CS323 Project Progress Report

The Compiler of a Functional Programming Language

01.07.2024

Zhezhen Cao Fan Club

1 Language Design

Overview of the Design

“*Most functional languages* ***are very similar****, and* ***vary largely in syntax****.*”

– Simon L. Peyton Jones《函数式编程语言的实现》

We represent the following things:

- **Defining functions**

```
defn f x = {x + 1}
```

- **Declaring data types**

```
data List = { Nil, Cons Int List }
// data type List is either 'Nil' or 'Cons Int List'
```

- **Applying fuctions**

```
f 10
```

Overview of the Design (ii)

- **Arithmetic**
  - $+, -, \times, \div, \%$
  - bitwise operations
  - boolean operations
- **Algebraic data types**
  - list: `[0, 1, 2,]`
  - tuple: `(0, 1, 2,)`
- **Pattern matching** (to operate on data types)

```
case l of {
  Nil -> { 0 }
  Cons x xs -> { x }
}
```

Lexical Specification

1. Arithmetic operators
   - $+, -, \times, \div, \%, \ldots$
2. Basic data value:
   - Integer: `[0-9]+`
   - Float: `[0-9]*\.[0-9]+([eE][-+]?[0-9]+)? | [0-9]+[eE][-+]?[0-9]+`
   - String: `\"([^\\\"\n]|\\.)*\"`
3. Reserved words:
   - Function and data type definition: `defn` and `data`
   - Pattern matching: `case` and `of`
   - Boolean value: `True` and `False`
   - Data types: `Int`, `String`, `Float` and `Bool`
4. Identifiers:
   - LID: `[a-z][a-zA-Z_]*`
   - UID: `[A-Z][a-zA-Z_]*`
5. Sematic symbols: `{`, `}`, `(`, `)`, `[`, `]`, `,`, `->`, `do` and `=`

Syntax Specification: Function Definition

Basically, the program is a set of function definitions.

Definition can be written as follows:

```
defn LID lowercaseParams = { aAdd }
```

where `LID` is a identifier starts with lowercase letter. `lowercaseParams` are parameters which consists of zero or more `LID`(s) seperated by `whitespace`. `aAdd` is an expression.

Example

```
defn f a b = { a + b }
```

Syntax Specification: Data Definition

We can also define data types.

```
data UID = { uppercaseParams, ... }
```

where `UID` is a identifier starts with uppercase letter. `uppercaseParams` are parameters which consists of zero or more `UID`(s) seperated by `whitespace`.

Example

```
data List = { Nil, Cons Int List }
```

Syntax Specification: Pattern Matching

Pattern matching can be used to operate on data types.

```
case aAdd of {
  pattern -> { aAdd }
  ...
  pattern -> { aAdd }
}
```

where `pattern` is `LID` or `UID lowercaseParams`.

Example

```
case l of {
  Nil -> { 0 }
  Cons x xs -> { x }
}
```

Syntax Specification: Tuple and List

Tuple can be written as

```
(aAdd, aAdd, ...,)
```

List can b e written as

```
[aAdd, aAdd, ...,]
```

Refer to Python, we decide to use `,)` and `,]` as the right close token of tuple and list, respectively.

Example

```
(0,)
[0, 1, 2,]
```