout_of(ty::ParamEnv::empty().and(def_ty)).unwrap();
            if extern_decl_layout.size != shim_size || extern_decl_layout.align.abi != shim_align {
                throw_unsup_format!(
                    "`extern` static `{name}` from crate `{krate}` has been declared \
```

The developer has created a new name to avoid a long line.

- Could address it manually.
- Could create a rule for the special case of nested function call contexts (probably not worth it for one case).

An alternate semantic patch

```
@@
expression tcx, arg;
@@


  tcx.
-     type_of(arg)
+     bound_type_of(arg).subst_identity()
```

Putting tcx in the context ensures any comments will be preserved.

A refinement

```
@@
TyCtxt tcx;
expression arg;
@@

  tcx.
-    type_of(arg)
+    bound_type_of(arg).subst_identity()
```

Specifying the type of `tcx` protects against changing other uses of `type_of`.

A refinement

```
@@
TyCtxt tcx;
expression arg;
@@

  tcx.
-    type_of(arg)
+    bound_type_of(arg).subst_identity()
```

Specifying the type of `tcx` protects against changing other uses of `type_of`.

Alternative specifications?:

- `(.*::)*TyCtxt tcx;`
- `*::TyCtxt tcx;`
- `struct TyCtxt tcx;`

An example: multiple cases

```
commit 298ae8c721102c36243335653e57a7f94e08f94a
Author: Michael Goulet <michael@errs.io>
Date:   Wed Feb 22 22:23:10 2023 +0000

    Rename ty_error_with_guaranteed to ty_error, ty_error to ty_error_misc

diff --git a/compiler/rustc_borrowck/src/region_infer/opaque_types.rs b/compiler/.../opaque_types.rs
@@ -156,3 +156,3 @@ pub(crate) fn infer_opaque_types(
                     });
-                    prev.ty = infcx.tcx.ty_error_with_guaranteed(guar);
+                    prev.ty = infcx.tcx.ty_error(guar);
                 }
@@ -248,3 +248,3 @@ fn infer_opaque_definition_from_instantiation(
         if let Some(e) = self.tainted_by_errors() {
-            return self.tcx.ty_error_with_guaranteed(e);
+            return self.tcx.ty_error(e);
         }
...
diff --git a/compiler/rustc_hir_analysis/src/astconv/mod.rs b/compiler/rustc_hir_analysis/src/astconv/mod.rs
@@ -429,2 +429,2 @@ fn provided_kind(
                         self.inferred_params.push(ty.span);
-                        tcx.ty_error().into()
+                        tcx.ty_error_misc().into()
```

32 files changed, 121 insertions(+), 140 deletions(-)

An example: multiple cases

Two changes:

- From `ty_error_with_guaranteed` to `ty_error` (1 argument)
- From `ty_error` to `ty_error_misc` (no arguments)

```
@@
expression tcx, arg;
@@
- tcx.ty_error_with_guaranteed(arg)
+ tcx.ty_error(arg)

@@
expression tcx, arg;
@@
- tcx.ty_error()
+ tcx.ty_error_misc()
```

An example: searching for variants

```
commit f3f9d6dfd92dfaeb14df891ad27b2531809dd734
Author: Eduard-Mihai Burtescu <edy.burt@gmail.com>
Date:   Fri Jun 14 00:48:52 2019 +0300

    Unify all uses of 'gcx and 'tcx.

diff --git a/src/librustc/infer/error_reporting/mod.rs b/src/librustc/infer/error_reporting/mod.rs
@@ -460,6 +460,6 @@ impl<'gcx, 'tcx> Printer<'gcx, 'tcx> for AbsolutePathPrinter<'gcx, 'tcx> {
             type DynExistential = !;
             type Const = !;

-            fn tcx<'a>(&'a self) -> TyCtxt<'gcx, 'tcx> {
+            fn tcx<'a>(&'a self) -> TyCtxt<'tcx> {
                 self.tcx
             }
@@ -1977,4 +1976,4 @@ pub fn enter_global<'gcx, F, R>(gcx: &'gcx GlobalCtxt<'gcx>, f: F) -> R
     pub unsafe fn with_global<F, R>(f: F) -> R
     where
-        F: for<'gcx, 'tcx> FnOnce(TyCtxt<'gcx, 'tcx>) -> R,
+        F: for<'tcx> FnOnce(TyCtxt<'tcx>) -> R,
     {
```

341 files changed, 3109 insertions(+), 3327 deletions(-)

An example: searching for variants

A first attempt:

```
@type@
@@
- TyCtxt<'gcx, 'tcx>
+ TyCtxt<'tcx>
```

An example: searching for variants

A first attempt:

```
@type@
@@
- TyCtxt<'gcx, 'tcx>
+ TyCtxt<'tcx>
```

This does part of the work, but some change sites are overlooked:

- `DepNodeParams<'gcx, 'tcx>`
- `TyCtxt<'tcx, 'tcx>`, `TyCtxt<'_, '_>`

An example: searching for variants

A first attempt:

```
@type@
@@
- TyCtxt<'gcx, 'tcx>
+ TyCtxt<'tcx>
```

This does part of the work, but some change sites are overlooked:

- `DepNodeParams<'gcx, 'tcx>`
- `TyCtxt<'tcx, 'tcx>`, `TyCtxt<'_, '_>`
- And others?

An example: searching for variants

A more general attempt:

```
@type@
identifier Ty;
@@
- Ty<'gcx, 'tcx>
+ Ty<'tcx>
```

An example: searching for variants

A more general attempt:

```
@type@
identifier Ty;
@@
- Ty<'gcx, 'tcx>
+ Ty<'tcx>
```

How to find other change sites, like `TyCtxt<'tcx, 'tcx>`, `TyCtxt<'_, '_>`:

- Want to change all uses of types that are somewhere used with `'gcx`.

An example: searching for variants

Fixing other uses in the same file:

```
@r type@
identifier Ty;
@@
- Ty<'gcx, 'tcx>
+ Ty<'tcx>

@type@
identifier r.Ty;
@@
(
- Ty<'tcx, 'tcx>
+ Ty<'tcx>
|
- Ty<'_, '_>
+ Ty<'_>
)
```

An example: using more metavariables

A more general attempt:

```
@r type@
identifier Ty;
@@
- Ty<'gcx, 'tcx>
+ Ty<'tcx>

@type@
identifier r.Ty;
lifetime a, b;
@@
- Ty<a, b>
+ Ty<b>
```

An example: using more metavariables

A more general attempt:

```
@r type@
identifier Ty;
@@
- Ty<'gcx, 'tcx>
+ Ty<'tcx>

@type@
identifier r.Ty;
lifetime a, b;
@@
- Ty<a, b>
+ Ty<b>
```

Should lifetime metavariables have ' ?

Summary: Features seen so far

- Semantic patches:
  Patch-like transformation specification, abstracted using metavariables.
- Multiple rules/rule ordering.
- Inheritance.
- Disjuctions.
- Typed metavariables.
- $*$ for matching without transformation.

All of these features are implemented!

Future features: . . . in parameter lists

One parameter case: (supported already)

```
@@
identifier f, P, p;
type T1, T2;
@@

- f<P: T1>(p: P) -> T2
+ f(p: impl T1) -> T2
      { ... }
```

Future features: ... in parameter lists

One parameter case: (supported already)

```
@@
identifier f, P, p;
type T1, T2;
@@

- f<P: T1>(p: P) -> T2
+ f(p: impl T1) -> T2
      { ... }
```

Maybe this should also modify the case of `where`?

Future features: ... in parameter lists

Multiple parameter case:

```
@@
identifier f, P, p;
type T1, T2;
@@

  f
-  <P: T1>
       (...,
-       p: P
+       p: impl T1
       ,...)
  { ... }
```

Future features: ... in parameter lists

Multiple parameter case:

```
@@
identifier f, P, p;
type T1, T2;
@@

  f
-  <P: T1>
       (...,
-       p: P
+       p: impl T1
       ,...)
  { ... }
```

Likewise for function arguments.

Future features: . . . across control-flow paths

A sequence of statements: (works already)

```
@@
identifier e;
expression rt;
@@
-    let mut e = tokio_executor::enter().unwrap();
-    e.block_on(rt.shutdown_on_idle());
+    rt.shutdown_on_idle();
```

tokio commit 47e2ff48d9f1daac7dba9f136b24eed64c87cf40

Future features: . . . across control-flow paths

The statements may not be contiguous:

```
@@
identifier e;
expression rt;
@@
-    let mut e = tokio_executor::enter().unwrap();
     ...
-    e.block_on(rt.shutdown_on_idle());
+    rt.shutdown_on_idle();
```

tokio commit 47e2ff48d9f1daac7dba9f136b24eed64c87cf40

Future features: . . . across control-flow paths

A safer variant:

```
@@
identifier e;
expression rt;
expression e1;
@@
-    let mut e = tokio_executor::enter().unwrap();
     ... when != e = e1
-    e.block_on(rt.shutdown_on_idle());
+    rt.shutdown_on_idle();
```

tokio commit 47e2ff48d9f1daac7dba9f136b24eed64c87cf40

Future features: Isomorpshisms

Isomorphism: A rewrite on the semantic patch to match and transform essentially equivalent code.

Examples for C:

- Explicitly defined isomorphisms:

```
Expression
@ not_ptr1 @
expression *X;
@@
 !X => X == NULL
```

```
Expression
@ paren @
expression E;
@@
 (E) => E
```

- Implicit isomorphisms
  - On a function definition the return type, `static`, `inline`, etc. can be omitted.
  - `e1 = e2` also matches a variable initialization.

Future features: An isomorphism for Rust

For `shutdown_on_idle`, the code is always written as:

```
let mut e = tokio_executor::enter().unwrap();
e.block_on(rt.shutdown_on_idle());
```

Future features: An isomorphism for Rust

For `shutdown_on_idle`, the code is always written as:

```
let mut e = tokio_executor::enter().unwrap();
e.block_on(rt.shutdown_on_idle());
```

But it could be written as:

```
tokio_executor::enter().unwrap().block_on(rt.shutdown_on_idle());
```

Future features: An isomorphism for Rust

```
@@
expression rt;
@@
- tokio_executor::enter().unwrap().block_on(rt.shutdown_on_idle());
+ rt.shutdown_on_idle();
```

Future features: An isomorphism for Rust

```
@@
expression rt;
@@
- tokio_executor::enter().unwrap().block_on(rt.shutdown_on_idle());
+ rt.shutdown_on_idle();
```

Potential implicit isomorphisms:

- Introduce `let` to name all possible subterms.
- Introduce `...` and `when` to allow other code between the `let` and the use.

Future features: An isomorphism for Rust

```
@@
expression rt;
@@
- tokio_executor::enter().unwrap().block_on(rt.shutdown_on_idle());
+ rt.shutdown_on_idle();
```

Potential implicit isomorphisms:

- Introduce `let` to name all possible subterms.
- Introduce `...` and `when` to allow other code between the `let` and the use.
- Caveat: Complexity may drastically increase if the `...` crosses a loop.

Future features: Another isomorpshism for Rust

Developers can use use with more or less information.

One example:

```
- use std::sync::Mutex;
+ use crate::loom::sync::Mutex;
```

Another example:

```
-use std::sync::{Arc, Mutex};
+use crate::loom::sync::{Arc, Mutex};
```

Options:

- Specify one change at a time?
- Merge changed code?
- Merge changed code with unchanged code?

Some more future Coccinelle features

- Position variables.
- Script code.
- Constraints on metavariables.
- Fresh identifiers.

Some Coccinelle internals

Input: Parsing provided by Rust Analyzer.

- Used both for Rust code and for semantic patch code.
- Will provide type inference, when needed (currently, loses concurrency).

Some Coccinelle internals

Input: Parsing provided by Rust Analyzer.

- Used both for Rust code and for semantic patch code.
- Will provide type inference, when needed (currently, loses concurrency).

Output: Pretty printing provided by `rustfmt`.

- To avoid problems with code not originally formatted with `rustfmt` (or formatted with a different version), the `rustfmt`ed changes are dropped back into the original code.
- Preserves comments and whitespace in the unchanged part of the code.

Some Coccinelle internals

In the middle:

- Wrap Rust code and semantic patch code, eg to indicate metavariables.
- Match semantic patch code against Rust code, to collect change sites and metavariable bindings.
- On a successful match, apply the changes, instantiated according to the metavariable bindings, reparse, and repeat with the next rule.

Practical issues

```
Usage: cfr [OPTIONS] --coccifile <COCCIFILE> <TARGETPATH>

Arguments:
 <TARGETPATH>  Path of Rust Target file/folder path

Options:
 -c, --coccifile <COCCIFILE>
         Path of Semantic Patch File path
 -o, --output <OUTPUT>
         Path of transformed file path
 -r, --rustfmt-config <RUSTFMT_CONFIG>
         rustfmt config file path
 -i, --ignore <IGNORE>
         [default: ]
 -d, --debug
     --apply
     --suppress-diff
     --suppress-formatting
     --no-parallel
     --worth-trying <WORTH_TRYING>
         strategy for identifying files that may be matched by the semantic patch
         [default: cocci-grep] [possible values: no-scanner, grep, git-grep, cocci-grep]
 -h, --help
         Print help
 -V, --version
         Print version
```

Conclusion

- Transformation on atomic terms completed (expressions, types, etc).
- Transformation on terms connected by a control-flow path (...) in progress.
- Small-scale testing has been done:
  - Replicating real changes on real Rust code.
- Patchparse extended to Rust, to find test cases at a larger scale.

Conclusion

- Transformation on atomic terms completed (expressions, types, etc).
- Transformation on terms connected by a control-flow path (...) in progress.
- Small-scale testing has been done:
  - Replicating real changes on real Rust code.
- Patchparse extended to Rust, to find test cases at a larger scale.

https://gitlab.inria.fr/coccinelle/coccinelleforrust.git