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
## Current status

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

<table>
<thead>
<tr>
<th>Component</th>
<th>Status 1</th>
<th>Status 2</th>
<th>Change</th>
<th>Target Date</th>
<th>Next Target Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upstream initial patchset</td>
<td>100%</td>
<td>100%</td>
<td>-</td>
<td>13th Nov 2022</td>
<td>13th Dec 2022</td>
</tr>
<tr>
<td>Update GCC’s master branch</td>
<td>0%</td>
<td>39%</td>
<td>+39</td>
<td>01st Jan 2023</td>
<td></td>
</tr>
<tr>
<td>Final set of upstream patches</td>
<td>31%</td>
<td>38%</td>
<td>+7%</td>
<td>16th Nov 2022</td>
<td>30th Apr 2023</td>
</tr>
<tr>
<td>Intrinsics and builtin</td>
<td>18%</td>
<td>18%</td>
<td>-</td>
<td>6th Sept 2022</td>
<td>TBD</td>
</tr>
<tr>
<td>Borrow checking</td>
<td>0%</td>
<td>0%</td>
<td>-</td>
<td>TBD</td>
<td>TBD</td>
</tr>
<tr>
<td>Const Generics 2</td>
<td>0%</td>
<td>0%</td>
<td>-</td>
<td>TBD</td>
<td>TBD</td>
</tr>
<tr>
<td>Rust-for-Linux compilation</td>
<td>0%</td>
<td>0%</td>
<td>-</td>
<td>TBD</td>
<td>TBD</td>
</tr>
</tbody>
</table>
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

```rust
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

```rust
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
```
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
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

```cpp
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

```bash
arthur@platypus ~/G/r/gccrs (master) [1]> build/gcc/rustl test.rs
error: fatal error: gccrs is not yet able to compile Rust code properly. Most of the errors produced will 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

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:

- --rust-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="--rust-incomplete-and-experimental-compiler-do-not-use"

as an environment variable.

Compilation terminated.
```
Community

![Image of two mugs and a stuffed toy with text: compilers are hard and git merge github/gccrs]
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
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
Questions?

github.com/Rust-GCC/gccrs/
gcc-rust.zulipchat.com/
irc.oftc.net #gccrust