Computer Science Club

Colin Shaw

Dec. 10, 2015

0. INTRODUCTION
    a. Lots of clubs for specific languages
    b. Need to focus on solving problems
    c. Need more options to have healthy community
    d. Practice delivering complex discourse
    e. Video
        i. Incentive for better presentation
        ii. Accessible to more people
        iii. Demonstrates competence of our local developers

1. MOTIVATION
    a. Why this problem
        i. Fascinated me
        ii. Didn't study it in depth before
    b. General introduction to how ingrained regular expressions are in languages
    c. Illustrate examples based on string metaphor
        i. Constructors (/Colin/)
        ii. Operators (=~)

2. BACKGROUND
    a. FSM
        i. State diagram
            1. Vending machine diagram (machine metaphor)
            2. States
                a. Start
                b. Acceptor
            3. Transition function
        ii. Idea of a language / alphabet we are using (coins, letters)
        iii. Concepts of Finite and State

b. Finite Automata
    i. John Conway (game of life, etc.)
    ii. Simple example /colin/
        1. Break it up under concatenation
        2. Options with disjunctive union
        3. Add a cycle
        4. Add an epsilon
        5. Kleene star
            a. Regular expression coined by Stephen Kleene
            b. Regular expressions as in regular sets

c. General Properties
    i. DFA / NFA dichotomy
        1. Determined state = Deterministic Finite Automata (DFA)
        2. Indeterminate state = Non-deterministic Finite Automata (NFA)
        3. Complexity dictates determinism
            a. Number of acceptor states
                i. Single = DFA
                ii. Multiple = NFA
            b. Disjunctive unions
            c. Epsilon
    ii. DFA and NFA are equivalent
    iii. Interests we need tonight
        1. Disjunctive union
        2. Concatenation
        3. Kleene star
    iv. Other properties that hold (more on this later)
        1. Conjunctive union
        2. Complement

3. REGULAR EXPRESSIONS
    a. An aside about things we are not talking about
        i. Context free grammars
        ii. Turing machines (Alan Turing)
        iii. Recursive structures
        iv. Pushdown automata
            1. Stacks
            2. Memory

b. Definition: R is a regular expression if R is
    i. $\varnothing$ (the null pattern)
    ii. ε (the null character)
    iii. a (for some a in an alphabet)
    iv. Concatenation $R_1\ R_2$ (where R_1 and R_2 are regular expressions)
    v. Disjunctive union $R_1 \mid R_1$ (where R_1 and R_2 are regular expressions)
    vi. Kleene star R* (where R is a regular expression)
    vii. Notes
        1. R+ is simply RR*
        2. Generalized Regular Expressions
            a. Conjunction union
            b. Complement

c. Brzozozwksi's derivatives (Janusz Brzozozwksi)
    i. Preliminaries
        1. Prescriptive generator
            a. Recursive
            b. Generates NFA
        2. Derivative idea is notional
    ii. Define a function δ such that
        1. $\delta(\varnothing) = \varnothing$ (zero)
        2. $\delta(\varepsilon) = \varepsilon$ (unit)
        3. $\delta(c) = \varnothing$ (variable)
        4. $\delta(R_1\ R_2) = \delta(R_1)\ \delta(R_2)$ (multiplication)
        5. $\delta(R_1 \mid R_2) = \delta(R_1) \mid \delta(R_2)$ (addition)
        6. $\delta(R^*) = \varepsilon$
    iii. Example
        1. D_f (foo|far)* = (oo|ar)(foo|far)*
    iv. Define derivative with respect to a character c recursively
        1. $D_c(\varnothing) = \varnothing$
        2. $D_c(\varepsilon) = \varnothing$
        3. $D_c(c) = \varepsilon$
        4. $D_c(c') = \varnothing$ if $c \neq c'$
        5. $D_c(R_1\ R_2) = \delta(R_1)\ D_c(R_2) \mid D_c(R_1)\ R_2$ (think multiplication)
        6. $D_c(R_1 \mid R_2) = D_c(R_1) \mid D_c(R_2)$ (think addition)
        7. $D_c(R^*) = D_c(R)\ R^*$
    v. Observations
        1. End states
            a. $\varnothing$ (no match)
            b. ε (has match)
        2. All other relations are recursive

4. PROGRAMMING
    a. OCaml benefits
        i. Language
            1. Originally for mathematical theorem proving
            2. Polymorphic disjunctive union (sum) typing
            3. Constructor matching
            4. Hindley-Milner type checking
            5. Modules and Functors
            6. Object Orientation
        ii. Execution
            1. Bytecode compiler with forward and reverse step debugging
            2. Fast native compiler
            3. Various REPLs
        iii. In the wild
            1. Financial institutions (Jane Street Capital)
            2. .Net CLR (F#)
            3. Compilers
                a. FFTW
                b. Coq (More advanced formal proof language)

    b. Walk through regex code