Unicode Support for GCC Rust Frontend

Raiki Tamura

March 28 2023

1. Abstract

The Rust programming language supports Unicode in identifiers, similarly to other modern programming languages such as Go, C++, and JavaScript. The main goals of this project are supporting Unicode identifiers in GCC Rust and improving their location information for better error messages.

2. Introduction and Goals

The goals of this projects are:

- supporting Unicode identifiers and whitespaces;
- adding check for values of `crate_name` attributes;
- improving location information of identifiers; and
- modifying name mangling schemes (secondary goal).

Compiling Rust programs works properly without modifying the current implementation of name mangling, but it is desirable to use properly mangled names in the backend. Therefore this is set as a secondary goal.

3. Background

3.1 Overview of Unicode in Rust

Rust allows the use of Unicode characters in several places, including identifiers, literals, whitespaces, and `crate_name` attributes [1]:

- 11 characters including non-ASCII characters can be used as whitespaces.
- Unicode characters and Unicode escape are used for character and string literals.
- Rust adopts identifiers defined in UAX#31 (Unicode Standard Annex #31). After tokenized, identifiers are normalized to the NFC form described in UAX#15.
- Only Unicode alphanumeric characters and underscore (U+005F) can be used for values of `crate_name` attributes.

Identifiers of functions are mangled in one of two ways, legacy mangling scheme or v0 mangling scheme. [2] The former uses codepoints as mangled names while the latter converts Unicode strings to punycode.

3.2 Present status of gccrs

3.2.1 Support for Unicode

gccrs supports Unicode in literals and `crate_name` attributes, but not in identifiers or whitespaces for now. Values of `crate_name` attributes are not checked so checks have to be added for them. Also, gccrs implements part of the v0 mangling scheme, lacking conversion from Unicode to punycode.

3.2.2 Location information of identifiers

The current implementation uses `std::string` to represent identifiers and keeps location information of them separately. In order to manage location information better, we should add a new kind of AST node representing identifiers which contains a byte string and the location information of it.

3.3 Idea for the implementation

The implementation can be divided into four steps.

The first step

First, the lexer has to be modified so that it can tokenize Unicode identifiers and whitespaces. In order to tokenize and normalize identifiers, I plan to modify `libcpp/ucnid.h`, which is in the folder shared with all the gcc frontends. (For details, please see the end of this section.) In this step, I also plan to add a definition of valid whitespaces to the lexer. I have already made a small lexer [3] based on gccgo, which supports Unicode identifiers as well, so I have a concrete idea for this step.

The second step

Next, values of `crate_name` attributes will be checked. We can use `libcpp/ucnid.h`, which is modified in the first step, to check if given crate names only contain Unicode alphanumeric characters and underscores.

The third step

In this step, a new class `Rust::Identifier` will be introduced. It requires replacing all the use of std::string which represents identifiers with `Rust::Identifier`, and this change ripples across the codegen stage and all of the existing AST-based static analyses. Replacing them one by one is expected to be the most difficult and time-consuming part of this project.

The last step

Rust RFC defines the v0 mangling scheme, which requires conversion from Unicode to punycode. In this step, I plan to implement the punycode conversion using the GNU IDN library [4].

Utilize `libcpp/ucnid.h`

`libcpp/ucnid.h` is used only in the C++ frontend, but I plan to reuse it in gccrs. This file defines a lookup table of `XID_Start`, `XID_Continue`, NFC, etc. In order to reuse it to look up Unicode alphanumeric, some rows will be added to it. Then the table will be used to tokenize and normalize identifiers and to check values of `crate_name` attributes. I discussed this on the gcc mailing list. [5]

5. Timeline

Week 1 (May 14 - May 20)
- Start the first step
- Modify `libcpp/ucnid.h`
- Add functions `is_XID_start` and `is_XID_continue` to check if their parameters are Unicode `XID_Start` or `XID_Continue` and use them in the lexer.

Week 2 (May 21 - May 28)
- Modify the function `is_whitespace` to accept all whitespaces defined in the Rust Reference.
- Add unit tests and integration tests for the lexer.

Week 3 (May 28 - June 3)
- Add the new function `normalize_identifier` to normalize identifiers to their NFC form.

Week 4 (June 4 - June 10)
- Start the second step
- Add the function `is_utf8_alphanumeric` and use it to check values of the `crate_name` attributes.

Week 5 (June 11 - June 17)
- Add unit tests and integration tests for the `crate_name` attributes

Week 6 (June 18 - June 24)
- Start the third step
- Add a new class `Rust::Identifier` which holds their location information and string data.
- Remove the existing type `Rust::Identifier`.

Week 7 (June 25 - July 1)
- Prepare for the first evaluation and write the report for it.

Week 8 (July 2 - July 8)
- Modify all the uses of the previous `Rust::Identifier` in AST and HIR.

Week 9 (July 9 - July 15)
- Modify all the uses of the previous `Rust::Identifier` in the backend.

Week 10 (July 16 - July 22)
- Exam week

Week 11 (July 23 - July 29)
- Exam week

Week 12 (July 30 - Aug 5)
- Spare week in case of getting delayed and for other additional works

Week 13 (Aug 6 - Aug 12)
- Start of the last step
- Add the GNU IDN library to the build configuration.
- Add the new function `convert_to_punycode`.

- Add the new function `replace_hiphen_with_underscore` to use punycode as symbol names in the backend.

Week 14 (Aug 13 - Aug 19)

- Add unit and integration tests for the v0 mangling scheme.

Week 15 (Aug 20 - Aug 27)

- Prepare for the final evaluation and write the final report.

6. Personal Information

Name: Raiki Tamura
Residence: Japan
Email: tamaron1203@gmail.com
Affiliation: Undergraduate student at Kyoto University
Timezone: UTC+9
GitHub: https://github.com/tamaroning
CV:
https://drive.google.com/file/d/1lUDoX8HfoMVXhgkIceFqtaG_0Suus-wD/view?usp=sharing

6.1 Availability

The spring semester of my university starts in April and ends in July, but I will be able to spend at least 35 hours per week because the load of lectures in the semester is not heavy. In the last year, a student in the same university took part in and completed a GSoC project in the semester. During the exams in mid July, which lasts two weeks, however, it will be difficult to spend much time. So this project will be extended to end a few weeks later.

6.2 Previous Experiences

I have experience in building a C-subset compiler [6] from the ground up and contributing to Clippy, a Rust linter, so I am not concerned about the ability to implement parsers and to use interfaces of compilers. In addition, I made a small ELF linker [7], so I am also familiar with stuff close to the compiler backend.

Recently, thanks to advice from a potential mentor, Arthur, I implemented new features for GCC Rust. The PR#1708 [8] adds support for the syntax of Declarative Macro 2.0 and the PR#1651 [9] fixes the lexer dump. Besides, I sent a few small patches which are already merged to master. [10] [11]

6.3 Motivation

I enjoy learning system programming and love computer science for more than four years. I am especially interested in compilers and static analysis these days.

I am deeply interested in participating in the project, because of my prior experience with compiler frontend and this is a precious opportunity to get familiar with other parts of compiler development.

7 References

[1] Rust Reference: https://doc.rust-lang.org/reference/
[2] Rust RFCs: https://rust-lang.github.io/rfcs/
[3] My experimental lexer: https://github.com/tamaroning/gccrs-unicode
[4] GNU IDN library: https://www.gnu.org/software/libidn/
[5] Discussion on the mailing list: https://gcc.gnu.org/pipermail/gcc/2023-March/240872.html
[6] My C-subset compiler: https://github.com/tamaroning/ccr
[7] My ELF linker: https://github.com/tamaroning/myld
[8] Gccrs PR#1708: https://github.com/Rust-GCC/gccrs/pull/1708
[9] Gccrs PR#1651: https://github.com/Rust-GCC/gccrs/pull/1651
[10] Gccrs PR#1706: https://github.com/Rust-GCC/gccrs/pull/1706
[11] Gccrs PR#1633: https://github.com/Rust-GCC/gccrs/pull/1633