Functors

15-150 Lec 2, Frank Pfenning
Lecture 17
Tuesday, March 24, 2020

Learning Objectives

Learning Objectives

- Why and how to parameterize modules

Learning Objectives

- Why and how to parameterize modules
- In SML

Learning Objectives

- Why and how to parameterize modules
- In SML
    - Expressing parameters in signatures

Learning Objectives

- Why and how to parameterize modules
- In SML
  - Expressing parameters in signatures
  - Instantiating signatures

Learning Objectives

- Why and how to parameterize modules
- In SML
    - Expressing parameters in signatures
    - Instantiating signatures
    - Type classes as certain signatures

Learning Objectives

- Why and how to parameterize modules
- In SML
  - Expressing parameters in signatures
  - Instantiating signatures
  - Type classes as certain signatures
  - Functors as parameterized structures

Review

Core Language	Module Level
Type	Signature
Expression	Structure
Function	Functor

Review

Review

- Abstract and concrete types in signatures

Review

- Abstract and concrete types in signatures
- Transparent and opaque signature ascription

Review

- Abstract and concrete types in signatures
- Transparent and opaque signature ascription
- Persistent data structures

Partial Summary

Partial Summary

- Every type in a signature is one of

Partial Summary

- Every type in a signature is one of
  - Concrete: implementation and client both know

Partial Summary

- Every type in a signature is one of
  - Concrete: implementation and client both know
  - Abstract: client does not know, implementation defines

Partial Summary

- Every type in a signature is one of
  - Concrete: implementation and client both know
  - Abstract: client does not know, implementation defines
  - Parameter: client defines, implementation doesn’t know

Module Constructs

Module Constructs

```
signature <sig> = sig … end
```

Module Constructs

```
signature <sig> = sig … end

<sig> where type <tp1> = <tp2>
```

Module Constructs

```
signature <sig> = sig … end

<sig> where type <tp1> = <tp2>

structure <str> :> <sig> = struct … end
```

Module Constructs

```
signature <sig> = sig … end

<sig> where type <tp1> = <tp2>

structure <str> :> <sig> = struct … end

functor <fctr> (<args>) :> <sig> = struct … end
```

Why Parameterize Modules?

Why Parameterize Modules?

- Client has an implementation that library needs
  - Example: `structure K : ORDERED`

Why Parameterize Modules?

- Client has an implementation that library needs
    - Example: `structure K : ORDERED`
- Library has an implementation that client needs
    - Example: `insert : ‘a dict * ‘a entry -> ‘a dict`

Why Parameterize Modules?

- Client has an implementation that library needs
  - Example: `structure K : ORDERED`
- Library has an implementation that client needs
  - Example: `insert : ‘a dict * ‘a entry -> ‘a dict`
- Unfortunately, the syntax is the same!

Summary

Core Language	Module Level
Type	Signature
Expression	Structure
Function	Functor