Language Processing system

1 What is language processing system 04/11

- interpreter and compiler
  - difference and advantage
- what process is followed in the compiler
- lexical analzer and syntax analyzer
  - programming language must be constructed
- regular expression and powerset construction
- Brzozowski's derivatice
  - better way to construct DFA

2 Lexical analyzer and parsing tree 04/18

Today I learnt the rule and flow of the lexical analyzer and the basic of the parsing tree.

- Lexical analyzer categorize a written program into some peaces
  - "if", "x", "string" and so on
- There are two rules in a lexical analyzer
  - longest match
  - first match
- Context free language and parsing tree
- Ambiguity of the grammer
  - the way to solve the ambiguity

3 Parsing tree 04/25

- Buttom-up parsing and Top-down parsing
- CKY parsing algorithm
  - search match from length 1
  - Advantage : van apply to all context free language
  - Disadvantage : slow ($O(n^3)$)

- Naive top-down parsing
  - Advantage: easy to understand
  - Disadvantage: need backtrack
  - Disadvantage: vulnerable to left recursion
- LL parser
  - look ahead

Listing 1 basic idea

```
parseS() =
  match lookahead2() with
    |aa -> parseA(); parseC()
    |ab -> parseB(); parseD()
    |_ -> error()
```

  - In order to realize, we need Nulls, and FIRST and FOLLOW
    - NULL is a function that generates ϵ^*
    - FIRST is a function that generates language that starts with "a"
    - FOLLOW is a function that generates language that "a" follows after X
  - To generate LL Table is an easier way to analyze the language
  - fixed point theorem
    - In order to calculate Null, FIRSR and FOLLOW

4 LR parser 05/10

LR parser is said to be one of the important topics in this lecture because LR parser is used in many sytems. LR parser can interpret more languages than LL parser.

- Two processes
  - shift: $(\Gamma, aw) \to (\Gamma a, w)$
  - reduce: $(\Gamma X_1 \dots X_k, w) \to (\Gamma A, w) \quad if A \to X_1 \dots X_k$
- The problem is that which shoulb be applied, shift or reduce
- Judge which should be applied from the value of the stack
- We create an automaton that represents the rules depending on the stack
  - $q_0 = closure(\{S \to .\alpha\})$
  - $\delta(q, X) = closure(\{A \to \alpha X.\beta | A \to \alpha.X\beta \in q\})$
  - "." represents a point that parser sees now
- We can also create a table from the automaton

5 Improvement of the LR parser 05/17

Today's lecture is about the improvement of the LR parser and how to create parse tree in the program. LALR, which is stated last of the class, are most commonly used.

- SLR
  - control reduce using Follow set
  - When use reduce, A $\rightarrow$ a must be included in the LR(0) and next c must be included in the Follow set
- LR(1)
  - more qualified Follow set which specified by the information about the parent node
- LALR(1)
  - in order to reduce the number of conditnions, merge some states
  - Merged states has the states in common but next signitures are different
- How to use ocamlyacc

6 semantic analysis and type judgement 05/24

- What semantic analysis does
  - detect mistakes that cannot be found in a parser
  - correct data that will need in the later processes
    - reference relation
    - revise overrode
- type judgement
  - Type judgement algorithms are created based on the type rule
  - Type judgement are executed by making a tree (it's like a proof tree)

7 Mid-term exam 05/31

8 How interpreter works 06/06

Today's lecture is about the action of interpreter after parsing. The evaluation function is called recursively passing the environment. There are two ways to define functions. One is called static scope and the other is called dynamic scope. Most of the language used now is based on the static scope.

9 intermediate representation 06/13

- generalize the work of compiler as much as possible
  - machine code varies from architecture to architecture

- using frame pointer, specify the place for storing variavles
  -
$$fp := sp;;$$
$$sp := sp + framesize$$
- how to realize nested functions
  - One is that to add the variable as a parameter of the top function, which is called lifting
  - Another is that to use static link
- Partial evaluation (Futamura)
  - First one is $I_p = [P]_S(x)$
    - I_p is equal to a code that P (written in language S) is compiled to language L
  - Second one is $[PE_I]_L(P) = I_P$
    - PE_I is equal to the compiler that is from language S to language L
  - Third one is $PE_{PE}(I) = PE_I$
    - $PE_{PE}$is compiler compiler that receives interpreter and outputs compiler

10 Code generator 06/20

Today's lecture is about a step that interprets intermediate code into the machine code. This interpretation is composed of two steps, one is temporary code generation and the other is an allocation of registers. In forst steps, we treat computer as if it had an infinite registers.

- temporary code generation
  - temporary code generarion is like a putting a tile
  - Every instruction has some tiles and we decide which tile should be used using dynamic proggramming
- Register allocation
  - allocation is conducted based on the graph coloring Algorithms
  - liveness analysis, which checks which variables should be stored in registers at each instructions, is conducted based on the fixed point theorem
  - Using liveness check, we can construct a tree.
  - We color it and decide where to store each variables

11 optimization 07/04

- fixed point
- code optimization
  - temporary code level oprimization
  - optimization with the hardware