```ts
export type Infer<FujiSchema> = FujiSchema extends Fuji<
  infer $RuleType,
  infer Value
>
  ? Value extends AnyShapeSchema
    ? InferRecord<Value>
    : Value extends Array<AnyShapeSchema>
    ? InferArrayOfRecords<Value>[]
    : RequiredType extends $RuleType
    ? NullableType extends $RuleType
      ? Value | null
      : Value
    : DefaultToType extends $RuleType
    ? Value
    : NullableType extends $RuleType
    ? Value | null
    : Value | undefined
  : never

type InferArrayOfRecords<Value extends Array<AnyShapeSchema>> = ...
type InferRecord<Shape extends Record<string, Fuji<any, any>>> = ...
type RequiredKeys<T extends AnyShapeSchema> = ...
type OptionalKeys<T extends AnyShapeSchema> = ...
```

```ts
type InferRecord<Shape extends Record<string, Fuji<any, any>>> = {
  [K in RequiredKeys<Shape>]: Infer<Shape[K]>
} & {
  [K in OptionalKeys<Shape>]?: Infer<Shape[K]>
}
```

```ts
type InferArrayOfRecords<Value extends
Array<AnyShapeSchema>> =
  Value extends Array<infer RecordValue>
    ? RecordValue extends AnyShapeSchema
      ? InferRecord<RecordValue>
      : never
    : never
```

```ts
type RequiredKeys<T extends AnyShapeSchema> = {
  [K in keyof T]: Rule<RequiredType, any> extends T[K]['rules'][number]
    ? K
    : Rule<DefaultToType, any> extends T[K]['rules'][number]
    ? K
    : never
}[keyof T]

type OptionalKeys<T extends AnyShapeSchema> = Exclude<keyof T, RequiredKeys<T>>
```

```
export type Infer<FujiSchema> = FujiSchema exter
  infer $RuleType,
  infer Value
>
  ? Value extends AnyShapeSchema
    ? InferRecord<Value>
    : Value extends Array<AnyShapeSchema>
    ? InferArrayOfRecords<Value>[]
    : RequiredType extends $RuleType
    ? NullableType extends $RuleType
      ? Value | null
      : Value
    : Default
    ? Value
    : Nullable
    ? Value |
    : Value |
  : never

type InferArra
type InferRec
type Required
type Optional
```

```
type InferRecord<Shape extends Record<string, Fuji<
  [K in RequiredKeys<Shape>]: Infer<Shape[K]>
} & {
  [K in OptionalKeys<Shape>]?: Infer<Shape[K]>
}
```

```
type InferArrayOfRecords<Value extends
Array<AnyShapeSchema>> =
  Value extends Array<infer RecordValue>
```

```
type RequiredKeys<T extends AnyShapeSchema> = {
  [K in keyof T]: Rule<RequiredType, any> extends T[K]['rules'][number]
    ? K
    : Rule<DefaultToType, any> extends T[K]['rules'][number]
    ? K
    : never
}[keyof T]

type OptionalKeys<T extends AnyShapeSchema> = Exclude<keyof T, RequiredKeys<T>>
```

Generics

Unions/Intersections

Conditional types

Type inference

keyof/typeof/infer Operators

Mapped types

Function overloads

Recursive types