CS 476 – Programming Language Design

William Mansky

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.

OCaml: Variants

- Next step up from sum types: *variants*

```
inl “hi” : string + int      inr 123 : string + int
Text “hi” : [Text : string; Num : int; …]
```

- Can have any number of choices, and each one is named
- Choice (“field”) names are like OCaml constructors!

OCaml: Variants

L ::= … | <Ident> *L*
| (**match** *L* **with**
| <Ident> <ident> **->** *L*
| …
| <Ident> <ident> **->** *L*)

T ::= … | [<Ident> **of** *T*; …; <Ident> **of** *T*]

OCaml: Variants

$$\frac{\Gamma \vdash l : \tau_i}{\Gamma \vdash C_i\ l : [C_1 \texttt{ of } \tau_1; \ldots; C_i \texttt{ of } \tau_i; \ldots; C_n \texttt{ of } \tau_n]}$$

$$\frac{?}{\Gamma \vdash (\texttt{match}\ l\ \texttt{with}\ C_1\ x_1 \texttt{ -> } l_1 \mid \ldots \mid C_n\ x_n \texttt{ -> } l_n) : \tau}$$

OCaml: Variants

$$\frac{\Gamma \vdash l : \tau_i}{\Gamma \vdash C_i\ l : [C_1 \text{ of } \tau_1; \ldots; C_i \text{ of } \tau_i; \ldots; C_n \text{ of } \tau_n]}$$

$$\frac{\Gamma \vdash l : [C_1 \text{ of } \tau_1; \ldots; C_n \text{ of } \tau_n] \quad \Gamma[x_1 \mapsto \tau_1] \vdash l_1 : \tau \ \ \ldots\ \ \Gamma[x_n \mapsto \tau_n] \vdash l_n : \tau}{\Gamma \vdash (\texttt{match}\ l\ \texttt{with}\ C_1\ x_1\ \texttt{->}\ l_1 \mid \ldots \mid C_n\ x_n\ \texttt{->}\ l_n) : \tau}$$

OCaml: Variants

$C\ v$ is a value

$$\frac{l \rightarrow l'}{C\ l \rightarrow C\ l'}$$

$$\frac{}{(\texttt{match}\ l\ \texttt{with}\ C_1\ x_1\ \texttt{->}\ l_1\ |\ \ldots\ |\ C_n\ x_n\ \texttt{->}\ l_n) \rightarrow ?}$$

OCaml: Variants

$C\ v$ is a value

$$\frac{l \rightarrow l'}{C\ l \rightarrow C\ l'}$$

$$\frac{l \rightarrow l'}{(\texttt{match}\ l\ \texttt{with}\ C_1\ x_1\ \texttt{->}\ l_1 \mid \ldots \mid C_n\ x_n\ \texttt{->}\ l_n) \rightarrow (\texttt{match}\ l'\ \texttt{with}\ C_1\ x_1\ \texttt{->}\ l_1 \mid \ldots \mid C_n\ x_n\ \texttt{->}\ l_n)}$$

$$\frac{}{(\texttt{match}\ C_i\ v\ \texttt{with}\ C_1\ x_1\ \texttt{->}\ l_1 \mid \ldots \mid C_n\ x_n\ \texttt{->}\ l_n) \rightarrow [x_i \mapsto v]l_i}$$

OCaml: Variants

```
match lookup gamma x with
| Some t -> subtype c t
| None -> false
```

Exercise: If `lookup gamma x` returns `Some IntTy`, what steps will this program take?

$$\frac{l \to l'}{(\texttt{match}\ l\ \texttt{with}\ C_1\ x_1\ \texttt{->}\ l_1 \mid \ldots \mid C_n\ x_n\ \texttt{->}\ l_n) \to (\texttt{match}\ l'\ \texttt{with}\ C_1\ x_1\ \texttt{->}\ l_1 \mid \ldots \mid C_n\ x_n\ \texttt{->}\ l_n)}$$

$$\frac{}{(\texttt{match}\ C_i\ v\ \texttt{with}\ C_1\ x_1\ \texttt{->}\ l_1 \mid \ldots \mid C_n\ x_n\ \texttt{->}\ l_n) \to [x_i \mapsto v]l_i}$$

OCaml: Variants

```
match … with
| Some t -> subtype c t
| None -> false
```

$$\frac{l \rightarrow l'}{(\texttt{match}\ l\ \texttt{with}\ C_1\ x_1\ \texttt{->}\ l_1 \mid \dots \mid C_n\ x_n\ \texttt{->}\ l_n) \rightarrow (\texttt{match}\ l'\ \texttt{with}\ C_1\ x_1\ \texttt{->}\ l_1 \mid \dots \mid C_n\ x_n\ \texttt{->}\ l_n)}$$

$$\frac{}{(\texttt{match}\ C_i\ v\ \texttt{with}\ C_1\ x_1\ \texttt{->}\ l_1 \mid \dots \mid C_n\ x_n\ \texttt{->}\ l_n) \rightarrow [x_i \mapsto v]l_i}$$

OCaml: Variants

```
match Some IntTy with
| Some t -> subtype c t
| None -> false
```

$$\frac{l \to l'}{(\texttt{match}\ l\ \texttt{with}\ C_1\ x_1\ \texttt{->}\ l_1 \mid \ldots \mid C_n\ x_n\ \texttt{->}\ l_n) \to (\texttt{match}\ l'\ \texttt{with}\ C_1\ x_1\ \texttt{->}\ l_1 \mid \ldots \mid C_n\ x_n\ \texttt{->}\ l_n)}$$

$$\frac{}{(\texttt{match}\ C_i\ v\ \texttt{with}\ C_1\ x_1\ \texttt{->}\ l_1 \mid \ldots \mid C_n\ x_n\ \texttt{->}\ l_n) \to [x_i \mapsto v]l_i}$$

OCaml: Variants

```
match Some IntTy with
| Some t -> subtype c t
| None -> false
```

$$\frac{l \to l'}{(\texttt{match}\ l\ \texttt{with}\ C_1\ x_1\ \texttt{->}\ l_1 \mid \ldots \mid C_n\ x_n\ \texttt{->}\ l_n) \to (\texttt{match}\ l'\ \texttt{with}\ C_1\ x_1\ \texttt{->}\ l_1 \mid \ldots \mid C_n\ x_n\ \texttt{->}\ l_n)}$$

$$\frac{}{(\texttt{match}\ C_i\ v\ \texttt{with}\ C_1\ x_1\ \texttt{->}\ l_1 \mid \ldots \mid C_n\ x_n\ \texttt{->}\ l_n) \to [x_i \mapsto v]l_i}$$

OCaml: Variants

```
match Some IntTy with
| Some t -> subtype c t
| None -> false
```

$$\frac{l \to l'}{(\texttt{match}\ l\ \texttt{with}\ C_1\ x_1\ \texttt{->}\ l_1 \mid \ldots \mid C_n\ x_n\ \texttt{->}\ l_n) \to (\texttt{match}\ l'\ \texttt{with}\ C_1\ x_1\ \texttt{->}\ l_1 \mid \ldots \mid C_n\ x_n\ \texttt{->}\ l_n)}$$

$$\frac{}{(\texttt{match}\ C_i\ v\ \texttt{with}\ C_1\ x_1\ \texttt{->}\ l_1 \mid \ldots \mid C_n\ x_n\ \texttt{->}\ l_n) \to [x_i \mapsto v]l_i}$$

OCaml: Variants

```
match Some IntTy with
| Some t -> subtype c t
| None -> false
```

$\rightarrow$ `[t` $\mapsto$ `IntTy](subtype c t)`

$$\frac{l \rightarrow l'}{(\texttt{match}\ l\ \texttt{with}\ C_1\ x_1\ \texttt{->}\ l_1 \mid \ldots \mid C_n\ x_n\ \texttt{->}\ l_n) \rightarrow (\texttt{match}\ l'\ \texttt{with}\ C_1\ x_1\ \texttt{->}\ l_1 \mid \ldots \mid C_n\ x_n\ \texttt{->}\ l_n)}$$

$$\frac{}{(\texttt{match}\ C_i\ v\ \texttt{with}\ C_1\ x_1\ \texttt{->}\ l_1 \mid \ldots \mid C_n\ x_n\ \texttt{->}\ l_n) \rightarrow [x_i \mapsto v]l_i}$$

OCaml: Variants

```
match Some IntTy with
| Some t -> subtype c t  →  subtype c IntTy
| None -> false
```

$$\frac{l \rightarrow l'}{(\texttt{match}\ l\ \texttt{with}\ C_1\ x_1\ \texttt{->}\ l_1 \mid \ldots \mid C_n\ x_n\ \texttt{->}\ l_n) \rightarrow (\texttt{match}\ l'\ \texttt{with}\ C_1\ x_1\ \texttt{->}\ l_1 \mid \ldots \mid C_n\ x_n\ \texttt{->}\ l_n)}$$

$$\frac{}{(\texttt{match}\ C_i\ v\ \texttt{with}\ C_1\ x_1\ \texttt{->}\ l_1 \mid \ldots \mid C_n\ x_n\ \texttt{->}\ l_n) \rightarrow [x_i \mapsto v]l_i}$$

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.

OCaml: Variants

L ::= … | <Ident> *L*
| (`match` *L* `with`
| <Ident> <ident> `->` *L*
| …
| <Ident> <ident> `->` *L*)

T ::= … | [<Ident> `of` *T*; …; <Ident> `of` *T*]

OCaml: Inductive Datatypes

L ::= … | <Ident> *L* | match *L* with …

TD ::= (`type` <ident> `=` <Ident> `of` *T* | … | <Ident> `of` *T*)

T ::= … | <ident>

P ::= *TD* … *TD L*

```
type exp = Num of int | Add of exp * exp
```

- Every inductive type has a name, so constructors can take the type being declared as an argument!

OCaml: Inductive Datatypes

$C\ v$ is a value

$$\frac{l \to l'}{C\ l \to C\ l'}$$

$$\frac{l \to l'}{\begin{array}{c}(\texttt{match}\ l\ \texttt{with}\ C_1\ x_1\ \texttt{->}\ l_1 \mid \ldots \mid C_n\ x_n\ \texttt{->}\ l_n) \to \\ (\texttt{match}\ l'\ \texttt{with}\ C_1\ x_1\ \texttt{->}\ l_1 \mid \ldots \mid C_n\ x_n\ \texttt{->}\ l_n)\end{array}}$$

$$\frac{}{(\texttt{match}\ C_i\ v\ \texttt{with}\ C_1\ x_1\ \texttt{->}\ l_1 \mid \ldots \mid C_n\ x_n\ \texttt{->}\ l_n) \to [x_i \mapsto v]l_i}$$

OCaml: Inductive Datatypes

- We can store type definitions in the type context Γ
- Γ maps variables to types, type names to definitions

$$\frac{\text{lookup\_constr}(\Gamma, C) = (\texttt{type}\ t = C\ \texttt{of}\ \tau\ |\ \ldots) \qquad \Gamma \vdash l : \tau}{\Gamma \vdash C\ l : t}$$

$$\frac{\Gamma \vdash l : [C_1\ \texttt{of}\ \tau_1; \ldots; C_n\ \texttt{of}\ \tau_n] \quad \Gamma[x_1 \mapsto \tau_1] \vdash l_1 : \tau\ \ \ldots\ \ \Gamma[x_n \mapsto \tau_n] \vdash l_n : \tau}{\Gamma \vdash (\texttt{match}\ l\ \texttt{with}\ C_1\ x_1\ \texttt{->}\ l_1\ |\ \ldots\ |\ C_n\ x_n\ \texttt{->}\ l_n) : \tau}$$

OCaml: Inductive Datatypes

- l has to be of a variant type
- The cases of the match should be the cases of l's type
- Each case should return something of type τ
  - where each case's variable gets the type of the constructor

$$\frac{\Gamma \vdash l : [C_1 \text{ of } \tau_1; \ldots; C_n \text{ of } \tau_n] \qquad \Gamma[x_1 \mapsto \tau_1] \vdash l_1 : \tau \;\; \ldots \;\; \Gamma[x_n \mapsto \tau_n] \vdash l_n : \tau}{\Gamma \vdash (\texttt{match}\; l \;\texttt{with}\; C_1\; x_1 \texttt{ -> } l_1 \mid \ldots \mid C_n\; x_n \texttt{ -> } l_n) : \tau}$$

OCaml: Inductive Datatypes

- l has to be of an inductive type
- The cases of the match should be the cases of l's type
- Each case should return something of type τ
  - where each case's variable gets the type of the constructor

$$\frac{\Gamma \vdash l : t \quad \Gamma(t) = (C_1 \texttt{ of } \tau_1 \mid \ldots \mid C_n \texttt{ of } \tau_n) \quad \Gamma[x_1 \mapsto \tau_1] \vdash l_1 : \tau \ \ldots \ \Gamma[x_n \mapsto \tau_n] \vdash l_n : \tau}{\Gamma \vdash (\texttt{match } l \texttt{ with } C_1\ x_1 \texttt{ -> } l_1 \mid \ldots \mid C_n\ x_n \texttt{ -> } l_n) : \tau}$$

Inductive Datatypes: Summary

- In OCaml (and most other functional languages), we can define datatypes with cases, and match on those cases
- Each datatype has a list of *constructors*, which build values of the type and are the patterns we match on
- Sum types: two constructors
- Variant types: any number of constructors, not recursive
- Inductive types: any number of constructors, recursive!
  - Because datatypes have names (*nominal types*), they can take instances of the same type as arguments

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.

PL Courses Next Semester

- CS 472: Provably Correct Programming
  - Logic, functional programming, proving programs correct
- CS 473: Compiler Design
  - Lexing and parsing, translation to assembly
- CS 474: Object-Oriented Languages and Environments
  - Much deeper study (without inference rules) of OO language features

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.