CS 476 – Programming Language Design

William Mansky

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.

Typed Lambda Calculus with Recursion

L ::= <ident> | λ(<ident>: *T*). *L* | *L* *L* | <#> | *L* **+** *L*

  | `ifzero` *L* `then` *L* `else` *L* | `let rec` <ident> : *T* = *L* `in` *L*

T ::= int | *T* → *T*

Exercise: What features of OCaml does this language not have yet?

From Typed Lambda Calculus to OCaml

- User-friendly syntax
- Basic types, tuples, records
- Inductive datatypes and pattern-matching
- Local declarations
- References
- Type inference
- Generics/polymorphism

Typed Lambda Calculus with Recursion

L ::= <ident> | λ(<ident>: T). L | L L | <#> | L **+** L

  | `ifzero` L `then` L `else` L | `let rec` <ident> : T `=` L `in` L

T ::= int | $T \to T$

Simple OCaml

L ::= <ident> | `fun` (<ident> : *T*) `->` *L* | *L* *L* | <#> | *L* `+` *L*

  | `ifzero` *L* `then` *L* `else` *L* | `let rec` <ident> : *T* `=` *L* `in` *L*

T ::= int | *T* → *T*

Simple OCaml: Tuples

$L ::= \ldots \mid (L, L) \mid \texttt{fst}\ L \mid \texttt{snd}\ L$

$T ::= \ldots \mid T * T$

$$\frac{\Gamma \vdash l_1 : \tau_1 \quad \Gamma \vdash l_2 : \tau_2}{\Gamma \vdash (l_1, l_2) : \tau_1 * \tau_2}$$

$$\frac{\Gamma \vdash l : \tau_1 * \tau_2}{\Gamma \vdash \texttt{fst}\ l : \tau_1}$$

$$\frac{\Gamma \vdash l : \tau_1 * \tau_2}{\Gamma \vdash \texttt{snd}\ l : \tau_2}$$

Simple OCaml: Tuples

L ::= … | (L, L) | fst L | snd L

T ::= … | $T * T$

(v_1, v_2) is a value

$$\frac{l_1 \rightarrow l_1'}{(l_1, l_2) \rightarrow (l_1', l_2)} \qquad \frac{l_2 \rightarrow l_2'}{(l_1, l_2) \rightarrow (l_1, l_2')}$$

$$\frac{}{\texttt{fst}\,(v_1, v_2) \rightarrow v_1} \qquad \frac{}{\texttt{snd}\,(v_1, v_2) \rightarrow v_2}$$

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.

Simple OCaml: Records

```
type person = { name : string; age : int; id : int };;
let a = { name = “Alice”; age = 22; id = 123456 };;
```

- Nominal (“named”) type: the type of **`a`** is **`person`**
- Structural type: the type of **`a`** is
  `{ name : string; age : int; id : int }`

Simple OCaml: Records

L ::= … | { <ident> = *L*; …; <ident> = *L* } | *L*.<ident>

T ::= … | { <ident> : *T*; …; <ident> : *T* }

$$\frac{\Gamma \vdash l_1 : \tau_1 \quad \dots \quad \Gamma \vdash l_n : \tau_n}{\Gamma \vdash \{f_1 = l_1; \dots; f_n = l_n\} : \{f_1 : \tau_1, \dots, f_n : \tau_n\}}$$

$$\frac{\Gamma \vdash l : \{f_1 : \tau_1, \dots, f_n : \tau_n\}}{\Gamma \vdash l.f_i : \tau_i}$$

Simple OCaml: Records

L ::= … | { <ident> = *L*; …; <ident> = *L* } | *L*.<ident>

T ::= … | { <ident> : *T*; …; <ident> : *T* }

$\{f_1 = v_1; \ldots; f_n = v_n\}$ is a value

$$\frac{l_i \rightarrow l_i'}{\{f_1 = l_1; \ldots; f_i = l_i; \ldots; f_n = l_n\} \rightarrow \{f_1 = l_1; \ldots; f_i = l_i'; \ldots; f_n = l_n\}}$$

$$\frac{}{\{f_1 = v_1; \ldots; f_i = v_i; \ldots; f_n = v_n\}.f_i \rightarrow v_i}$$

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.

From Typed Lambda Calculus to OCaml

- User-friendly syntax
- Basic types, tuples, records
- ➢ Inductive datatypes and pattern-matching
- Local declarations
- References
- Type inference
- Generics/polymorphism

OCaml*: Sum Types

```
int + bool
(* like type int+bool = inl int | inr bool *)

inl 3 : int + bool
inr true : int + bool

match (a : int + bool) with
| inl i -> i + 1
| inr b -> if b then 1 else 0
```

OCaml*: Sum Types

L ::= … | `inl` *L* | `inr` *L*

  | `(match` *L* `with` `inl` <ident> `->` *L* | `inr` <ident> `->` *L*`)`

T ::= … | *T* + *T*

OCaml*: Sum Types

$$\frac{\Gamma \vdash l : \tau_1}{\Gamma \vdash \texttt{inl}\ l : \tau_1 + \tau_2} \qquad \frac{\Gamma \vdash l : \tau_2}{\Gamma \vdash \texttt{inr}\ l : \tau_1 + \tau_2}$$

$$\frac{?}{\Gamma \vdash (\texttt{match}\ l\ \texttt{with}\ \texttt{inl}\ x_1 \texttt{->}\ l_1 \mid \texttt{inr}\ x_2 \texttt{->}\ l_2) : \tau}$$

Exercise: How should we typecheck a match statement?

OCaml*: Sum Types

$$\frac{\Gamma \vdash l : \tau_1}{\Gamma \vdash \texttt{inl}\ l : \tau_1 + \tau_2}$$

$$\frac{\Gamma \vdash l : \tau_2}{\Gamma \vdash \texttt{inr}\ l : \tau_1 + \tau_2}$$

$$\frac{\Gamma \vdash l : \tau_1 + \tau_2 \quad \Gamma[x_1 \mapsto \tau_1] \vdash l_1 : \tau \quad \Gamma[x_2 \mapsto \tau_2] \vdash l_2 : \tau}{\Gamma \vdash (\texttt{match}\ l\ \texttt{with inl}\ x_1\ \texttt{->}\ l_1\ |\ \texttt{inr}\ x_2\ \texttt{->}\ l_2) : \tau}$$

OCaml*: Sum Types

$\mathrm{inl}\ v$ and $\mathrm{inr}\ v$ are values

$$\frac{l \rightarrow l'}{\texttt{inl}\ l \rightarrow \texttt{inl}\ l'} \qquad \frac{l \rightarrow l'}{\texttt{inr}\ l \rightarrow \texttt{inr}\ l'}$$

$$\frac{}{(\texttt{match}\ l\ \texttt{with}\ \texttt{inl}\ x_1\ \texttt{->}\ l_1\ |\ \texttt{inr}\ x_2\ \texttt{->}\ l_2) \rightarrow ?}$$

OCaml*: Sum Types

$$\frac{l \rightarrow l'}{\begin{array}{c}(\texttt{match}\ l\ \texttt{with inl}\ x_1\ \texttt{->}\ l_1\ \texttt{|}\ \texttt{inr}\ x_2\ \texttt{->}\ l_2) \rightarrow \\ (\texttt{match}\ l'\ \texttt{with inl}\ x_1\ \texttt{->}\ l_1\ \texttt{|}\ \texttt{inr}\ x_2\ \texttt{->}\ l_2)\end{array}}$$

$$\frac{}{(\texttt{match inl}\ v\ \texttt{with inl}\ x_1\ \texttt{->}\ l_1\ \texttt{|}\ \texttt{inr}\ x_2\ \texttt{->}\ l_2) \rightarrow l_1}$$

$$\frac{}{(\texttt{match inr}\ v\ \texttt{with inl}\ x_1\ \texttt{->}\ l_1\ \texttt{|}\ \texttt{inr}\ x_2\ \texttt{->}\ l_2) \rightarrow l_2}$$

OCaml*: Sum Types

$$\frac{l \rightarrow l'}{(\texttt{match}\ l\ \texttt{with inl}\ x_1\ \texttt{->}\ l_1 \mid \texttt{inr}\ x_2\ \texttt{->}\ l_2) \rightarrow (\texttt{match}\ l'\ \texttt{with inl}\ x_1\ \texttt{->}\ l_1 \mid \texttt{inr}\ x_2\ \texttt{->}\ l_2)}$$

$$\frac{}{(\texttt{match inl}\ v\ \texttt{with inl}\ x_1\ \texttt{->}\ l_1 \mid \texttt{inr}\ x_2\ \texttt{->}\ l_2) \rightarrow [x_1 \mapsto v]l_1}$$

$$\frac{}{(\texttt{match inr}\ v\ \texttt{with inl}\ x_1\ \texttt{->}\ l_1 \mid \texttt{inr}\ x_2\ \texttt{->}\ l_2) \rightarrow [x_2 \mapsto v]l_2}$$

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.

HW6 Overview

- Interpreter for a simple functional language
- Ignore the definition of substitution: it’s complicated, but for our purposes it just works