iCalendar Language

Mengfei Ren(mr3258)
Chang Hu(ch2950)
Yu Kang(yk2540)
JiaCheng Chen(jc3940)
Yun Feng(yf2294)

Motivation

- Fast-paced modern life.
- Build our own event models.
- Manage events in our calendar.

Introduction To iCalendar Language

- Simple language for event and calendar processing.
  - Simple, sparse syntax
  - C- like structure
- Provide simple structure for programmer to manipulate event and calendar.

iCalendar Language Example

```
Event myEve{
        int time;
        string name;
        string place;
}

void main(){
        myEve e1 = [19,"plt presentation","cs building"];
        print(e1.time);
}
```

Features of iCalendar

- Event and Calendar as two primary data types.
  - Full set of operators provided for building event and store it in calendar.

Language Tutorial

-iCalendar Data Types

- Int
- Float
- String
- Bool
- Void
- Event_type
- Calendar

```
let string_of_dt = function
        Int ->"int"
        | Float ->"float"
        | String ->"String"
        | Boolean ->"bool"
        | Void ->"void"
        | Event_type(myEvent) -> myEvent
        | Calendar-> "Calendar"
```

Language Tutorial

-iCalendar Expressions

- Literal

  IntLit(i), FloatLit(f),

  BoolLit(b), StringLit(s)
- Id
- Binop and Uniop
- Assign
- Call
- Noexpr
- ObjValue

  myEvent e =[“2012”, “CS”];

  Calendar c = [e1,e2,e3];

Language Tutorial

-iCalendar Statements

Statement:

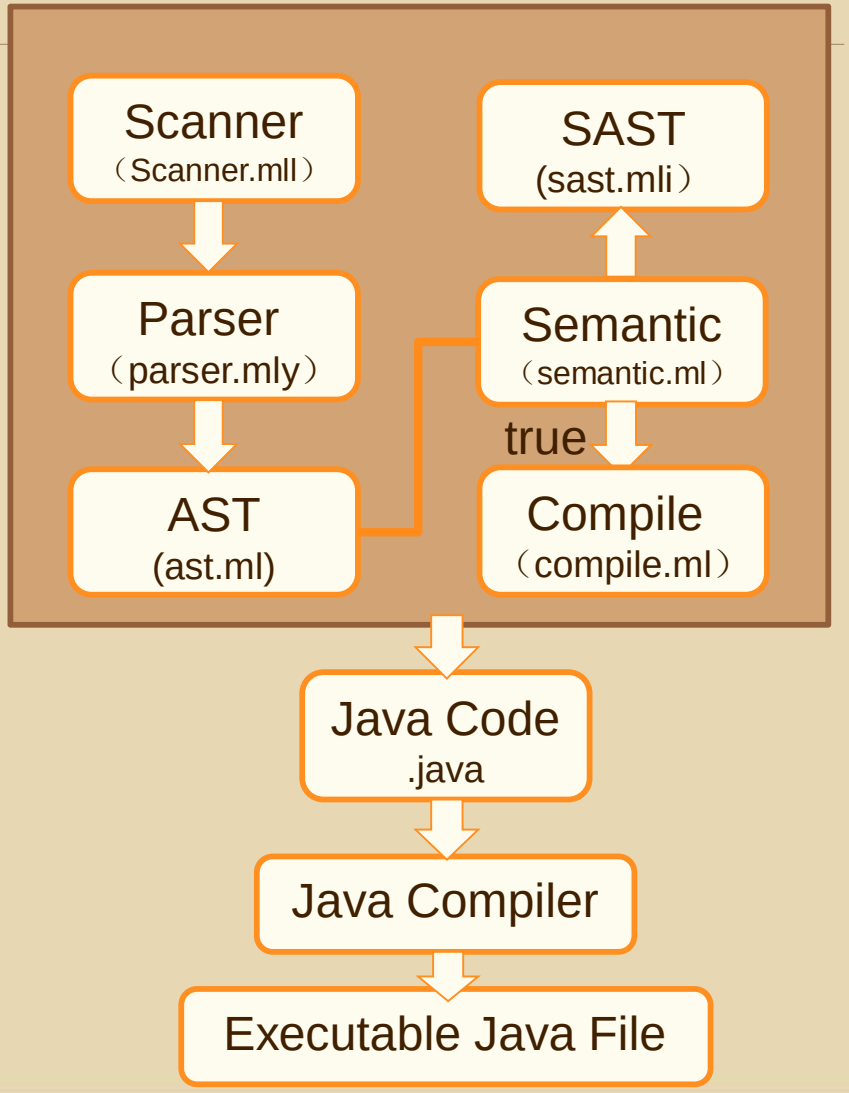
- Block → stmt list
- Expr → expression
- Return → expr
- ReturnVoid
- If → if(expr * stmt) * {stmt}
- For →for(expr * expr * expr)* {stmt}
- While → while(expr) * {stmt}
- Vardecl → var_decl
- Empty

Program:

- Event define list
- Global Variable declarations
- Function list

Language Implementation

- Scanner
  - Recognizes language tokens
- Parser
  - Consumes tokens and validates program in syntactically correct
- AST
  - Generated with parsing
- Semantic analysis
  - Semantic check according to AST and generate SAST
- Java Code Generator
  - Generate corresponding Java Code on AST



Language Implementation

```
type symbol_table = {
  parent : symbol_table option;
  mutable vars : (t * string) list;
  mutable funcs : (t * string * (t list)) list;
  mutable events : event_table list;
  is_loop : bool
}
```

```
type event_table = {
  type_name : string;
  member_list : (t * string) list
}
```

iCalendar Language Result

- After the implementation, we write two tests about our language.
  - The gcd function
  - The event function

Lessons Learned

- At the beginning, we designed a language called iChemi. However, we found two problems later:
  - The chemical formula could not be expressed correctly, even though we thought out some ways, but we could not get its molecules after parser
  - We thought about the language incorrectly at the beginning, just mixed up the user and compiler

Lessons Learned

- Start early, even though it is hard at the beginning.
- Test with the compiler after each file.
- Keep things simple. More restrictive syntax, more semantic analysis.

Thank You!