Embedding a DSL in Rust

Kyle Headley

kyleheadley.github.io

DSL in Rust

- Parse with Rust parser
- Type check with Rust type checker
- Run as a Rust executable

Example:

kyleheadley.github.io find “Traitlang DSL”

```
macro_rules! {
 (pattern1) => (template1);
 (pattern2) => (template2);
}
```

- patterns checked in order
- works like text replacement
- hygienic
- repeated calls

- parameters
- repeaters
- procedural macros different

```
macro_rules! expr {

 (0) => (Num(Zero));

 ($a:tt $p:tt) => (App(expr![$a],expr![$p]));

  ($n:tt + $($ns:tt)+)
  => (Plus(expr![$n],expr![$($ns)+]));
}
```

BNF Grammar

```
T :=
  (T)              parens
  n                base type
  T1 -> T2 -> …    arrow
```

```
E :=
  E1 E2            application
  E1 E2 E3 …       multi-
                   application
```

Macro definition

```
macro_rules! typ {
    (($($ts:tt)+)) => (typ![$ts]);
    (N) => (Number);
    ($t:tt -> $($ts:tt)+)
      => (Arrow(typ![$t],typ![$($ts)+]));
}
```

```
macro_rules! expr {
    ($a:tt $p:tt)
      => (App(expr![$a],expr![$p]));
    ($a:tt $p:tt $($ps:tt)+)
      => (expr![{^App(expr![$a],expr![$p])} $($ps),+]);
}
```

Traitlang

Functional Programming

Operational Semantics Rules

```
struct NewType;

trait NewTrait {}

impl NewTrait for NewType {}
```

```
struct NewType;

trait NewTrait {}

impl NewTrait for NewType {}
```

Key Features:

Parameters

Associated Types

Constraints

*Haskell: Type Families

Equality for natural numbers

```
trait NatEq<N> { type Eq; }

impl NatEq<Zero> for Zero { type Eq=True; }

impl<N> NatEq<Succ<N>> for Zero { type Eq=False; }

impl<N> NatEq<Zero> for Succ<N> { type Eq=False; }

impl<N1,N2,E> NatEq<Succ<N1>> for Succ<N2> where
   N2: NatEq<N1,Eq=E>
{ type Eq=E; }
```

*But complex functions are much harder

Wellformedness

```
trait Expr {}

struct Plus<N1,N2>(N1,N2);
impl<N1:Expr,N2:Expr>
  Expr for Plus<N1,N2> {}
```

Assigning Types

```
trait Typed<Ctx> { type T; }
```

Typing
Rule

$$\frac{\Gamma \vdash E1 : T1 \rightarrow T2 \qquad \Gamma \vdash E2 : T1}{\Gamma \vdash E1\ E2 : T2}$$

Typing
Rule

(Forall Γ,E1,E2)
(Case Γ ⊢ E1 E2)

```
Γ ⊢ E1:T1 -> T2
Γ ⊢ E2:T1
```

```
Γ ⊢ E1 E2:T2
```

Typing
Code

```
impl<Ctx,E1,E2,T1,T2>
Typed<Ctx> for App<E1,E2> where
  E1:Typed<Ctx,T=Arrow<T1,T2>>,
  E2:Typed<Ctx,T=T1>

{ type T=T2; }
```

Thanks!

DSL in Rust

- Parse with Rust parser
  Resembling BNF grammar
- Type check with Rust type checker
  Resembling Typing judgement rules
- Run as a Rust executable
  Using regular code

kyleheadley.github.io find “Traitlang”