CSCI-344
Programming Language Concepts (Section 3)

Lecture 25

Hindley-Milner Type System

Instructor: Hossein Hojjat

November 9, 2016

Where we are

Done:

- Type Inference Examples

This session:

- Formalize Type Inference

Formalizing Type Inference

Hindley-Milner Type System

- **Sad news:** Full type inference for polymorphism is undecidable
- **Solution:** A restricted form of polymorphism:

  Parameters can only have monomorphic types

- **Consequence:** Polymorphic functions are not first class citizens

Example

Variables bound with `let` may be polymorphic

```
-> (letrec ((f (lambda (x) x))) (pair (f #t) (f 1)))
(PAIR #t 1) : (pair bool int)
```

but function arguments may not be:

```
-> (val-rec f (lambda (id) (begin (id 10) (id #t))))
type error: cannot make int equal to bool
```

- Type abstractions are introduced at `let` binding

Hindley-Milner type system: Quantifiers

- Quantifiers cannot be nested inside type expressions
  - $\forall\alpha.\alpha \to \forall\alpha.\alpha$ is not allowed
- Type Scheme: $\forall\alpha_1 \cdots \alpha_n.\ \tau$ where no quantifiers occur in τ
- Allowing quantifiers in τ makes it undecidable

Hindley-Milner types

```
datatype ty
  = TYCON  of name                    (* int, pair, ... *)
  | CONAPP of ty * ty list            (* (int) list, ...*)
  | TYVAR  of name                    (* 'a, 'b, ...*)

datatype type_scheme
  = FORALL of name list * ty
```

Typing Judgment

- $\Gamma \vdash e : \tau$ Given the type environment Γ, expression e can be used at type τ

- Type environment Γ binds variables to type schemes
- `car`: $\forall \alpha.\ \alpha\ \texttt{list} \rightarrow \alpha$
- `cdr`: $\forall \alpha.\ \alpha\ \texttt{list} \rightarrow \alpha\ \texttt{list}$
- `pair`: $\forall \alpha, \beta.\ (\alpha\ \ \beta) \rightarrow (\texttt{pair}\ \ \alpha\ \ \beta)$

Instantiation

- Typed μScheme: explicit instantiation of polymorphic values

```
-> (@ '() int)
() : (list int)
-> (@ '() bool)
() : (list bool)
```

- Nano-ML: system automatically instantiates polymorphic values
- Both judgments are valid in nano-ML:
  - $\Gamma \vdash \mathrm{LITERAL(NIL)}$: `int list`
  - $\Gamma \vdash \mathrm{LITERAL(NIL)}$: `bool list`

Instantiation

$\tau' <: \tau$

- τ' is an instance of τ
- τ is more general than τ'

Examples:

- `int` $<:$ `'`a
- `int list` $<:$ `'`a
- `int list` $<:$ `'`a `list`
- not `int` $<:$ `'`a `list`

Instantiation

- Instantiation of variable's type scheme:

$$\frac{\Gamma(x) = \delta \qquad \tau <: \delta}{\Gamma \vdash x : \tau} \text{ (Var)}$$

- Any instance $\tau$of δ is acceptable

Instances and substitution

- τ_1 is an instance of τ_2 ($\tau_1 <: \tau_2$) if and only if there exists a substitution θ such that $\tau_1 = \theta(\tau_2)$
- A substitution is a map from type variables to types

$$\{\alpha_1 \mapsto A_1, \alpha_2 \mapsto A_2, \cdots, \alpha_n \mapsto A_n\}$$

- **Theorem:** 'a is the most general type
- **Proof:** let $\theta = \{\alpha \to \tau\}$

More Instantiation

- Another example:

```
(val f (lambda (x, y) (- y 1)))
(val g (lambda (z) (f 0 z)))
```

- In application `(f 0 z)`

$$\texttt{f} : \forall \texttt{'a.}\ \texttt{'a int} \to \texttt{int}$$

$$\texttt{(0 z)} : \forall \texttt{'b.}\ \texttt{int 'b}$$

- We should instantiate both `f` and `z` correctly
- Application is clearly OK, assuming that

$$\texttt{'a} \mapsto \texttt{int}, \texttt{'b} \mapsto \texttt{int}$$

- Finding the right substitution is **unification**

Unification

- To make the idea precise, we say that:
- τ_1 and τ_2 are unified by θ, if $\theta\tau_1 = \theta\tau_2$
- `'a * int` and `int * 'b` are unified by
  $\{$`'a` $\mapsto$ `int`, `'b` $\mapsto$ `int`$\}$

- Unifier is not unique, `'a` and `'b list` are unified by
  - $\{$`'a` $\mapsto$ `'b list`$\}$
  - $\{$`'a` $\mapsto$ `bool list list`, `'b` $\mapsto$ `bool list`$\}$

- Most general unifier of types τ_1 and τ_2 is a substitution θ such that
  1. τ_1 and τ_2 are unified by θ $\quad(\theta\tau_1 = \theta\tau_2)$
  2. there is no more general θ that unifies θ_1 and θ_2

Summary

This Lecture

- Hindley-Milner Type System

Next Lecture

- Type Inference Rules