INF400 Homework II
Type Inference Rules Report

November 26, 2025

Contents

1 Introduction

This report formally defines the type inference rules for the semantic analysis phase of the Kiraz programming language compiler using turnstile notation. Kiraz is a statically typed language where all type errors are caught at compile time.

1.1 Features of the Kiraz Language

The main features of the Kiraz language are:

- Static type system
- Purely static scoping
- Inner scopes inherit parent scopes
- Forward declaration support

1.2 Builtin Types

- `Integer64`: 64-bit integer
- `String`: String type
- `Boolean`: Boolean type
- `Void`: Void return type

1.3 Builtin Functions

- `and(Boolean, Boolean) : Boolean`
- `or(Boolean, Boolean) : Boolean`
- `not(Boolean) : Boolean`

2 Type Inference Rules

2.1 Rule 1: Function Scope

$$\frac{\Gamma \vdash T_1, \ldots, T_n : \text{Type} \quad \Gamma \vdash R : \text{Type} \quad \Gamma, a_1 : T_1, \ldots, a_n : T_n \vdash \text{body}}{\Gamma \vdash \texttt{func}\ f(a_1 : T_1, \ldots, a_n : T_n) : R\{\text{body}\} : \texttt{Function}} \tag{1}$$

Description: A function is valid if:

1. All argument types are defined
2. Return type is defined
3. Function body is valid
4. Function name is unique

Constraints:

- Argument names must be unique
- Argument names must differ from function name

2.2 Rule 2: If Statement

$$\frac{\Gamma \vdash e : \texttt{Boolean} \quad \Gamma \vdash s_1 : \tau_1 \quad \Gamma \vdash s_2 : \tau_2}{\Gamma \vdash \texttt{if } (e)\{s_1\} \texttt{ else } \{s_2\} : \texttt{Void}} \tag{2}$$

Description: An if statement is valid if:

1. Test expression is of Boolean type
2. Then block is valid
3. Else block is valid
4. Inside Func or Method scope

Constraints:

- Test must be Boolean only
- Cannot be used in Module or Class scope

2.3 Rule 3: While Statement

$$\frac{\Gamma \vdash e : \texttt{Boolean} \quad \Gamma \vdash s : \tau}{\Gamma \vdash \texttt{while } (e)\{s\} : \texttt{Void}} \tag{3}$$

Description: A while loop is valid if:

1. Test expression is of Boolean type
2. Loop body is valid
3. Inside Func or Method scope

Constraints:

- Test must be Boolean only
- Cannot be used in Module or Class scope

2.4 Rule 4: Logic Builtins

2.4.1 And Operator

$$\frac{\Gamma \vdash e_1 : \texttt{Boolean} \quad \Gamma \vdash e_2 : \texttt{Boolean}}{\Gamma \vdash \texttt{and}(e_1, e_2) : \texttt{Boolean}} \tag{4}$$

2.4.2 Or Operator

$$\frac{\Gamma \vdash e_1 : \texttt{Boolean} \quad \Gamma \vdash e_2 : \texttt{Boolean}}{\Gamma \vdash \texttt{or}(e_1, e_2) : \texttt{Boolean}} \tag{5}$$

2.4.3 Not Operator

$$\frac{\Gamma \vdash e : \texttt{Boolean}}{\Gamma \vdash \texttt{not}(e) : \texttt{Boolean}} \tag{6}$$

Description: Logical operators:

1. Only accept Boolean type
2. Return Boolean
3. Cannot be overridden
4. Automatically defined in module scope

2.5 Rule 5: Let Statement

2.5.1 Initializer Only

$$\frac{\Gamma \vdash e : T \quad x \notin \Gamma}{\Gamma \vdash \texttt{let}\ x = e : T} \tag{7}$$

2.5.2 Type Only

$$\frac{\Gamma \vdash T : \texttt{Type} \quad x \notin \Gamma}{\Gamma \vdash \texttt{let}\ x : T : T} \tag{8}$$

2.5.3 Type and Initializer

$$\frac{\Gamma \vdash T : \texttt{Type} \quad \Gamma \vdash e : T \quad x \notin \Gamma}{\Gamma \vdash \texttt{let}\ x : T = e : T} \tag{9}$$

Description: A let statement is valid if:

1. Type is defined
2. Initializer type matches
3. Variable name starts with lowercase
4. Variable has not been defined before

Special Rules:

- null can only be used in let initialization
- Control flow constructs cannot be assigned

3 Additional Rules

3.1 Arithmetic Operators

$$\frac{\Gamma \vdash e_1 : T \quad \Gamma \vdash e_2 : T \quad T \in \{\texttt{Integer64}, \texttt{String}\}}{\Gamma \vdash e_1 + e_2 : T} \tag{10}$$

Addition operator is defined for Integer64 and String.

3.2 Comparison Operators

$$\frac{\Gamma \vdash e_1 : \texttt{Integer64} \quad \Gamma \vdash e_2 : \texttt{Integer64}}{\Gamma \vdash e_1 \sim e_2 : \texttt{Boolean}} \tag{11}$$

Where $\sim \in \{<, >, \leq, \geq, ==, \neq\}$

3.3 Assignment

$$\frac{\Gamma \vdash x : T \quad \Gamma \vdash e : T}{\Gamma \vdash x = e : T} \tag{12}$$

For assignment, left and right side types must match.

3.4 Function Call

$$\frac{\Gamma \vdash f : (T_1, \ldots, T_n) \to R \quad \Gamma \vdash e_i : T_i}{\Gamma \vdash f(e_1, \ldots, e_n) : R} \tag{13}$$

For function calls, argument count and types must match.

4 Conclusion

In this report, the semantic analysis rules of the Kiraz language have been formally defined using turnstile notation. Key points:

- Static type system catches all errors at compile-time
- Forward declaration support is available
- Scope rules are strict
- Builtins cannot be overridden