

COMP3012/G53CMP: Lecture 10

Contextual Analysis: Implementing a Type Checker

Henrik Nilsson

University of Nottingham, UK

COMP3012/G53CMP: Lecture 10 - p.1/19

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

```

COMP3012/G53CMP: Lecture 10 - p.4/19

LTXL Operator Types

Unary LTXL operator types:

```

\ : bool → bool
- : int → int  unary minus

```

Binary LTXL operator types:

```

||, && : (bool, bool) → bool
<, ==, > : (int, int) → bool
+, -, *, / : (int, int) → int

```

COMP3012/G53CMP: Lecture 10 - p.7/19

This Lecture

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)

COMP3012/G53CMP: Lecture 10 - p.2/19

LTXL Types

LTXL type syntax:

```

T → int  integer type
   | bool  boolean type
   | (T, T)  product (pair)
   | T → T  function

```

COMP3012/G53CMP: Lecture 10 - p.5/19

LTXL Operator Representation

```

data UnOp = Not | Neg

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

```

COMP3012/G53CMP: Lecture 10 - p.8/19

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 → n          literal integer
   | x          variable
   | ⊖ e        unary operator app.
   | e ⊗ e      binary operator app.
   | if e then e else e  conditional expression
   | let (T x = e)* in e  let-expression

```

COMP3012/G53CMP: Lecture 10 - p.3/19

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

```

COMP3012/G53CMP: Lecture 10 - p.6/19

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))

```

COMP3012/G53CMP: Lecture 10 - p.9/19

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**.

$$\Gamma \vdash n : \mathbf{int} \quad (\text{T-LITINT})$$
2. A variable (or operator) has whatever type it is declared to have.

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

COMP3012/GS3CMP: Lecture 10 - p.15/19

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.

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

COMP3012/GS3CMP: Lecture 10 - p.15/19

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.

COMP3012/GS3CMP: Lecture 10 - p.15/19

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.

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

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

COMP3012/GS3CMP: Lecture 10 - p.15/19

LTXL Typing Rules (4)

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

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

$$\frac{\Gamma \vdash \bar{e}_1 : \bar{T}_1 \quad \Gamma, \bar{x} : \bar{T}_1 \vdash e : T}{\Gamma \vdash \mathbf{let } \bar{T}_1 \bar{x} = \bar{e}_1 \mathbf{ in } e : T} \quad (\text{T-LET})$$

COMP3012/GS3CMP: Lecture 10 - p.14/19

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
```

COMP3012/GS3CMP: Lecture 10 - p.17/19

Exercise: LTXL Typing Rules

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

`x + 3`

in the environment:

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

(On whiteboard)

COMP3012/GS3CMP: Lecture 10 - p.15/19

All LTXL Typing Rules

$$\Gamma \vdash n : \mathbf{int} \quad (\text{T-LITINT})$$

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

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

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

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

$$\frac{\Gamma \vdash \bar{e}_1 : \bar{T}_1 \quad \Gamma, \bar{x} : \bar{T}_1 \vdash e : T}{\Gamma \vdash \mathbf{let } \bar{T}_1 \bar{x} = \bar{e}_1 \mathbf{ in } e : T} \quad (\text{T-LET})$$

COMP3012/GS3CMP: Lecture 10 - p.15/19

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.

COMP3012/GS3CMP: Lecture 10 - p.15/19

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 ()
```