Lecture 9: Regular Expressions in Haskell/ Context-Free Grammars

CSCI 101
Spring, 2019

Kim Bruce

General simulation of DFSM

- Provide transition function as set of triples
- Build machine from start state, triples, and set of final states
  - myFSM = FSM start triples final
- To apply write
  - gAccept myFSM input
- Returns True iff it accepts it.

Big Idea in Implementation

- Suppose you want to see if configuration (s,w) $\vdash^*$ (t,ε), where t is a final state.
- If w = aw' and (s,a) = u, then just need to check if (u,w') $\vdash^*$ (t,ε)
- I.e., making a move equivalent to lopping off first input and running on new DFSM w/ start state u.

Modeling Regular Expressions

- Create functions that allow construction of machines that model regular expressions: (a|b)*
  - E.g. `star(union(once 'a') (once 'b'))` will return a value `amstar` s.t. `match amstar inp` will return true iff `inp` is in set generated by (a|b)*

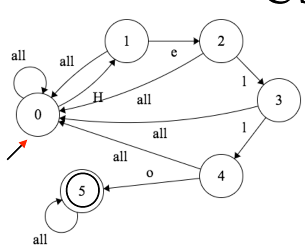
Modeling Regular Expressions

- Meaning of regular expression will be a DFSM that is equivalent to regular expression.
- In class:
  - Regular expression ⇒ NDFSM
  - NDFSM ⇒ DFSM
- We will do both at once.
  - Meaning of regular expression will be DFSM whose states are sets of states from NDFSM.

Modeling Regular Expressions

- The meaning of a regular expression is a tuple with number of states, the starting state (a set), the, transition function, and the set of accepting states
  - Keeping track of number of states tell us set of states:
    - If 6 states, then they are {0,1,2,3,4,5}

Using RegExp




```
hello = Regex 6 (Set.singleton 0) f (Set.singleton 5)
  where
    f 'H' 0 = Set.fromList [0,1]
    f 'e' 1 = Set.fromList [0,2]
    f 'l' 2 = Set.fromList [0,3]
    f 'l' 3 = Set.fromList [0,4]
    f 'o' 4 = Set.fromList [0,5]
    f _ 5 = Set.singleton 5 -- stay in 5 forever as saw "Hello"
    f _ _ = Set.singleton 0 -- if get something unexpected go to 0
```

Building Regular Expressions

- Look at
  - empty (representing empty set)
  - epsilon (representing empty string {ε})
  - dot (matches anything)
  - build machines for singletons and union

Using DFSM model

- Build up regular expression equivalent in prefix form:
  - (a|b)* represented by
    - aorbstar = star (union (once ‘a’) (once ‘b’))
    - where once is a singleton, so once ‘a’ represents {‘a’}
  - Once build use match to apply to string
    - match aorbstar “ababa”

Context-Free Grammars

CFGs are Useful!

- Use to describe programming and natural languages
  - ForStatement:
    for (ForInit_{opt} ; Expression_{opt} ; ForIncr_{opt}) Statement
  - English:
    - Sentence ::= NP VP
    - NP ::= Art Nominal | Nominal | ProperNoun | ...
    - Nominal ::= N | Adj N
    - N := cat | dog | girl | boy | ...

Definitions

- A context-free grammar is a quadruple, G = (V, Σ, R, S) in which
  - V is a finite set of variables, containing terminals and nonterminals.
  - $\Sigma \subseteq V$ is the set of terminals
  - R is a finite set of productions of the form $U \rightarrow \alpha$, where U is a single nonterminal and α is a (possibly empty) string of terminals and nonterminals.
    I.e., OK to write $U \rightarrow \varepsilon$
  - S is an element of V called the start symbol.

One-Step

- Define $w \Rightarrow_G w'$ so that $\forall x, y \in V^*$,
  - $w \Rightarrow_G w'$ iff
    $w = \alpha A \beta$, $w' = \alpha\gamma\beta$ and there is a rule $A \rightarrow \gamma$ in R
  - $\Rightarrow_G^*$ is the reflexive, transitive closure
    - means derivable in 0 or more steps.
- $L(G) = \{ w \in \Sigma^* \mid S \Rightarrow_G^* w \}$
- L is a context-free language if there is a cfg G s.t. $L = L(G)$

Examples

- *Note: Often only state rules, rather than all 4 pieces*
- $S \rightarrow w \mid w'$ abbreviates two rules: $S \rightarrow w$, $S \rightarrow w'$
- Language of balanced parens:
  - $S \rightarrow SS \mid (S) \mid \varepsilon$
  - Show derivation of ()(())
- $L = \{0^n 1^n \mid n \geq 0\}$ is a context-free language
- $L = \{w w^R \mid w \in \Sigma^*\}$ is cfl

CFLs Richer Than Regular

- Regular languages $\subsetneq$ context-free languages
  - Because regular grammars are context-free & above examples not regular
  - Power comes because of recursive embedding:
    - $A \Rightarrow^* wAw'$ for $w,w' \neq \varepsilon$

Closure

- CFL's closed under
  - Concatenation
  - Kleene *
  - Reversal
  - Union
  - Substitution
- What about complement, intersection, difference, ...?