```ts
type RuleFunc<A, B = A> = (ctx: VContext<A>) ⇒ VContext<B>

type Rule<Type extends RuleType, A, B = A> = {
  type: Type
  func: RuleFunc<A, B>
}

export type Fuji<Types extends RuleType, Value> = {
  rules: Rule<Types, Value>[]
}
```

```
export type RuleType =
    | NullableType
    | ShapeMismatchType
    | StringType
    | RequiredType
    | RequiredIfType
    | PositiveType
    | BoolType
    | IncludesType
    | OneOfType
    ...
```

```
string() // Rule<'string', string, string>
required() // Rule<'required', any, any>
f(string(), required()) // Fuji<'string' | 'required', string>
```

```ts
type RuleFunc<A, B = A> = (ctx: VContext<A>) => VContext<B>

type Rule<Type extends RuleType, A, B = A> = {
  type: Type
  func: RuleFunc<A, B>
}

export type Fuji<Types extends RuleType, Value> = {
  rules: Rule<Types, Value>[]
}
```

```ts
export type RuleType =
  | NullableType
  | ShapeMismatchType
  | StringType
  | RequiredType
  | RequiredIfType
  | PositiveType
  | BoolType
  | IncludesType
  | OneOfType
```

```ts
string() // Rule<'string', string, string>
required() // Rule<'required', any, any>
f(string(), required()) // Fuji<'string' | 'required', string>
```

```ts
const len = (s: string): number => s.length
const isZero = (n: number): boolean => n === 0

function pipe<A, B, C>(
  f1: (x: A) => B,
  f2: (x: B) => C
) {
  return (arg: A) => f2(f1(arg))
}

const isEmpty = pipe(len, isZero)
const res = isEmpty('hello')
```