Compilers

INF-400

Burak Arslan
ext-inf400@burakarslan.com

Galatasaray Üniversitesi

Lecture IX
2023-12-14

Course website

burakarslan.com/inf400

Types in Kiraz (cont'd)

Semantic Analysis

Types in Kiraz

Two categories of types:

- Builtins (primitives plus special types like `Module`, `Function` etc.)
- User-defined types (classes, defined in terms of builtins)

Semantic Analysis

Types in Kiraz

Type checking is two-stage process:

- Propagating the types
- Verifying the types and associated operations

Semantic Analysis

Types in Kiraz

Type checking formalism:

- Just simple logic with some fancy notation
- Translated to C++ manually

Semantic Analysis

Types in Kiraz

If e_1 has type `Integer64` and e_2 has type `Integer64`,

- then $e_1 + e_2$ has type `Integer64`

becomes:

$$(e_1 : \text{Integer64} \land e_2 : \text{Integer64}) \Rightarrow e_1 + e_2 : \text{Integer64}$$

Semantic Analysis

Types in Kiraz

... which is called an inference rule:

$$\text{Hypothesis}_1 \wedge \cdots \wedge \text{Hypothesis}_n \Rightarrow \text{Conclusion}$$

Semantic Analysis

Types in Kiraz

With a more compact notation:

$$\frac{\vdash H_1 \cdots \vdash H_n}{C}$$

eg.

$$\frac{\vdash e_1 : \texttt{Integer64} \quad \vdash e_2 : \texttt{Integer64}}{\vdash e_1 + e_2 : \texttt{Integer64}}$$

($\vdash$: It is provable that ...)

Semantic Analysis

Types in Kiraz

$$\dfrac{\dfrac{\vdash 1 \text{ is an int. lit.}}{\vdash 1 : \texttt{Integer64}} \qquad \dfrac{\vdash 2 \text{ is an int. lit.}}{\vdash 2 : \texttt{Integer64}}}{\vdash 1 + 2 : \texttt{Integer64}}$$

(You may notice that the structure is the same as the AST)

Semantic Analysis

Types in Kiraz

Note that hypotheses are optional:

$$\frac{}{\vdash \mathsf{let}\ \mathsf{x} : \mathsf{Type} : \mathsf{Type}}$$

Semantic Analysis

Types in Kiraz

You need to cover every single **corner case**.

During the language design phase, a considerable amount of time is spent on dealing with corner cases.

Semantic Analysis

Types in Kiraz

You won't always be successful!

C/C++ is full of undefined/unspecified/implementation defined behavior. eg:

```
int a = 5;
int b = (++a + a++); // UB!
```

Semantic Analysis

Types in Kiraz

An example from kiraz:

$$\frac{\vdash \; e_1 : \texttt{Boolean} \; e_2 : \texttt{StatementList}}{\vdash \text{while } e_1 \text{ do } e_2 : \texttt{Void}}$$

Any statement whose type computes to `Void` can exist in isolation, but can't be moved around

Or could it?

Semantic Analysis

Type Environment

Let's look at the following code fragment:

```
let a = x + y;
```

Where does the type information come from?

Semantic Analysis

Type Environment

The compiler needs to maintain a separate type environment in addition to the symbol table to remember _*previously seen stuff*_

Semantic Analysis

Type Environment

Type environment is:

- A function that maps identifiers to types.
- See the `get_symbol(SymbolTable &)` and `get_subsymbol(SymbolTable &)` functions in the kiraz codebase

Semantic Analysis

Type Environment

Some properties of the type environment:

- The type environment gives types to the free identifiers in the current scope
- The type environment is passed down the AST from the root towards the leaves
- Types are computed up the AST from the leaves towards the root