Unicode Support for GCC Rust Frontend

Raiki Tamura

3, Feb (Mon), FOSDEM 2024

Outline

1. About My Project
2. Unicode in Rust
3. Implementation
4. Mangling schemes
5. Summary

Who am I

Raiki TAMURA

Undergraduate student at Kyoto University, Japan

Participated Google Summer of Code 2023 (GCC)

My interests are compilers and low-level programming

About My Project

- I worked on Unicode support for GCC Rust as a GSoC2023 project
  - *Google Summer of Code is a global, online program focused on bringing new contributors into open source software development. GSoC Contributors work with an open source organization on a 12+ week programming project under the guidance of mentors.* (https://summerofcode.withgoogle.com/)
- And now, I am working on supporting the new Rust mangler in GCC Rust

Unicode in Rust

- Non-ASCII newlines and white-spaces can be used
- `#![crate_name="..."]` accepts Unicode alphabetic and numeric characters
  - Note that Unicode alphabetic characters include non-ASCII codepoints
- More characters can be used for identifiers
  - e.g. Gödel, ほげ, 안녕하세요

Unicode in Rust: Identifiers

Rust adopts the syntax of identifiers defined in UAX #31

- Also adopted by ECMAScript, C++ (C++23), Python (3.0), etc.

After being tokenized, Identifiers are normalized to NFC

```
IDENTIFIER_OR_KEYWORD :
    XID_Start XID_Continue*
  | _ XID_Continue+
```

https://doc.rust-lang.org/reference/identifiers.html

Implementation

There are other frontends supporting Unicode

- libcpp/: C preprocessor with lexer
  - C++ adopts the same syntax of Unicode identifiers as Rust
- gcc/go/: Go frontend

Implementation

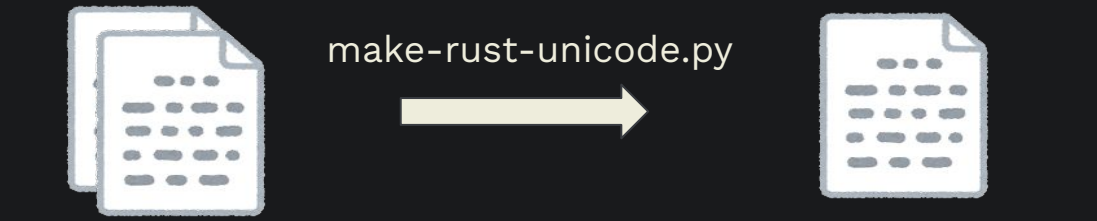
Implementation is divided into 3 parts

1. Modify the lexer to accept Unicode characters
2. Implement #![crate_name=“xxxx”] attribute
3. Modify the manglers to handle Unicode identifiers

Implementation: the Lexer part

In order to look up character properties, we reuse some functions in libcpp/

For other missing properties, we generate a header file from Unicode data files



```
26    const std::map<uint32_t, std::vector<uint32_t>> DECOMPOSITION_MAP = {
27      // clang-format off
28      {0x00c0, {0x0041, 0x0300, }},
29      {0x00c1, {0x0041, 0x0301, }},
30      {0x00c2, {0x0041, 0x0302, }},
31      {0x00c3, {0x0041, 0x0303, }},
32      {0x00c4, {0x0041, 0x0308, }},
33      {0x00c5, {0x0041, 0x030a, }},
34      {0x00c7, {0x0043, 0x0327, }},
35      {0x00c8, {0x0045, 0x0300, }},
36      {0x00c9, {0x0045, 0x0301, }},
37      {0x00ca, {0x0045, 0x0302, }},
38      {0x00cb, {0x0045, 0x0308, }},
39      {0x00cc, {0x0049, 0x0300, }},
40      {0x00cd, {0x0049, 0x0301, }},
41      {0x00ce, {0x0049, 0x0302, }},
42      {0x00cf, {0x0049, 0x0308, }},
43      {0x00d1, {0x004e, 0x0303, }},
44      {0x00d2, {0x004f, 0x0300, }},
45      {0x00d3, {0x004f, 0x0301, }},
46      {0x00d4, {0x004f, 0x0302, }},
47      {0x00d5, {0x004f, 0x0303, }},
48      {0x00d6, {0x004f, 0x0308, }},
49      {0x00d9, {0x0055, 0x0300, }},
50      {0x00da, {0x0055, 0x0301, }},
51      {0x00db, {0x0055, 0x0302, }},
52      {0x00dc, {0x0055, 0x0308, }},
53      {0x00dd, {0x0059, 0x0301, }},
54      {0x00e0, {0x0061, 0x0300, }},
55      {0x00e1, {0x0061, 0x0301, }},
56      {0x00e2, {0x0061, 0x0302, }},
57      {0x00e3, {0x0061, 0x0303, }},
58      {0x00e4, {0x0061, 0x0308, }},
```

https://github.com/Rust-GCC/gccrs/blob/7c0daba10e43586df2ede9cd4037c50b85648e6a/gcc/rust/util/rust-unicode-data.h

Implementation: `#![crate_name]` attribute

Generate a codepoint table of Unicode alphabetic and numeric

Use it to validate values of the attribute

```
for (Codepoint &c : uchars)
  {
    if (!(is_alphabetic (c.value) || is_numeric (c.value) || c.value == '_'))
{
  error = Error (UNDEF_LOCATION,
    "invalid character %<%s%> in crate name: %<%s%>",
    c.as_string ().c_str (), crate_name.c_str ());
  return false;
}
  }
```

https://github.com/Rust-GCC/gccrs/blob/7c0daba10e43586df2ede9cd4037c50b85648e6a/gcc/rust/rust-session-manager.cc

Implementation: the Mangler part

Modify the default (legacy) mangler to handle Unicode

- legacy mangling scheme escapes non-ASCII characters as their codepoints

Implement the new mangling scheme (v0)

- identifiers are encoded as Punycode
- This part is now in progress

Mangling Schemes

- There are two mangling schemes: legacy and v0
  - You can pass options to switch a mangling scheme:
    - rustc -C symbol-mangling-version=v0
    - gcc -frust-mangling=[legacy | v0]
- v0 was introduced to rustc on 2019 and it is used in the Rust for Linux project

Mangling Schemes: legacy vs v0

	Legacy	v0
Prefix	_Z	_R
Characters	A-Z, a-z, 0-9, _, $, .	A-Z, a-z, 0-9, _
Contains type info	No	Yes
Unicode identifiers	Escaped as XX	Punycode

Example : legacy and v0

Example: fn 関数() {}

legacy: `_ZN7example15_$u95a2$$u6570$17hb64df414284d985b`

v0: `_RNvCsjZmpILMU2JV_7exampleu7kdvt68h`

Example : legacy and v0

Example: fn 関数() {}

legacy: `_ZN7example15_$u95a2$$u6570$17hb64df414284d985b`

v0: `_RNvCsjZmpILMU2JV_7exampleu7kdvt68h`

- non-ASCII characters are escaped in legacy
- In v0, they are encoded as Punycode

Summary

As a result of GSoC 2023, gccrs supports Unicode

Rust compilers use Unicode normalization and Punycode encoding

Implementing the new v0 mangler to gccrs is in progress

Acknowledgements

Attendance at FOSDEM2024 was supported by the GNU Toolchain fund, a part of the FSF's Working Together for Free Software Fund: https://my.fsf.org/civicrm/contribute/transact?reset=1&id=57

Also I would like to thank to my mentors (Philip Herron, Arthur Cohen), GCC Rust team, and another GSoC student (Muhammad Mahad)

List of References

[1] The Rust RFC Book, https://rust-lang.github.io/rfcs/2603-rust-symbol-name-mangling-v0.html

[2] GCC Rust, https://github.com/Rust-GCC/gccrs

[3] Swift documentation, https://github.com/apple/swift/blob/467f684a430cff032ce4ee42bb4c172c9bbcbc85/docs/ABI/Mangling.rst