COMP3048: Lecture 11

Contextual Analysis: Implementing a Type Checker

Matthew Pike

University of Nottingham, Ningbo

This Lecture (and the next)

Step by step development of a type checker for LTXL:

- LTXL abstract syntax
- LTXL types
- Informal typing rules for LTXL
- Formal typing rules for LTXL
- Additional infrastructure (handout)
- Implementing the type checker (interactively)

LTXL Abstract Syntax

LTXL example program, ***concrete*** syntax:

```
let int x = 7; int y = 5 in x * y + 7
```

Typing rule/handwriting friendly version of the LTXL abstract syntax:

$e \rightarrow$	n	*literal integer*
$\mid$	x	*variable*
$\mid$	$\ominus\ e$	*unary operator app.*
$\mid$	$e \otimes e$	*binary operator app.*
$\mid$	$\texttt{if}\ e\ \texttt{then}\ e\ \texttt{else}\ e$	*conditional expression*
$\mid$	$\texttt{let}\ (T\ x\ =\ e)^*\ \texttt{in}\ e$	`let`-*expression*

LTXL AST Representation (recap)

```
type Id = String

data Exp
    = LitInt    Int
    | Var       Id
    | UnOpApp   UnOp Exp
    | BinOpApp  BinOp Exp Exp
    | If        Exp Exp Exp
    | Let       [(Id, Type, Exp)] Exp
```

LTXL Types

LTXL type syntax:

$$
\begin{array}{rcll}
T & \rightarrow & \texttt{int} & \textit{integer type} \\
 & | & \texttt{bool} & \textit{boolean type} \\
 & | & \texttt{(}T\texttt{,}T\texttt{)} & \textit{product (pair)} \\
 & | & T \rightarrow T & \textit{function}
\end{array}
$$

LTXL Type Representation

The following Haskell data type is used to represent LTXL types:

```
data Type = TpUnknown
          | TpBool
          | TpInt
          | TpProd Type Type    -- pair
          | TpArr  Type Type    -- function
          deriving Eq
```

LTXL Operator Types

Unary LTXL operator types:

$$\texttt{\textbackslash} \; : \; \texttt{bool} \to \texttt{bool}$$

$$\texttt{-} \; : \; \texttt{int} \to \texttt{int} \qquad \textit{unary minus}$$

Binary LTXL operator types:

$$\texttt{||}, \texttt{\&\&} \; : \; (\texttt{bool}\,,\,\texttt{bool}) \to \texttt{bool}$$

$$\texttt{<}, \texttt{==}, \texttt{>} \; : \; (\texttt{int}\,,\,\texttt{int}) \to \texttt{bool}$$

$$\texttt{+}, \texttt{-}, \texttt{*}, \texttt{/} \; : \; (\texttt{int}\,,\,\texttt{int}) \to \texttt{int}$$

LTXL Operator Representation

```
data UnOp = Not | Neg

data BinOp = Or
           | And
           | Less
           | Equal
           | Greater
           | Plus
           | Minus
           | Times
           | Divide
```

Example: An LTXL Program

The LTXL example program again:

```
let int x = 7; int y = 5 in x * y + 7
```

Representation:

```
Let [("x", IntType, LitInt 7),
     ("y", IntType, LitInt 5)]
    (BinOpApp Plus
              (BinOpApp Times
                        (Var "x")
                        (Var "y"))
              (LitInt 7))
```

LTXL Typing Rules (1)

The LTXL expression typing relation is a ***ternary*** (or ***trinary***) ***relation***:

$$\Gamma \vdash e : T$$

Read: expression e has type T in type environment Γ

1. A literal integer has type **`int`**.

   $\Gamma \vdash n : \texttt{int}$ (T-LITINT)

2. A variable (or operator) has whatever type it is declared to have.

   $$\frac{x : T \in \Gamma}{\Gamma \vdash x : T}$$ (T-VAR)

LTXL Typing Rules (2)

3. The types of the argument(s) to a unary or binary operator must match the type(s) of the formal parameters of the operator.

4. The result type of a unary or binary operator application is the result type of the operator.

$$\frac{\Gamma \vdash \ominus : T_1 \rightarrow T_2 \quad \Gamma \vdash e_1 : T_1}{\Gamma \vdash \ominus\ e_1 : T_2} \qquad \text{(T-UNOPAPP)}$$

$$\frac{\Gamma \vdash \otimes : (T_1\ ,\ T_2) \rightarrow T_3 \quad \Gamma \vdash e_1 : T_1 \quad \Gamma \vdash e_2 : T_2}{\Gamma \vdash e_1 \otimes e_2 : T_3} \qquad \text{(T-BINOPAPP)}$$

Exercise: LTXL Typing Rules

Let us use the rules we have seen thus far to type check the program

```
x + 3
```

in the environment:

$$\Gamma_1 = \begin{array}{l} \texttt{+} : (\texttt{int}\ ,\ \texttt{int}) \rightarrow \texttt{int}, \\ \texttt{*} : (\texttt{int}\ ,\ \texttt{int}) \rightarrow \texttt{int}, \\ \texttt{x} : \texttt{int} \end{array}$$

(On whiteboard)

LTXL Typing Rules (3)

5. The type of the condition in a conditional expression must be **`bool`**.
6. The two branches of a conditional expression must have the same type.

$$\frac{\Gamma \vdash e_1 : \texttt{bool} \quad \Gamma \vdash e_2 : T \quad \Gamma \vdash e_3 : T}{\Gamma \vdash \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 : T} \quad \text{(T-IF)}$$

LTXL Typing Rules (4)

7. The declared type of a variable must match the type of the defining expression.

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \qquad \text{(T-VAR)}$$

$$\frac{\Gamma \vdash \overline{e_1} : \overline{T_1} \quad \Gamma, \overline{x} : \overline{T_1} \vdash e : T}{\Gamma \vdash \mathbf{\texttt{let}}\ \overline{T_1}\ \overline{x} \ = \ \overline{e_1}\ \mathbf{\texttt{in}}\ e : T} \qquad \text{(T-LET)}$$

All LTXL Typing Rules

$$\Gamma \vdash n : \texttt{int} \qquad \text{(T-LITINT)}$$

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \qquad \text{(T-VAR)}$$

$$\frac{\Gamma \vdash \ominus : T_1 \rightarrow T_2 \quad \Gamma \vdash e_1 : T_1}{\Gamma \vdash \ominus\, e_1 : T_2} \qquad \text{(T-UNOPAPP)}$$

$$\frac{\Gamma \vdash \otimes : \texttt{(} T_1 \texttt{,} T_2 \texttt{)} \rightarrow T_3 \quad \Gamma \vdash e_1 : T_1 \quad \Gamma \vdash e_2 : T_2}{\Gamma \vdash e_1 \otimes e_2 : T_3} \qquad \text{(T-BINOPAPP)}$$

$$\frac{\Gamma \vdash e_1 : \texttt{bool} \quad \Gamma \vdash e_2 : T \quad \Gamma \vdash e_3 : T}{\Gamma \vdash \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 : T} \qquad \text{(T-IF)}$$

$$\frac{\Gamma \vdash \overline{e_1} : \overline{T_1} \quad \Gamma, \overline{x} : \overline{T_1} \vdash e : T}{\Gamma \vdash \texttt{let}\ \overline{T_1}\ \overline{x}\ = \ \overline{e_1}\ \texttt{in}\ e : T} \qquad \text{(T-LET)}$$

Modified LTXL Scope Rules

1. The scope of a variable is ***only*** the body of the `let`-expression in which the definition of the variable occurs. (Implied by T-LET.)
2. A definition of a variable hides, for the extent of its scope, any definition of a variable with the same name from an outer `let`-expression.
3. At most one definition may be given for a variable in the list of definitions of a `let`-expression.

LTXL Type Environment

A suitable environment implementation is given. These operations enforce scope rules 2 and 3.

```
type VarAttr = (Int, Type)

data Env -- Abstract


initEnv   :: [(Id, Type)] -> [(UnOp, Type)]
             -> [(BinOp, Type)] -> Env
enterVar  :: Id -> Int -> Type -> Env
             -> Either Env String
lookupVar :: Id -> Env -> Either VarAttr String
lookupUO  :: UnOp -> Env -> Type
lookupBO  :: BinOp -> Env -> Type
```

Exercise (for home)

The original first LTXL scope rule read:

1. The scope of a variable is ***all subsequent definitions and the body*** of the `let`-expression in which the definition of the variable occurs. A variable is ***not*** in scope in the RHS of its definition.

Suggest a version of T-LET that corresponds to this rule, and then change the LTXL implementation correspondingly.

Type-Checking Utilities

```
compatible :: Type -> Type -> Bool
compatible TpUnknown _         = True
compatible _         TpUnknown = True
compatible t1        t2        = t1 == t2

illTypedOpApp :: Type -> Type -> String
illTypedCond :: Type -> String
incompatibleBranches :: Type -> Type -> String
declMismatch :: Type -> Type -> String

emitErrD :: SrcPos -> String -> D ()
```