Programming Languages 2024/25

Final Submission Report

David Nunes - Ist102934

I. Static Type-Checker

ASTTypes

Because static type checking operates on types rather than values, a new construct ASTType was introduced to represent the type system of the language. This abstraction captures the variety of types expressible in the language. The following specialized ASTType variants were implemented:

- ASTTArrow – Describes a Lambda function type as ArgumentType → FunctionBodyType
- ASTTBool – Describes a boolean
- ASTTId – Describes an Identifier
- ASTTInt – Describes an Integer
- ASTTList – Describes a List (Eager and Lazy) with a certain ASTType
- ASTTRef – Describes a box() reference to another ASTType
- ASTTString – Describes a String
- ASTTStruct – Describes a product type (structs)
- ASTTUnion – Describes a sum type (union)
- ASTTUnit – Describes a unit (null/void)

ASTNode

All AST nodes have been extended to include a typecheck() method. This method mirrors the structure of the eval() function but focuses exclusively on computing and verifying types. It accepts an environment mapping identifiers to their corresponding ASTType, ensuring correct type associations throughout the program.

The typecheck() method performs a static traversal of the AST, determining the type of each node and checking type safety without executing any code. This ensures early detection of type errors.

To support additional language features introduced in the type system, the following AST node types were added:

- ASTMatchUnion – Match construct for Union variables
- ASTString – Creation of Strings (Also comes with a new VString value)
- ASTStruct – Creation of product types (structs) (Also comes with a new VStruct value)
- ASTUnion – Creation of sum types (union) (Also comes with a new VUnion value)
- ASTUnit – Creation of unit's (Also comes with a new ASTUnit value)
- ASTTypeDef – Defining types associated with an identifier

Type Bindings

To handle field labeling in structures and variant labeling in unions, a new class named TypeBindList was introduced. This class manages the association between labels and their corresponding ASTTypes.

II. Recursive Types

Handling recursive types requires a careful approach to avoid infinite loops during type resolution. When a recursive type is defined, the interpreter initially registers the type of each label using their associated identifiers (Ids) without immediately resolving these identifiers to their underlying ASTTypes. This deferred resolution is essential because the identifiers may refer back to the type being defined, either directly or indirectly.

Later, when the recursive type is used in the AST, the interpreter performs a lazy resolution of these identifiers—resolving each Id only at the point where its concrete type information is actually needed. This demand-driven approach ensures that type resolution halts as soon as sufficient information is obtained, preventing unnecessary or infinite expansions of self-referential types.

By deferring translation and resolving only as needed, the interpreter remains efficient and avoids getting trapped in recursive loops while still correctly typing recursive structures.

END