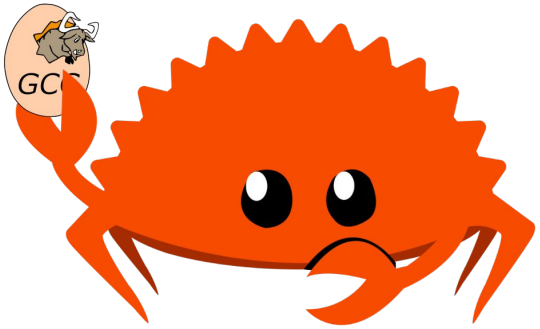
EMBECOSM®

OPEN SOURCE SECURITY

Rust-GCC



Arthur Cohen
<arthur.cohen@embecosm.com>

Summary

- What is GCC?
- What is Rust GCC?
- How do we do that?
  - Our parser
  - Our AST
  - Our HIR
  - Our “backend”
  - All the extra fun stuff we handle
- Workflow
- Community
- What’s coming: next steps, target code, target goals…

What is GCC?

- GNU Compiler Collection
- Old!
- Written in C++
- Multiples languages in one
- Not usable as a library
  - `libgccjit`
  - Or in-tree

What is Rust GCC?

- Full Implementation of Rust on top of GNU Toolchain
  - Project originally started in 2014, revived in 2019
    - Progress stalled with the frequency of language changes
    - Receives contributions from many GCC and non GCC developers
  - Thanks to Open Source Security, inc and Embecosm

Motivations of Rust GCC

- Upstream with mainline GCC
- Alternative implementation of Rust
  - Help drawing out Rust specification
- Reuses the GNU toolchain (`ld`, `as`, `gdb`)
- Reusing official Rust `libcore`, `libstd`, `libproc`
- Reuse existing GCC improvements
  - LTO, CFI, analyzers, security plugins...
- Drive adoption of Rust through backporting
- Backend support for more systems
- https://github.com/Rust-GCC/gccrs/wiki/Frequently-Asked-Questions

Current status

Milestone	Last Week	This Week	Delta	Start Date	Completion Date	Target
Data Structures 1 - Core	100%	100%	-	30th Nov 2020	27th Jan 2021	29th Jan 2021
Control Flow 1 - Core	100%	100%	-	28th Jan 2021	10th Feb 2021	26th Feb 2021
Data Structures 2 - Generics	100%	100%	-	11th Feb 2021	14th May 2021	28th May 2021
Data Structures 3 - Traits	100%	100%	-	20th May 2021	17th Sept 2021	27th Aug 2021
Control Flow 2 - Pattern Matching	100%	100%	-	20th Sept 2021	9th Dec 2021	29th Nov 2021
Macros and cfg expansion	100%	100%	-	1st Dec 2021	31st Mar 2022	28th Mar 2022
Imports and Visibility	100%	100%	-	29th Mar 2022	13th Jul 2022	27th May 2022
Const Generics	100%	100%	-	30th May 2022	10th Oct 2022	17th Oct 2022
Initial upstream patches	100%	100%	-	10th Oct 2022	13th Nov 2022	13th Nov 2022
Upstream initial patchset	100%	100%	-	13th Nov 2022	13th Dec 2022	19th Dec 2022

Current status

Upstream initial patchset	100%	100%	-	13th Nov 2022	13th Dec 2022	19th Dec 2022
Update GCC's master branch	0%	39%	+39	01st Jan 2023	-	03rd Mar 2023
Final set of upstream patches	31%	38%	+7%	16th Nov 2022	-	30th Apr 2023
Intrinsics and builtins	18%	18%	-	6th Sept 2022	-	TBD
Borrow checking	0%	0%	-	TBD	-	TBD
Const Generics 2	0%	0%	-	TBD	-	TBD
Rust-for-Linux compilation	0%	0%	-	TBD	-	TBD

Current status

- Const generics
- Intrinsics
- Working on Borrow-checking
- Working towards running the `rustc` test-suite
- Working on targeting an older version of `libcore (1.49)`

Pipeline Overview

- Parsing
- Expansion
- Name resolution
- Lowering to HIR
- Type Checking
- Linting or error verification
- Lowering to `tree` (-> GCC middle-end)

Frontend Representations

- AST (Abstract Syntax Tree)
  - Raw AST (Structured C++ class hierarchy)
- HIR (high level IR)
  - Desugared AST
    - remove distinction between functions/methods
    - macros don’t exist anymore
    - much much more….
- Generic (GCC IR)

Macro Expansion

- Macro arguments are typed
  - `expr, stmt, path, pat, vis...`
- Repetitions
- Mathematical logic
  - Kleene Operators
    - `* ? +`
  - Follow-set Ambiguity Restrictions
  - That we need to implement!

Macro Expansion

```
macro_rules! add {
    ($e:expr) => { $e };
    ($e:expr, $($es:expr),*) => { $e + add!($($es),*) };
}

add!(1); // 1
add!(1, 2, 4); // 7
add!(1, add!(2, 3), five(), b, 2 + 4);
```

Macro Expansion

```
macro_rules! tuplomatron {
  ($($e:expr),* ; $($f:expr),*) => { ( $( ( $e, $f ) ),* ) };
}

let tuple = tuplomatron!(1, 2, 3; 4, 5, 6); // valid
let tuple = tuplomatron!(1, 2, 3; 4, 5); // invalid
```

Macro Expansion

```
macro_rules! invalid {
  ($e:expr forbidden) => {};
  //  Forbidden by the follow-set ambiguity restriction

  ($e:expr $(,)? $(;)* $(=>)* forbidden) => {};
  //  1      2     3     4        5          (matches)
}
```

Extra HIR checks

- Privacy pass
  - Privacy in Rust is very different from C++
  - `pub(in path)`, `pub(super)`, `pub(crate)`…
- Unsafe
  - Some actions are only allowed in `unsafe` contexts
    - Dereferencing raw pointers, calling unsafe or extern functions, use of mutable or extern static variables, inline assembly…

Other Rust specific shenanigans

- Macros are lazy
  - No they're not
- Code sharing between crates
  - Headers like C/C++?
  - Dark `ELF` magic?
    - AST Serializing/Deserializing
- Type system
  - Extremely complex and powerful
  - Never type, GATs…
  - Sum types
  - Not a lot of GCC-languages have that!
- Inline assembly
  - Different from GCC's
  - Translation required

Contributing | Reviewing | Merging | Upstreaming

Inspired from `rustc`'s workflow

- Github
- Zulip
- `bors r+`

But also...

- IRC
- gcc-rust@gcc.gnu.org
- Mailing list and patches

- No matter your background, you can contribute

GCC development is hard

- Email based code submitting/reviewing is difficult
- GCC Changelogs are hard to write
- Pushing directly to GCC's main branch
- `git send-email`

```
commit a5d7d39d552b490c60192ae042fe955f0fec590e (HEAD)
Author: Arthur Cohen <arthur.cohen@embecosm.com>
Date:   Wed Jan 18 12:23:03 2023 +0100

    macro: Allow builtin `MacroInvocation`s within the AST

    This commit turns AST::MacroInvocation into a sum type.
    The class can now represent a regular macro invocation (lazily expanded)
    or a builtin one (eagerly expanded)

    gcc/rust/ChangeLog:

            * expand/rust-macro-builtins.cc (make_macro_invocation): Add short hand
            function for returning fragments containing macro invocations.
            (MacroBuiltin::compile_error_handler): Add explanation for eager
            invocation
```

EMBECOSM®

GCC development is hard

- We submit patches/commits to GCC's mailing list for your contributions
- Lots of CI
- Lots more machines building and bootstrapping `gccrs`
- Commit format checkers
- Working on a bot to post the Changelog template

GCC development is hard

- GCC development stages
  - Some files cannot be edited from November to May
- We keep track of that
  - Maintaining a “GCC-ready” branch
  - As well as our main development branch

Is it working?

- More than 50 contributors in 2022 overall
- Multiple students
  - Multiple internships
- GCC developers
- Rust core team

Status | Future Work | Open Questions

When is it ready?

- Can compile `libcore` and actually works
  - Implements all necessary `lang` items
  - Unstable APIs, macros, attributes...
  - Passes the `rustc 1.49` testsuite!
- `libcore, liballoc...`
- `libproc`
  - Powerful procedural macros
  - Requires an RPC server in the front-end
- Borrow checking
  - Polonius project
    - Having it optional is a no go for the community
- We are part of this year's GSoC!

GSoC

- GSoC student Faisal Abbas ported large portions of C++ constexpr evaluation

```
const A: i32 = 1;

const fn test(a: i32) -> i32 {
    let b = A + a;
    if b == 2 {
        return b + 2;
    }

    a
}

const B: i32 = test(1);
const C: i32 = test(12);
```

GSoC

- HIR debugging dump
- Unicode support
- Metadata exports
- Better user error handling + Rust error codes

Tooling

- Testing project
  - Tries compiling various projects using `gccrs`
    - `blake3` cryptography library
    - `libcore` 1.49
    - All the valid cases from the `rustc` testsuite
      - in `#[no_std]` mode
      - in `#[no_core]` mode
  - Eventually add RfL to it!
- Testsuite generator
- Website
- Report generator and tooling
- `cargo-gccrs`
- Web dashboard

Finally...

- RiiR ?
  - Limited to Rust 1.49 for bootstrapping purposes
    - `gccrs` “1.0” will be able to compile `gccrs` “2.0”
  - Still a ways off :)
- The goal is NOT to break the ecosystem

```
arthur@platypus ~/G/r/gccrs (master) [1]> build/gcc/rust1 test.rs
rust1: fatal error: gccrs is not yet able to compile Rust code properly. Most of the errors produced wi
ll be gccrs' fault and not the crate you are trying to compile. Because of this, please reports issues
to us directly instead of opening issues on said crate's repository.

Our github repository: https://github.com/rust-gcc/gccrs
Our bugzilla tracker: https://gcc.gnu.org/bugzilla/buglist.cgi?bug_status=__open__&component=rust&produ
ct=gcc

If you understand this, and understand that the binaries produced might not behave accordingly, you may
 attempt to use gccrs in an experimental manner by passing the following flag:

`-frust-incomplete-and-experimental-compiler-do-not-use`

or by defining the following environment variable (any value will do)

GCCRS_INCOMPLETE_AND_EXPERIMENTAL_COMPILER_DO_NOT_USE

For cargo-gccrs, this means passing

GCCRS_EXTRA_FLAGS="-frust-incomplete-and-experimental-compiler-do-not-use"

as an environment variable.
compilation terminated.
```

Community

Links

- Github: https://rust-gcc.github.io/
- Reports: https://github.com/Rust-GCC/Reporting
- Zulip: https://gcc-rust.zulipchat.com/
- IRC: irc.oftc.net #gccrust
- https://gcc.gnu.org/mailman/listinfo/gcc-rust

Get Involved

- Goal is to make working on compilers fun
  - Lots of `good-first-pr` issues to work through
    - Refactoring work
    - Bugs
  - Lots of scope to make your mark on the compiler
- Google Summer of Code 2021 and 2022
- Status reporting
  - Weekly and Monthly
  - Shout out to contributors
  - Open and transparent
- Monthly Community Call and Weekly Syncup
  - In our calendar and Zulip
  - Open to everyone who is interested
  - Hosted on Jitsi


OPEN SOURCE SECURITY

ORACLE

Questions?

github.com/Rust-GCC/gccrs/

gcc-rust.zulipchat.com/

irc.oftc.net #gccrust