Lecture 5: Pumping Lemma

CSCI 81
Spring, 2015

Kim Bruce

Last Time:

- Regular languages are those
  - accepted by DFSM
  - accepted by NDFSM
  - described by regular expressions
  - generated by regular grammars
- How do we show languages not regular?
  - Show violate some property of regular languages

Pumping Lemma:

If L is regular, there is a number p (the pumping length) where, if $w \in L$ of length at least p, then there are x, y, & z with $w = xyz$, such that:

1. for each $i \geq 0$, $x y^i z \in L$;

2. $|y| > 0$; and

3. $|xy| \leq p$.

Use to show languages not regular!

Using Pumping

- Show $L = \{0^n 1^n \mid n \geq 0\}$ is *not* regular
  - Proof by contradiction. Assume regular.
  - Therefore exists p from P.L.
  - Let $w = 0^p 1^p \in L$
  - By P.L. can write $w = xyz$ s.t. $|xy| = k \leq p$ s.t. $xy^iz \in L$ for all i
    - But $|xy| \leq p \Rightarrow x, y$ consist of all 0's.
    - So $x = 0^i$, $y = 0^j$, $z = 0^{p-i-j}1^p$ where $j > 0$.
    - Pick $n = 2$, then $xy^2z = 0^{p+j}1^p \notin L$. Contradiction so not regular!

Pumping Lemma Game

- To show L not regular
  - Opponent picks p
  - I pick w s.t. $|w| \geq p$
  - They pick decomposition $w = xyz$ s.t. $|xy| \leq p$, $y \neq \varepsilon$
  - I show there is some i s.t. $x y^i z \notin L$
- If I succeed then I have shown L not regular!

Regular or Not?

- $L = \{a^i b^j : 0 \leq i < j < 2000\}$.
- $L = \{a^i b^j : i, j \geq 0 \text{ and } i < j \}$.
- $L = \{a^i b^j : i, j \geq 0 \text{ and } i \geq j \}$.
- $L = \{w \in \{a,b\}^* : |w| \text{ is a power of } 2\}$

Decision Problems w/FSM

- Let $L = \mathit{L}(M)$ be a regular language, where M is DFSM, & $w \in \Sigma^*$. It is decidable whether
  - $w \in L$
  - $\mathit{L}(M) = \varnothing$
    - Algo 1: Mark all reachable states. See if any are accepting.
    - Algo 2: Create unique minimal and see if any are accepting
  - $\mathit{L}(M) = \Sigma^*$

Decision Problems w/FSM

- Let $L = \mathit{L}(M)$ be a regular language, where M is DFSM, & $w \in \Sigma^*$. It is decidable whether
  - $\mathit{L}(M)$ is infinite
    - Use pumping lemma!
    - Claim: if L(M) infinite then must have w s.t. $|K| \leq |w| < 2|K| - 1$.
  - $\mathit{L}(M_1) \subseteq \mathit{L}(M_2)$
    - Difference is empty
  - $\mathit{L}(M_1) = \mathit{L}(M_2)$
    - Use above or compare canonical minimal DFSM's

Programming in Haskell!

According to Larry Wall (designer of PERL): ... a language by geniuses for geniuses

He's wrong — at least about the latter part
though you might agree when we talk about monads

Read Haskell Tutorials

- All on links page from course web page
- I like "Learn you a Haskell for greater good"
- O'Reilly text: "Real World Haskell" free on-line
- Print Haskell cheat sheet
- Use "The Haskell platform", available at
  - http://www.haskell.org/

Using GHC

- to enter interactive mode type: ghci
  - :load myfile.hs -- :l also works
  - after changes type :reload *or* :r
  - Control-d to exit
  - :set +t -- prints more type info when interactive
  - "it" is result of expression
    - Evaluate "it + 1" gives one more than previous answer.

Built-in data types

- Unit has only ()
- Bool: True, False with not, &&, ||
- Int: 5, -5, with +, -, *, ^, =, /=, <, >, >=, ...
  - div, mod defined as prefix operators (`div` *infix*)
  - Int fixed size (usually 64 bits)
  - Integer gives unbounded size
- Float, Double: 3.17, 2.4e17 w/ +, -, *, /, =, <, >, >=, <=, sin, cos, log, exp, sqrt, sin, atan.

More Basic Types

list of Char

- Char: 'n'
- String = [Char], not really primitive
  - "hello"++" there", length *Prefix op w/out ``!*
  - No substring, but `isInfixOf` for all lists
  - Also 'isPrefixOf`, `isSuffixOf' *import Data.List*
- Type classes (later) provide relations between classes.

Interactive Programming with ghci

- Type expressions and run-time will evaluate
- Define abbreviations with "let"
  - let double n = n + n
  - let seven = 7
- "let" not necessary at top level in programs loaded from files

Lists

- Lists
  - [2,3,4,9,12]: [Integer]
  - [] -- empty list
  - Must be homogenous
  - Functions: length, ++, :, map, rev
    - also head, tail, *but normally don't use!*

Polymorphic Types

- [1,2,3]:: [Integer]
- [“abc”, “def”]:: [[Char]], ...
- []:: [a]
- map:: (a → b) → ([a] → [b])
- *Use* :t exp *to get type of* exp

Pattern Matching

- Decompose lists:
  - [1,2,3] = 1:(2:(3:[]))
- Define functions by cases using pattern matching:

```
prod [] = 1
prod (fst:rest) = fst * (prod rest)
```

Pattern Matching

- Desugared through case expressions:
  - head' :: [a] -> a
    head' [] = error "No head for empty lists!"
    head' (x:_) = x
- equivalent to
  - head' xs = case xs of
    [] -> error "No head for empty lists!"
    (x:_) -> x

Type constructors

- Tuples
  - (17,”abc”, True) : (Integer , [Char] , Bool)
  - fst, snd defined only on pairs
- Records exist as well

More Pattern Matching

- (x,y) = (5 `div` 2, 5 `mod` 2)
- hd:tl = [1,2,3]
- hd:_ = [4,5,6]
  - "_" is wildcard.