Tema 3 LFA

Macarie Razvan Cristian 332CB

Generare de parser si lexer cu Antlr4

Pentru a genera parserul si lexerul pentru expresii regulate am folosit Antlr4.

Fisierul Regex.g

```
//parser
regex : regex KLEENSTAR #regexStar |
        '('regex')' #regexParanth |
        regex regex #regexConcat |
        regex OR regex #regexOr |
        REGEX_ATOM #regexAtom;

//lexer
REGEX_ATOM : [a-z]; //un caracter
OR : '|';
KLEENSTAR : '*';
```

Regulile sunt denumite cu #nume_regula pentru ca antlr sa stie cum sa le numeasca cand genereaza codul din visitor. Am ales sa merg pe ruta asta pentru ca antlr nu se descurca cu left recursive rules (regula a -> regula b -> regula a).

Visitor implementation

Antlr ne ofera un mod de a parcurge parse tree-ul printr-un visitor design pattern, astfel fiecare regula are o functie definita.

Parcurgerea tree-ului este bottom up, se pornese din radacina cu parsarea insa se realizeaza operatii la intoarcerea din recursivitate. In decursul parcurgerii se creaza un NFA pas cu pas dupa fiecare nod din arbore.

Teorema lui Thompson

Pentru a crea nfa-ul folosim teorema de constructie a lui Thompson. Teorema are 3 cazuri de baza din care depistam doar unul, cel in care avem un singur caracter. Constructiile pentru Or, KleenStar si Concatenare sunt cele prezentate in laborator (sau pe wikipedia). Ne bazam astfel pe constructii succesive si recursive de la atom pana la operatia din radacina.

Fiecare regula creeaza un nfa si sa il puna intr-o stiva de nfa-uri (in principiu cazul de baza creeaza un nfa cu 2 stari una initiala si una finala).

Cand se intalneste un concat sau or:

- o sa fie neaparat intre 2 nfa-uri (care se gasesc pe stiva)
- indiferent de cat de simplu sau complicat e un nfa o sa aiba, tot timpul, doar o stare initiala si doar o stare finala
- cu nfa1 si df2 notam dfa-urile (dam pop de 2 ori din stiva pentru a le obtine)[1]
- starea finala din nfa2, sau o stare nou adaugata, devine finala pentru reuniunea lor
- concat: starea finala din nfa1 nu o sa mai fie finala si pointeaza catre starea initiala din nfa2 pe eps
- or: se adauga 2 stari una initiala si una finala (pentru reuniune), cea initiala are tranzitie pe eps catre ambele nfa-uri si ambele nfa-uri au de pe starile lor finale catre noua stare finala
- in final ne ramane un nfa cu o singura stare finala si una initiala

Cand se intalneste star:

- se da pop de pe stiva pentru ca nfa-ul pe care trebuie sa aplicam * se alfa in varful stivei[2]
- se adauga 2 stari, una finala si una initiala
- starea initiala pointeaza catre fosta stare initiala si catre starea finala
- starea finala pointeaza catre noua stare initiala

[1] este garantat ca ele se gasesc din felul in care functioneaza parserul
[2] este garantat ca ele se gasesc din felul in care functioneaza parserul

Transformare in DFA si printare

NFA-ul generat este de tip NFA din tema 2. Am modificat doar constructiorul clasei pentru a putea construi NFA-uri generice. Am redenumit modulul si l-am importat folosind functiile `NFA2DFA()` si `printOutput()` pentru a converti un NFA la DFA si pentru a printa DFA-ul rezultat intr-un fisier.

NFA-ul rezultat este printat de o alta functie definita in modulul main.