Compilers : Phase I : Lexical Analyzer

17. Basma Mahmoud Ali Taha
37. Abdel Rahman Hany
46. Amr Mahmoud Ahmed Bekhiet
67. Mennatollah Salah El Din El Kammah

a. A description of the used data structures:

The project is mainly divided into some classes as follows:

- **NFA class:**

```
class Nfa {
public:
  static unsigned int node_count;
  typedef struct Node {
    int state_no;
    bool accept_state;
    Node () {
      state_no = Nfa::node_count++;
      accept_state = false;
    }
    // holds next node with the transition input char to it.
    vector <pair<struct Node*, char> > Next_trans;
  } Node;
  Node *Start;
  Node *End;
```

- `struct Node`**: For each state in the Non-deterministic finite Automaton (NFA), thers is a Node representing it. This node carries the** `state_no` **and** `accept_state`**.**

- `state_no`**: Whenever a new Node (state) is created in the NFA, it is given a number representing it. That number is used later in the NFA to DFA conversion and its minimization.**

- `accept_state`**: A Boolean indicating whether that state is an acceptance state or not. Initially it has a false value, and is a Node is recognized to be an acceptance state while parsing, it will have a true value.**

- `vector <pair<struct Node*, char> > Next_trans:` **A vector of pairs holding for each node, its next nodes with the transition inputs leading to them.**

- `Node *Start`**: The start node of each new created NFA.**

- `Node *End:` **The end node for each new created NFA.**

- **Parser class:**

```
class Parser {
```

- `std::map<std::string, Nfa> cache;`
- `std::map<std::string, Nfa> components;`
- `std::map<std::string, int> op_prior;`
- `std::set<std::string> symbol_table;`
- `std::map<char, int> input_index;`
- `std::map<std::string, std::string> name_to_output;`
- `std::map<std::string, std::string> acceptance_to_expr;`
- `std::map<std::string, int> acceptance_priority;`
- `std::map<int, std::string> index_to_state;`

- `std::map<std::string, Nfa> cache:` **it contains the transient NFA.**

- `std::map<std::string, Nfa> components:` **it contains the NFA of the expressions.**

- `std::map<std::string, int> op_prior:` **The order of the operator in regex.**

- `std::set<std::string> symbol_table:` **The symbol table of the input.**

- `std::map<char, int> input_index:` **The index of the input in the transition table.**

- `std::map<std::string, std::string> name_to_output:` **Map the hashed name of the node to simpler name.**

- `std::map<std::string, std::string> acceptance_to_expr:` **Map the hashed name of the node to their acceptance states.**

- `std::map<std::string, int> acceptance_priority:` **It contains the priority of the expressions.**

- `std::map<int, std::string> index_to_state:` **It give the state of the given index**

b. Explanation of all algorithms and techniques used :

Mainly two algorithms are used. They are Thompson's algorithm and subset construction.

- **Thompson's algorithm is used as follows:**
  - `static Nfa* do_concat (Nfa *, Nfa *):`
    It concatenates two NFA's as the end of the first reaches the start of the second by an epslon transition.

  - `static Nfa* do_or (Nfa *, Nfa *):`
    It creats a new NFA by oring between the two NFA parameters. Two new nodes are created, one at the start and one at the end. That start goes with two epslon transitions to each start of the NFA's sent. Their ends go to that end node also with one epslon transition for each NFA.

  - `static Nfa* do_zero_more (Nfa *):`
    It creates a new NFA representing the zero or more transition .

  - `static Nfa* do_one_more (Nfa *):`
    It creates a new NFA representing the one or more transition .

  - `static Nfa* do_period (vector<char>):`
    It creates an NFA representing a range or input for the (a-z | A-Z). It creates the NFA to accept any input in that range.

  - `static Nfa* multiple_or (map<std::string, Nfa> &):`
    It deals with the multiple states oring each one of them and accepting any of them. It sets the acceptance state of each of them to true.

  - `void concat_chars (string):`
    It creates an NFA for the keywords (as: if, else, etc…) or in other words it takes the string parameter and creates an NFA to accept that string.

  - `Nfa* clone ():`
    It clones a present NFA to be used later to combine expressions with each other, for example id = letter (letter | digit)*. That function creates another NFA for the letter to be used another more time in that expression.

  - `Nfa* bfs ():`
    It is used to discover the resultant NFA graph and for each input for it, saves it and maps it to its next transitions depending on that input.

- **Subset Construction is used as follows:**
  It is used to convert from the NFA to DFA.

  - `std::vector<Node* > get_epslon_closure(Nfa::Node*):`
    For each node, this function returns a vector containing its epslon closure states.

  - `void build_dfa(Nfa::Node* src, std::vector<std::vector <std::string> > &transition_table, std::map<std::string, int> &name_to_index)`
    It represents the main flow for building the DFA from an NFA represented by the NFA's start (sent as a parameter), it calls multiple functions.

  - `Void add_fay_node (std::vector<std::vector <std::string> > &transition_table,std::map<std::string, int> &name_to_index)`
    This function adds the fay node to different data structures used.

  - `Void get_big_node_epslon (std::map<char, std::set<Nfa::Node*> > &inp,`
    `std::vector<std::vector <std::string> > &transition_table,`
    `std::map<std::string, int> &name_to_index, std::string cur_str,`
    `std::map<std::string, std::vector<Nfa::Node*> > &str_to_nodes,`
    `std::queue<std::string> &q)`
    This function iterates iterate over the new formed node to get their epslon closure.

  - `Void add_node_to_queue (std::queue<std::string> &q,`
    `std::map<std::string, std::vector<Nfa::Node*> > &str_to_nodes,`
    `std::string child_name, std::vector<Nfa::Node*> &output_nodes)`
    This function adds the new formed node to the queue if it is not visited.

  - `Void inspect_big_node_children (std::vector<Nfa::Node*> &cur_nodes,`
    `std::map<char, std::set<Nfa::Node*> > &inp)`
    This function iterate through the formed node and calls function inspect_children to form the new node.

  - `Void inspect_children (Nfa::Node* cur_node,`
    `std::map<char, std::set<Nfa::Node*> > &inp)`
    This function iterates over transition of the current node and insert the transitions with the non-epslon value.

- Void update_input_map (std::map<char, std::set<Nfa::Node*> > &inp,
  char transition, Nfa::Node* child)
  This function updates the children of the given input.

- Void inspect_children (Nfa::Node* cur_node,
  std::map<char, std::set<Nfa::Node*> > &inp)
  This function iterates over transition of the current node and insert the transitions with the non-epslon value.

- Void map_name_to_output (std::string child_node_name)
  This function maps the hashed name of the node to a simpler name S + digit.

- Void clean_transition_table (std::vector<std::vector <std::string> > &transition_table, std::map<std::string, int> &name_to_index)
  This function updates the transition table after minimization.

- Void add_node_to_transition_table (std::vector<std::vector <std::string> > &transition_table, std::map<std::string, int> &name_to_index, std::string cur_node_name, std::string child_node_name, char inp)
  This function adds the given node to different data structures used.

- Bool has_higher_acceptance (std::string cur_acc, std::string new_acc)
  This function returns the regex with higher priority.

- std::string concatenate(std::vector<Nfa::Node*> &nodes)
  This function hashes the name of the node.

- Void minimize_dfa (std::vector<std::vector <std::string> > &transition_table ,std::map<std::string, int> &name_to_index)
  This function minimizes the given DFA using the transition table built while forming the DFA.

- Void group_states (std::map<std::string, std::vector<std::string> > &grouping_states)
  This function groups the state according to their acceptance state.

c. The resultant transition table for the minimal DFA:

The resultant transition table could be found attached with the report with the name trans_table.txt

d. The resultant stream of tokens for the example test program:

int
id
,
id
,
id
,
id
;
while
(
id
relop
num
)
{
id
assign
id
addop
num
;
}

e. Any assumptions made and their justification:

The regex file shouldn’t be modified.