Scrivo questo documento per condividere alcune scelte progettuali riguardo alla soluzione proposta.

Per memorizzare i dati dell'automa non deterministico ho sfruttato la classe definita nella fase precedente.
Vi ho apportato alcune migliorie: in particolare, ho aggiunto il metodo specializzato

void Automaton<int, char, multimap<pair<int, char>, int>>::addTransition(const int& stateInReading, const char& inputAlpha, const int& toState)

Il metodo aggiunto implementa una politica di controllo durante l'inserimento di una nuova transizione. Le transizioni degli automi non deterministici, nella mia soluzione, vengono memorizzate all'interno di un oggetto tipo *multimap*; per evitare conflitti e duplicati di transizioni, si verifica il numero di transizioni esistenti con quello stato iniziale e quel simbolo dell'alfabeto.

Ho implementato una nuova classe, per migliorare la leggibilità del codice: class Regex .
Questa viene usata all'interno del programma esclusivamente per memorizzare l'alfabeto e la stringa di espressione.

Ho poi utilizzato le strutture dati e le funzioni fornite per memorizzare ed elaborare l'*Abstract Syntax Tree*.

Per implementare l'algoritmo di Thompson ho definito la funzione seguente:
Automaton<int, char, multimap<pair<int, char>, int>> REtoND(AST* abstractSyntaxTree, const Regex& regex)

REtoND restituisce l'oggetto NDFA completo derivato dalla espressione regolare in input.

Questa funzione lavora su uno stack stack<AST*> visitingAST = visitingASTforAutoma(abstractSyntaxTree);
La funzione visitingASTforAutoma restituisce lo stack con i nodi in ordine secondo un algoritmo di visita *DFS in post-order.*

Durante la visualizzazione dello stack, si procede alla costruzione di ogni componente dell'automa, memorizzato all'interno di stack<Automaton<int, char, multimap<pair<int, char>, int>>> automatons;

Dall'espressione regolare e dalla costruzione algoritmica, l'ultimo elemento di *automatons* rappresenta l'automa finale, costituito di tutti i componenti descritti.

In particolare, i componenti che costituiscono l'automa vengono realizzate in quattro speciali funzioni, tutte invocate all'interno della principale REtoND.

1. `Automaton<int, char, multimap<pair<int, char>, int>> AutomataPool_Symbol (const char symbol, int& stateCounter)`
2. `Automaton<int, char, multimap<pair<int, char>, int>> AutomataPool_KleeneStar (const Automaton<int, char, multimap<pair<int, char>, int>>& left, int& stateCounter)`
3. `Automaton<int, char, multimap<pair<int, char>, int>> AutomataPool_Union (const Automaton<int, char, multimap<pair<int, char>, int>>& left, const Automaton<int, char, multimap<pair<int, char>, int>>& right, int& stateCounter)`
4. `Automaton<int, char, multimap<pair<int, char>, int>> AutomataPool_Concatenation (const Automaton<int, char, multimap<pair<int, char>, int>>& left, const Automaton<int, char, multimap<pair<int, char>, int>>& right)`

Tutte le funzioni, escluse quella che implementa la concatenazione, lavorano con un contatore *stateCounter* che rappresenta l'indice di un eventuale stato successivo da inserire all'interno dell'automa.

Elia Pitzalis matr. 152541
284660@studenti.unimore.it