Lab #4. F- Type System

Prof. Jaeseung Choi

Dept. of Computer Science and Engineering

Sogang University

Remind: About the Labs

- **We have done four lab assignments so far**
  - Labs will count for 40% of the total score in semester
  - **Lab #1:** Warm-up Exercise (4%)
  - **Lab #2:** Interpreter for Imperative Language (8%)
  - **Lab #3:** Interpreter for Functional Language (8%)
  - **Lab #4: Type Inference (20%)**
- **This lab assignment (Lab #4) will take a large portion in your total score**

General Information

- **Check the *Assignment* tab of *Cyber Campus***
  - Skeleton code (**`Lab4.tgz`**) is attached together with this slide
  - Submission will be accepted in the same post, too
- **Deadline: 6/26 Wednesday 23:59**
  - Late submission deadline: **6/28 Friday 23:59 (-20% penalty)**
  - Delay penalty is applied uniformly **(not problem by problem)**
- **Please read the instructions in this slide carefully**
  - This slide is a step-by-step tutorial for the lab
  - It also contains important submission guidelines
    - If you do not follow the guidelines, **you will get penalty**

Skeleton Code Structure

- **Copy `Lab4.tgz` into CSPRO server and decompress it**
  - This course will use cspro2.sogang.ac.kr (**don't miss the 2**)
  - Don't decompress-and-copy; copy-and-decompress
- **`FMinusType`: Directory for static type system on `F-`**
- **`check.py`, `config`: Script and config file for self-grading (same as before)**

```
jschoi@cspro2:~$ tar -xzf Lab4.tgz
jschoi@cspro2:~$ cd Lab4/
jschoi@cspro2:~/Lab4$ ls
FMinusType  check.py  config
```

Directory Structure of `FMinusType`

- **Skeleton code structure of `src/` has slightly changed**
  - **`AST.fs`:** Syntax definition of the **`F-`** language
  - **`TypeSystem.fs`:** You have to **implement a static type system** for **`F-`** language here
  - **`Main.fs`:** Main driver code of the type inferer
  - **`Lexer.fsl`, `Parser.fsy`**: Parser (you don't have to care)
- **Do NOT fix any source files other than `TypeSystem.fs`**

```
jschoi@cspro2:~/Lab4$ cd FMinusType/
jschoi@cspro2:~/Lab4/FMinusType$ ls
FMinusType.fsproj  src  testcase
jschoi@cspro2:~/Lab4/FMinusType$ ls src
AST.fs  Lexer.fsl  Main.fs  Parser.fsy  TypeSystem.fs
```

F- Language Syntax

- **Same to the previous lab for F- interpreter**
  - The whole program is an expression

$$
\begin{aligned}
E \rightarrow\ & n \\
& \mid \textbf{true} \mid \textbf{false} \\
& \mid x \\
& \mid -E \\
& \mid E + E \mid E - E \\
& \mid E < E \mid E > E \mid E = E \mid E <> E \\
& \mid \textbf{if } E \textbf{ then } E \textbf{ else } E \\
& \mid \textbf{let } x = E \textbf{ in } E \\
& \mid \textbf{let } f\ x = E \textbf{ in } E \\
& \mid \textbf{let rec } f\ x = E \textbf{ in } E \\
& \mid \textbf{fun } x \rightarrow E \\
& \mid E\ \ E
\end{aligned}
$$

Expression

F- Language Typing Rules

- **$\Gamma \vdash e : t$ means that e is assigned type t in our system**
  - We use the same type domain with our lecture slide

$$\frac{}{\Gamma \vdash n : \text{int}} \qquad \frac{}{\Gamma \vdash \textbf{true} : \text{bool}} \qquad \frac{}{\Gamma \vdash \textbf{false} : \text{bool}} \qquad \frac{}{\Gamma \vdash x : \Gamma(x)}$$

$$\frac{\Gamma \vdash e : \text{int}}{\Gamma \vdash -e : \text{int}} \qquad \frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 + e_2 : \text{int}} \qquad \frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 - e_2 : \text{int}}$$

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 < e_2 : \text{bool}} \qquad \frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 > e_2 : \text{bool}}$$

$$\frac{\Gamma \vdash e_1 : t \quad \Gamma \vdash e_2 : t}{\Gamma \vdash e_1 = e_2 : \text{bool}} \; t = \text{bool} \vee t = \text{int} \qquad \frac{\Gamma \vdash e_1 : t \quad \Gamma \vdash e_2 : t}{\Gamma \vdash e_1 <> e_2 : \text{bool}} \; t = \text{bool} \vee t = \text{int}$$

F - Language Typing Rules

- **$\Gamma \vdash e : t$ means that e is assigned type t in our system**
  - We use the same type domain with our lecture slide

$$\frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : t \quad \Gamma \vdash e_3 : t}{\Gamma \vdash \textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3 : t}$$

$$\frac{\Gamma \vdash e_1 : t_1 \quad \Gamma[x \mapsto t_1] \vdash e_2 : t_2}{\Gamma \vdash \textbf{let}\ x = e_1\ \textbf{in}\ e_2 : t_2}$$

$$\frac{\Gamma[x \mapsto t_a] \vdash e_1 : t_r \quad \Gamma[f \mapsto (t_a \to t_r)] \vdash e_2 : t_2}{\Gamma \vdash \textbf{let}\ f\ x = e_1\ \textbf{in}\ e_2 : t_2}$$

$$\frac{\Gamma[x \mapsto t_a][f \mapsto (t_a \to t_r)] \vdash e_1 : t_r \quad \Gamma[f \mapsto (t_a \to t_r)] \vdash e_2 : t_2}{\Gamma \vdash \textbf{let rec}\ f\ x = e_1\ \textbf{in}\ e_2 : t_2}$$

$$\frac{\Gamma[x \mapsto t_a] \vdash e : t_r}{\Gamma \vdash \textbf{fun}\ x \to e : t_a \to t_r}$$

$$\frac{\Gamma \vdash e_1 : t_a \to t_r \quad \Gamma \vdash e_2 : t_a}{\Gamma \vdash e_1\ \ e_2 : t_r}$$

Implementing Type System

- **You must implement an algorithm to infer the type of an `F-` program, based on the previous typing rules**
  - Definition of **`Type`** and **`TypeEnv`** are provided in **`TypeSystem.fs`**
  - You can fix these types if needed, but it's your responsibility to ensure that the whole project **correctly compiles and runs**
  - Raise **`TypeError`** if the input program seems to have type error

```
exception TypeError

type Type =
  | Int
  | Bool
  | TyVar of string
  | Func of Type * Type

type TypeEnv = Map<string, Type>
```

Implementing Type System

- **You must implement an algorithm to infer the type of an `F-` program, based on the previous typing rules**
  - Your mission it to complete **`Type.infer()`** function
  - **`Type.toString()`** function is already provided for you
  - The driver code in **`Main.fs`** will call **`Type.infer()`** and **`Type.toString()`** to print the output of type inference

```
exception TypeError

type Type = ...
...

module Type =
  let rec toString (typ: Type): string = ... // Provided

  let infer (prog: Program) : Type = ... // TODO
```

Challenge

- **In the previous typing rules, you might have noticed that `=` and `<>` operators are overloaded**
  - Ex) `if (true = false) then (1 = 2) else (3 <> 3)` is a valid `F-` program with type **`bool`**
  - Implementing these rules can be a little bit challenging
- **Think about how to modify the type inference algorithm that we have discussed in the lecture slide**
  - Once you find a good direction, it only requires a slight change

$$\frac{\Gamma \vdash e_1 : t \quad \Gamma \vdash e_2 : t}{\Gamma \vdash e_1 = e_2 : \text{bool}} \; t = \text{bool} \vee t = \text{int}$$

$$\frac{\Gamma \vdash e_1 : t \quad \Gamma \vdash e_2 : t}{\Gamma \vdash e_1 <> e_2 : \text{bool}} \; t = \text{bool} \vee t = \text{int}$$

Assumption

- **Recall that for some `F-` programs, the type is not uniquely decided by our type system**
  - Ex) `fun x -> x`, `fun x -> (fun y -> x = y)`, …
  - I will **not** use such programs as test cases for the grading
  - In other words, all the test cases for grading will have uniquely decided type (or must be rejected by the type system)

$$\frac{\dots}{\phi \vdash \textbf{fun}\ x \to x : \text{bool} \to \text{bool}} \qquad \frac{\dots}{\phi \vdash \textbf{fun}\ x \to x : \text{int} \to \text{int}} \qquad \frac{\dots}{\phi \vdash \textbf{fun}\ x \to x : {'\text{a}} \to {'\text{a}}}$$

$$\frac{\dots}{\phi \vdash \textbf{fun}\ x \to (\textbf{fun}\ y \to x = y) : \text{bool} \to (\text{bool} \to \text{bool})}$$

$$\frac{\dots}{\phi \vdash \textbf{fun}\ x \to (\textbf{fun}\ y \to x = y) : \text{int} \to (\text{int} \to \text{bool})}$$

Building and Testing

- **In `testcase` directory, `tc-*` and `ans-*` files are provided**
  - After compiling the project with **`dotnet build -o out`** command, you can run type inference on the programs written in **`F-`**
  - It will **print the inferred type of the input program** (or report a type error)

```
jschoi@cspro2:~/Lab4/FMinusType$ cat testcase/tc-1
let f x = x + 1 in
...
jschoi@cspro2:~/Lab4/FMinusType$ dotnet build -o out
...
jschoi@cspro2:~/Lab4/FMinusType$ ./out/FMinusType testcase/tc-1
int
```

This result must match with the content of **`ans-1`** (expected output)

Self-Grading Script

- **If you think that your code prints correct outputs for all the test cases, run `check.py` as a final check**
  - '**O**': Correct, '**X**': Incorrect, '**E**': Unhandled exception in your code
  - '**C**': Compile error, '**T**': Timeout (maybe infinite recursion)
- **If you correctly raise `TypeError` exception for a program with type error, it will be graded as 'O' (not 'E')**
  - If you raise **`TypeError`** for a valid program, it is **'X'**

```
jschoi@cspro2:~/Lab4$ ls
FMinusType  check.py  config
jschoi@cspro2:~/Lab4$ $ ./check.py
[*] FMinusType : OOOO
```

Actual Grading

- **I will use different test case set during the real grading**
  - So you are encouraged to run you code with your own additional test cases (try to think of various inputs)
  - Some students ask me to provide more test cases, but **it is important to practice this on your own**
  - Especially in this lab, there are many tricky cases to consider
  - To get a high score, you must create high-quality test cases
- **You will get the point based on the number of test cases that you pass**
  - 20 test cases, **5 point** per test case (100 point in total)
  - But you will get **-1 point** if your answer is wrong

Submission Guideline

- **You should submit only one F# source code file**
  - **`TypeSystem.fs`**
- **If the submitted file fails to compile with skeleton code when I type "`dotnet build`", cannot give you any point**
- **Submission format**
  - Upload this file directly to *Cyber Campus* (**do not zip them**)
  - **Do not change the file name** (e.g., adding any prefix or suffix)
  - If your submission format is wrong, you will get **-20% penalty**