CS 241 Lec 8

Graham Cooper

June 1st, 2015

Linker Algorithm

Input: MERL files m1 and m2
Output: Single MERL file with m2 linked after m1

$\alpha \leftarrow$ m1.codeLen - 12
relocate m2.code by α
add α to every address in m2.symtbl
if m1.exports.labels $\cap$ m2.exports.labels $\neq \emptyset$, ERROR
for each <$addr_1$, label>$\in$ m1.imports
– if $\exists$<$addr_2$, label>$\in$ m2.exports
—- m1.code[$addr_1$] $\leftarrow$ $addr_2$
—- remove <$addr_1$, label >from m1.imports
—- add $addr_1$ to m1.relocates
for each <$addr_2$, label >$\in$ m2.imports
– if $\exists$ <$addr_1$, label >$\in$ m1.imports
—- m2.code[$addr_2$] $\leftarrow$ $addr_1$
—- remove <$addr_2$, label >from m2.imports
—- add $addr_2$ to m2.relocates
imports = m1.imports $\cup$ m2.imports
exports = m1.exports $\cup$ m2.exports
relocates = m1.relocates $\cup$ m2.relocates
output MERL cookie
output total codeLen + total(import,exports,relocates) + 12
output total codLen + 12
output m1.code
output m2.code
output imports, exports, relocates

Formal Languages

High level language $\rightarrow$ Compiler $\rightarrow$ Assembly Language

Assembly:

- simple structure
- easy to recognize
- straightforward, unambiguous translation to machine code

High-level

- more complex structure
- harder to recognize
- no single translation to machine code

How can we handle the complexity of a compiler?

Start with a formal theory of string recognition
- general principals that work for any programming language

Definitions

Alphabet:

- Finite set of symbols (eg. {a,b,c})
- typically denoted Σ, as in Σ = {a,b,c}

string(or word):

- finite sequence of symbols (from Σ)
- eg. a, aba, cbca, ...
- length of a word $|\omega|$ = number of characters in the word,
- eg. $|aba| = 3$
- empty string - an empty sequence of symbols use ϵ to denote the empty string. ϵ is not a symbol, $|\epsilon| = 0$

Language:

- a set of strings (words)
- eg $\{a^{2n}b|n \geq 0\}$ all words with an even number of a's followed by b
- NOTE: ϵ vs $\{\}$, epsilon is the empty word, but the empty braces is the empty language (contains no words). $\{\epsilon\}$ is a singleton language, that contains only ϵ

Our task

Question: How can we recognize automatically whether a given string belongs to a given language?

Answer: Depends on how complex the language is.

- $\{a^2nb|n \geq 0\}$ - easy
- {valid MIPS assembly programs} - almost as easy
- {valid Java Programs} - harder
- some languages - impossible

Characterize languages according to how difficult the recognition process is. - classes of languages based on "hardness"

- finite
- regular
- context-free
- context-sensitive
- recursive languages
- etc.

Top is easy, bottom is impossible

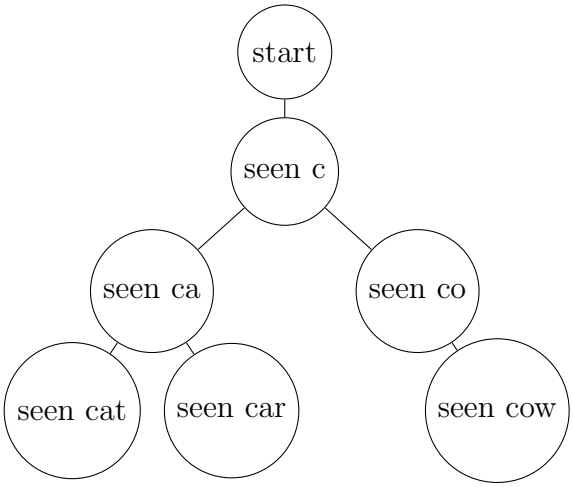
Want to work at as easy a level as possible, move to a harder level if neccessary.

Finite Languages:

- have finitely many words
- can recognize a word by comparing with every lement of the set (finite!)
- can we do this efficiently?

Exercise: L = {cat, car, cow}
Write code to answer $\omega \in L$ such that ω is scanned exactly once, without storing previously-seen chars.

Scan input left to right
if first character is c, move on, else error
if next character is a:
– if next char is t :
—- if input is empty, accept, else error
– else if char is r:
—- if input is empty, accept, else error
– else error else if character is o :
– if next character is w : —- if input is empty, accept, else error
– else error
else error



Bubbles are states, configurations f the program based on the input seen. The leaf nodes mean you accept if you stop there.

Since programming languages don't usually admit only finitely many programs, finite languages not that useful.