
# CSCI-344
# Programming Language Concepts (Section 3)

---

Lecture 24

Type Inference

Instructor: Hossein Hojjat

November 7, 2016

# Where we are

**Done:**

- Static Typing
- Typed Impcore: Monomorphic type system
- Typed $\mu$Scheme: Polymorphic type system

**This session:**

- Type Inference

# Nano-ML

- Nano-ML: type-safe bridge language which is derived from $\mu$Scheme
- Programmer never writes explicit type annotations
- Nano-ML uses type inference
- Type of every parameter, variable, and function is inferred

**Nano-ML vs. $\mu$Scheme**

- Different type systems
- No mutation in nano-ML
    - No assignment to variables, no override on existing binding
- `val-rec` in nano-ML allows recursive calls to itself

# Type Inference Example

## $\mu$Scheme

```
-> (define double (x) (+ x x))
double
```

## Typed $\mu$Scheme

```
-> (define int double ([x : int]) (+ x x))
double : (int -> int)
```

## Nano-ML

```
-> (define double (x) (+ x x))
double : (int -> int)
```

# Type Inference Example

The compiler infers types: there is lower annotation burden

**Typed $\mu$Scheme**

```
-> ((@ cons int) 1 ((@ cons int) 1 (@ '() int)))
(1 1) : (list int)
```

**Nano-ML**

```
-> (cons 1 (cons 1 '()))
(1 1) : (list int)
```

# Type Inference

## Type Inference Problem

- **Input:** term with no type annotations
- **Output:** valid typing for term, or, show that it has no valid typing

## Key Steps

1. Assign a preliminary type to every sub-expression
   - Use fresh type variables to represent unknown types
2. Generate equality constraints among the types and type variables
3. Solve the constraints to compute a typing

## Undecidability

- Undecidable for a language as powerful as Typed $\mu$Scheme

# Generate Constraints

Constraint generation for function application and definition

**Application**

- If type of `f` is $A$, type of `e` is $B$, type of `(f e)` is $C$, then $A = B \to C$

$$\underbrace{\begin{array}{cc} A & B \\ \downarrow & \downarrow \\ \texttt{(f} & \texttt{e)} \end{array}}_{C} \qquad A = B \to C$$

**Definition**

- If type of `f` is $A$, type of `x` is $B$, type of `e` is $C$, then $A = B \to C$

$$\begin{array}{ccc} A & & B\ C \\ \downarrow & & \downarrow\ \downarrow \\ \multicolumn{3}{l}{\text{(val f (lambda (x) e))}} \end{array} \qquad A = B \to C$$

# Example 1

```
-> (val-rec double (lambda (x) (+ x x)))
double : (int -> int)
```

# Example 1

```
-> (val-rec double (lambda (x) (+ x x)))
double : (int -> int)
```

**Step 1:** Assign preliminary types

| Sub-expression | Preliminary Type |
|---|---|
| `double` | $A$ |
| `x` | $B$ |
| `(+ x x)` | $C$ |
| `+` | `int` $\rightarrow$ `int` $\rightarrow$ `int` |

# Example 1

```
-> (val-rec double (lambda (x) (+ x x)))
double : (int -> int)
```

**Step 1:** Assign preliminary types

| Sub-expression | Preliminary Type |
|---|---|
| `double` | $A$ |
| `x` | $B$ |
| `(+ x x)` | $C$ |
| `+` | `int` $\rightarrow$ `int` $\rightarrow$ `int` |

**Step 2:** Generate constraints

$$\texttt{int} \rightarrow \texttt{int} \rightarrow \texttt{int} = B \rightarrow B \rightarrow C$$
$$A = B \rightarrow C$$

# Example 1

```
-> (val-rec double (lambda (x) (+ x x)))
double : (int -> int)
```

**Step 1:** Assign preliminary types

| Sub-expression | Preliminary Type |
|---|---|
| `double` | $A$ |
| `x` | $B$ |
| `(+ x x)` | $C$ |
| `+` | `int` $\rightarrow$ `int` $\rightarrow$ `int` |

**Step 2:** Generate constraints

$$\texttt{int} \rightarrow \texttt{int} \rightarrow \texttt{int} = B \rightarrow B \rightarrow C$$
$$A = B \rightarrow C$$

**Step 3:** Solve constraints

$$A = \texttt{int} \rightarrow \texttt{int} \ , \ B = \texttt{int} \ , \ C = \texttt{int}$$

# Example 2

```
-> (val-rec f (lambda (x) (if x 1 0)))
?
```

# Example 2

```
-> (val-rec f (lambda (x) (if x 1 0)))
 f :  (bool -> int)
```

**Step 1:** Assign preliminary types

| Sub-expression | Preliminary Type |
|---|---|
| `f` | $A$ |
| `x` | $B$ |
| `(if x 1 0)` | $C$ |
| `if` | $\forall\alpha.\,\texttt{bool} \to \alpha \to \alpha \to \alpha$ |
| `0 , 1` | `int` |

**Step 2:** Generate constraints

$$\texttt{bool} \to \alpha_1 \to \alpha_1 \to \alpha_1 \quad = \quad B \to \texttt{int} \to \texttt{int} \to C$$

$$A \quad = \quad B \to C$$

**Step 3:** Solve constraints

$$A = \texttt{bool} \to \texttt{int} \;,\; B = \texttt{bool} \;,\; C = \alpha_1 = \texttt{int}$$

# Examples on the Board

```
-> (val f (lambda (x) (if x x (+ 1 x))))
?
```

# Examples on the Board

```
-> (val f (lambda (x) (if x x (+ 1 x))))
 type error:  cannot make int equal to bool
```

# Examples on the Board

```
-> (val f (lambda (x) (if x x (+ 1 x))))
 type error:  cannot make int equal to bool
```

```
-> (val f (lambda (x f) (f x)))
?
```

# Examples on the Board

```
-> (val f (lambda (x) (if x x (+ 1 x))))
 type error:  cannot make int equal to bool
```

```
-> (val f (lambda (x f) (f x)))
 f :  (forall ('a 'b) ('a -> ('a -> 'b) -> 'b))
```

# Examples on the Board

```
-> (val f (lambda (x) (if x x (+ 1 x))))
 type error:  cannot make int equal to bool
```

```
-> (val f (lambda (x f) (f x)))
 f :  (forall ('a 'b) ('a -> ('a -> 'b) -> 'b))
```

```
-> (val f (lambda (xs) (car (car xs))))
?
```

# Examples on the Board

```
-> (val f (lambda (x) (if x x (+ 1 x))))
 type error:  cannot make int equal to bool
```

```
-> (val f (lambda (x f) (f x)))
 f :  (forall ('a 'b) ('a -> ('a -> 'b) -> 'b))
```

```
-> (val f (lambda (xs) (car (car xs))))
 f:  (forall 'a .  'a list list -> 'a)
```

# Summary

**This Lecture**

- Type Inference: Example

**Next Lecture**

- Formalizing Type Inference