Rust integration in Stak Scheme

@raviqqe

January 26, 2024

Contents

Stak Scheme

- A bytecode compiler and virtual machine (VM) for Scheme
    - The compiler is written in Scheme.
    - The VM is written in Rust.
- It aims to support the R7RS-small standard.
- Forked from Ribbit Scheme

Progress

- Dynamically-typed functions in Rust
- Rust integration in Stak Scheme

Dynamically-typed functions in Rust

- `any-fn` crate
  - A crate to define dynamically-typed functions in Rust.
- You can convert many of statically-typed functions in Rust automatically.
  - As long as types define `core::any::Any`.
- Different functions need different conversion.
  - We implement different generic traits for them.
  - Rust's trait resolution handles it (almost) automatically.
  - Other scripting languages in Rust, such as e.g. Rhai and Steel, uses the same mechanism.

Example 1

- Convert values with the `value` function.
- Convert functions with the `#fn` function.

```rust
use any_fn::{r#fn, value};

struct Foo {
    foo: usize,
}

fn foo(x: usize, y: &mut Foo) {
    y.foo = x;
}

let x = value(Foo { foo: 0 });
r#fn(foo).call(&[&value(42usize), &x]).unwrap();
assert_eq!(x.downcast_ref::<Foo>().unwrap().foo, 42);
```

Example 2

- We need to annotate conversion a bit to handle immutable reference parameters.
- There is no (positive) trait that distinguish immutable references and unboxed values.

```rust
use any_fn::{r#fn, Ref, value};

fn foo(x: usize, y: &usize, z: &mut usize) {
    *z = x + *y;
}

let x = value(0usize);

r#fn::<(_, Ref<_>, _), _>(foo)
    .call(&[&value(40usize), &value(2usize), &x])
    .unwrap();

assert_eq!(*x.downcast_ref::<usize>().unwrap(), 42);
```

Rust integration in Stak Scheme

- In Stak Scheme, you can define `PrimitiveSet` which is a set of primitive functions in Rust.
- They can handle only Scheme types.
- The new `DynamicPrimitiveSet` handles conversion of values and functions between Scheme and Rust.

Examples

```rust
struct Foo {
    bar: usize,
}

impl Foo {
    fn new(bar: usize) -> Self { ... }
    fn bar(&self) -> usize { ... }
    fn baz(&mut self, value: usize) { ... }
}

let mut functions = [
    r#fn(Foo::new),
    r#fn::<(Ref<_>,), _>(Foo::bar),
    r#fn(Foo::baz),
];

DynamicPrimitiveSet::<HEAP_CAPACITY>::new(&mut functions);
```

Future work

- Auto conversion of primitive types
    - e.g. usize, f64
- Garbage collection of foreign objects

Summary

- Building Rust integration is fun! 😃