Pointer Provenance

By Jonathan Louie

Agenda

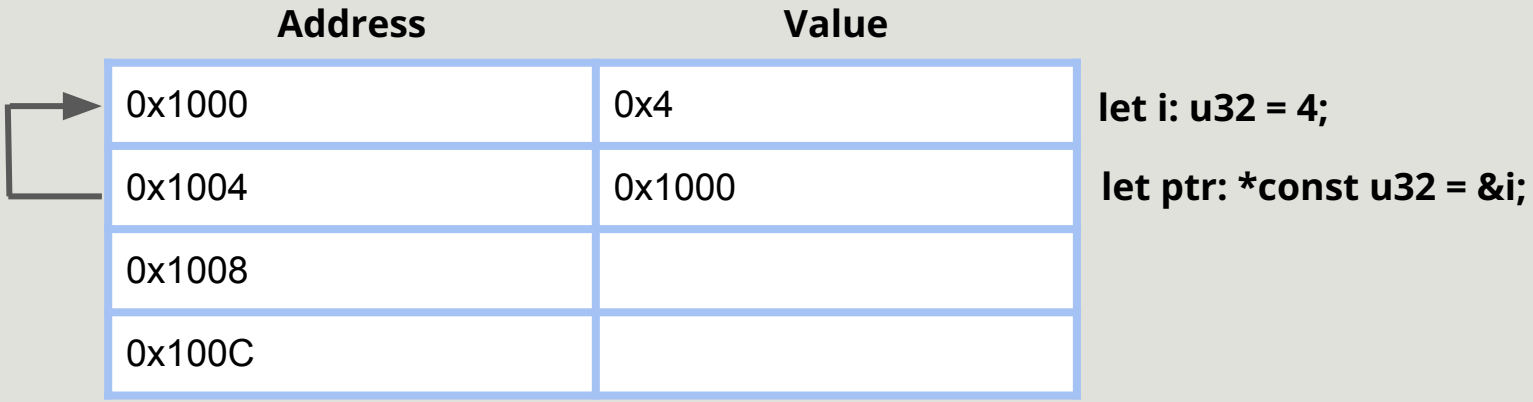
What is a Pointer?

What is Provenance?

Strict vs. Exposed Provenance

What is a Pointer?

Memory


Address
Value
0x1000
0x4
let i: u32 = 4;
0x1004
0x1000
let ptr: *const u32 = &i;
0x1008
0x100C

Pointers in Rust

Example:
https://play.rust-lang.org/?version=stable&mode=debug&edition=2021&gist=4d2ac0759ad599f95d0515bb5d4243d9

- **Used for manual memory management**
  - **e.g. FFI**
- **Requires usage of unsafe keyword to dereference**

Pointers vs. References

Example:
https://play.rust-lang.org/?version=stable&mode=debug&edition=2021&gist=9ad5536371d00b3a6de1e36ee4a13333

Rust's references are safe to dereference due to how the compiler enforces borrow checker rules

Safe Pointer Dereferencing

"For a pointer to be valid, it is necessary, but not always sufficient, that the pointer be dereferenceable. The provenance of the pointer is used to determine which allocated object it is derived from; a pointer is dereferenceable if the memory range of the given size starting at the pointer is entirely contained within the bounds of that allocated object. Note that in Rust, every (stack-allocated) variable is considered a separate allocated object."

https://doc.rust-lang.org/beta/std/ptr/index.html#safety

Pointers vs. Integers

- Pointers are represented by usize data type
- Pointers are not, in fact, just integers
  - Pointers have a provenance associated with them too
- Conversion from pointer to integer is (usually) lossy
  - Provenance is lost
  - Exception: https://doc.rust-lang.org/beta/std/ptr/fn.without_provenance.html

What is Provenance?

Dictionary Definition of Provenance

Provenance is the chronology of the ownership, custody or location of a historical object.

https://www.merriam-webster.com/dictionary/provenance

https://en.wikipedia.org/wiki/Provenance

Provenance in Rust

- A pointer consists of two components:
  - An address (usize)
  - Its provenance
- Provenance consists of three components:
  - Spatial: The set of addresses the pointer is allowed to read from
  - Temporal: The timespan that the pointer is allowed to access those addresses
  - Mutability: Whether a pointer has read or write access to memory
    - A subset of its addresses may be mutable, or mutability may only be allowed during a subset of its max timespan

Strict vs. Exposed Provenance

Strict Provenance

- A set of APIs designed to make working with provenance more explicit
- Only allows integer-to-pointer conversions via with_addr function
  - with_addr copies provenance to new pointer
  - pub fn with_addr(self, addr: usize) -> Self;

Examples

https://play.rust-lang.org/?version=stable&mode=debug&edition=2021&gist=2305941b5f3ec6dcabe5b389ea73adc7

Exposed Provenance

- Allows integer-to-pointer casting
  - Useful in some bare-metal platforms requiring synthesis of pointers "out of thin air" without obtaining proper provenance
- Less solid semantics than Strict Provenance
- Unclear whether a satisfying unambiguous semantics can be defined for Exposed Provenance (for now)

Key Takeaways

- **Pointers are not just integer addresses, since they also have a provenance**
- **Integer-to-pointer casts are not allowed by Strict Provenance semantics, but are allowed in Exposed Provenance semantics**
- **Use** `MIRIFLAGS=-Zmiri-strict-provenance` **to enforce Strict Provenance APIs to be used**
- **Remember: MIRI only checks code that is executed!**

Further Reading

https://doc.rust-lang.org/beta/std/ptr/index.html