CSCI203

Algorithms and Data Structures

Binary Expression Trees & Finite State Machines

Lecturer: Dr. Fenghui Ren

Room 3.203

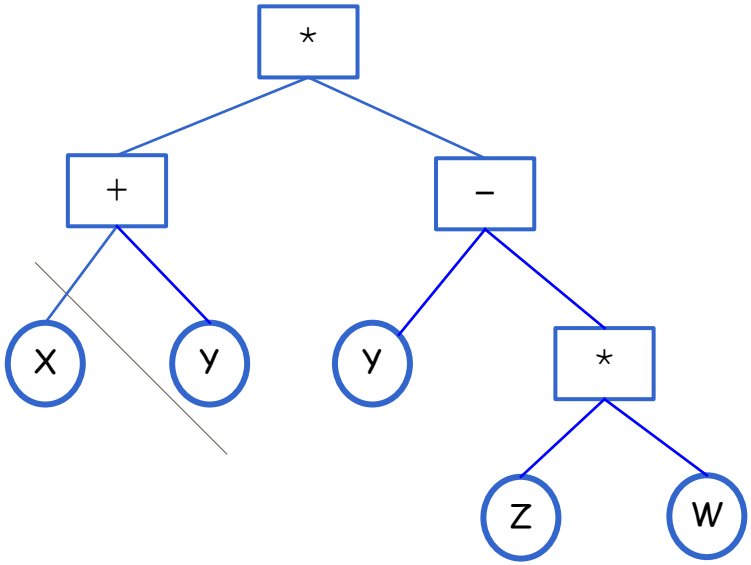
Email: fren@uow.edu.au

Fun With Binary Trees

- Expression trees.
- Building an expression tree:
  - with help from a stack.
- All trees are binary trees.

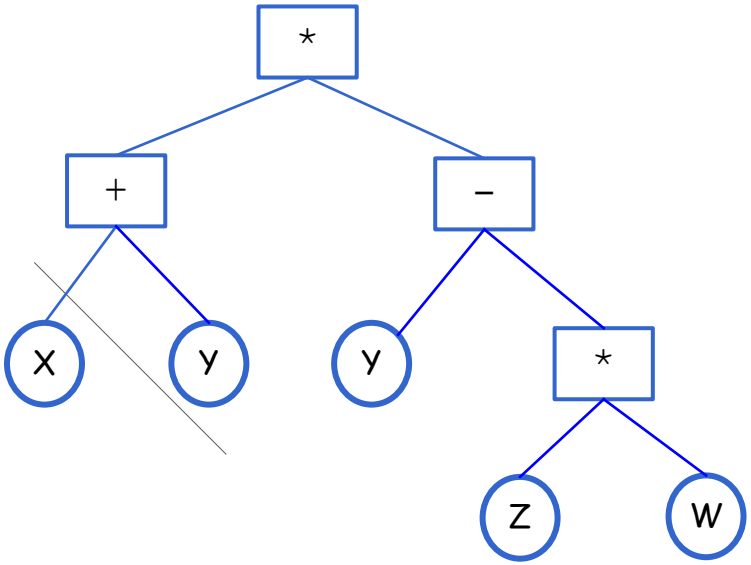
Binary Expression Trees

- Consider an algebraic expression:
  - $(x + y) * (y - (z * w))$
- We can represent this expression as a binary tree where:
  - The internal nodes are the operations;
  - The leaves are the variables.
- Thus:



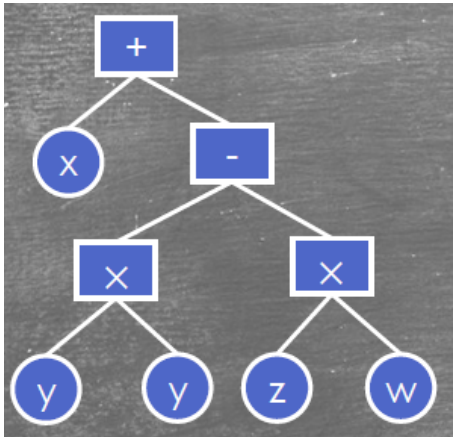
Traversal of an Expression Tree

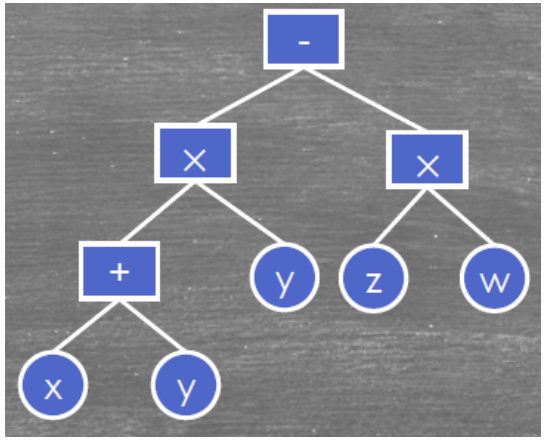
- We can traverse an expression tree:
- Pre-order:
  - $* + xy - y * zw$
- In-order
  - $x + y * y - z * w$
- Post-order
  - $xy + yzw * - *$
- Note that, although the in-order traversal almost produces the expression we started with, we have no parentheses.
  - The in-order traversal can represent many trees.

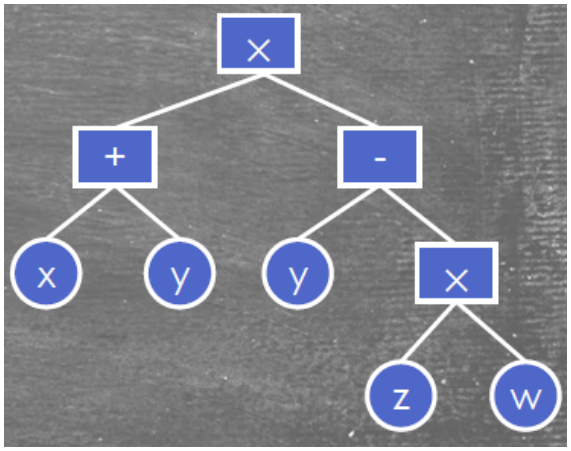


$x + y \times y - z \times w$ (In-Order Traversal)

- This expression can represent…



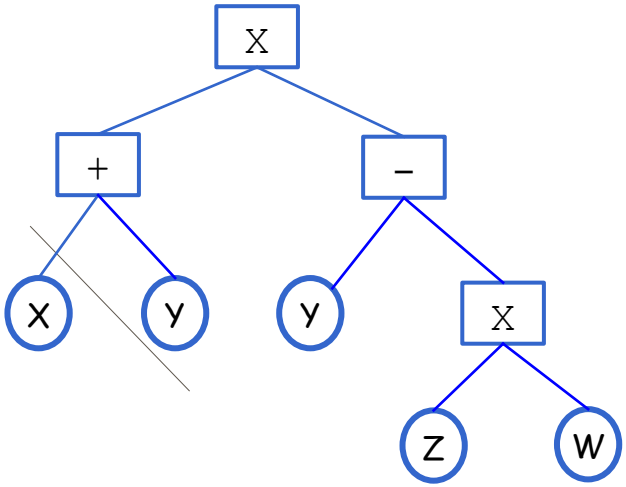




- …as well as many other trees
- We can modify the in-order traversal to insert parentheses:

$xy + yzw\times-\times$ (Post-Order Traversal)

- The post-order traversal is equivalent to a unique expression tree;
  - The one we started with.
- We can use this fact to evaluate an expression tree from the post-order traversal.
- We will do this with the aid of a stack:

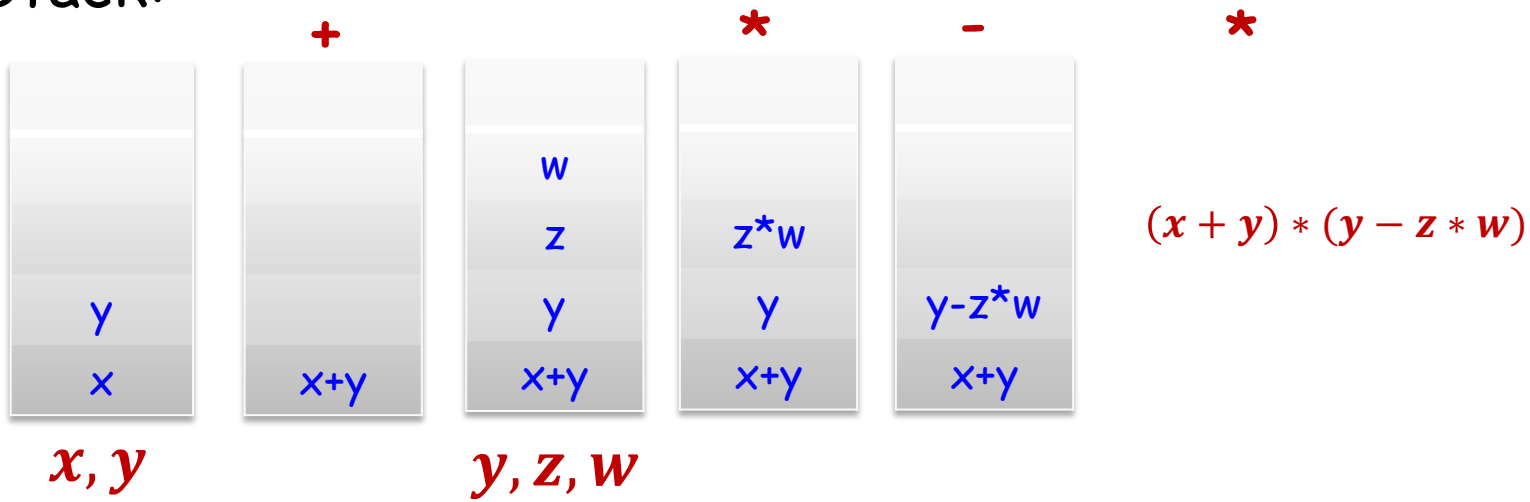


Evaluate an expression

- The rules we follow are:
  1. Start with an empty stack
  2. Take each symbol of the expression, left to right:
     a. If the symbol is a letter/value:
        - push it onto the stack
     b. If the symbol is an operator:
        - Pop the top two elements of the stack as R and L in order
        - evaluate (L operator R), and push the result to the stack
  3. Until finish processing the expression

An Example

- Consider the expression $xy + yzw *-*$
- Moving from left to right:
  - Symbol: $xy + yzw *-*$
- Stack:



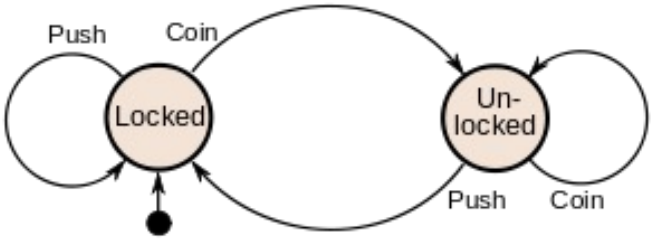
- Note: we can do something similar by processing the pre-order traversal from end to start.

Finite State Machines (FSMs)

- *A reactive system whose response to a particular input is not the same on every occasion, depending on its current "state".*

- FSM Examples
  - Vending Machines
  - Traffic Lights
  - Elevators
  - Alarm Clock
  - Microwave
  - Cash Registers

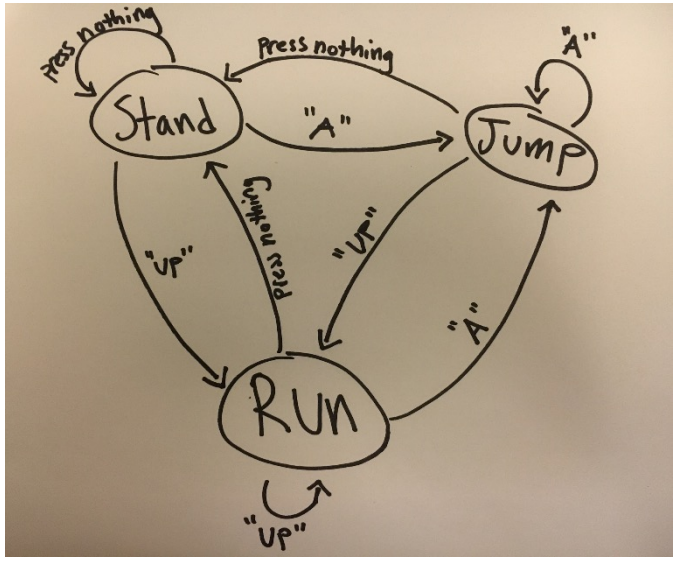
Examples



- A system where particular inputs cause particular changes in state can be represented using finite state machines.
- This example describes the various states of a turnstile. Inserting a coin into a turnstile will unlock it, and after the turnstile has been pushed, it locks again. Inserting a coin into an unlocked turnstile, or pushing against a locked turnstile will not change its state.

https://brilliant.org/wiki/finite-state-machines/

Examples



In the standing state, the player can press nothing and remain in the standing state, then, to transition to the running state, the user must press the "Up" button. In the running state, the user can continue to make their character run by pressing the "Up" button, and then to transition to the jump state, the user must press "A."

https://brilliant.org/wiki/finite-state-machines/

Finite State Machines

- This is a useful programming tool for processing streams of data.

- We define the process in terms of:
  1. A set of states that the program can be in.
  2. A set of possible inputs
  3. What action must be taken when we see a particular input and we are in a particular state. This includes the new state to change to.

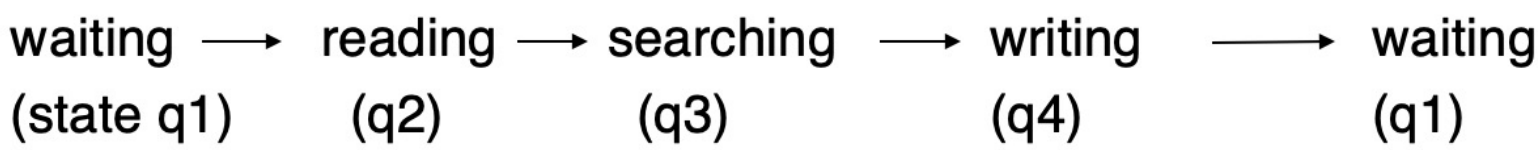
- We can represent this as a table.

Finite State Machines (FSM)

At its lowest level, observable behavior of a system can be described with Finite State Machines (FSMs), which are special cases of automata (abstract machine).

The state of a system can be determined by checking the value of one or more state variables.

Suppose that a system might have state variable q1, q2, q3, q4 with values from the set {waiting, reading, writing, searching}. The possible state sequence for the system might be:



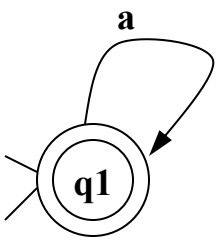
Definition of a finite-state machine

A finite-state machine M is defined by a five-tuple (Q, F, q0, S, δ) where

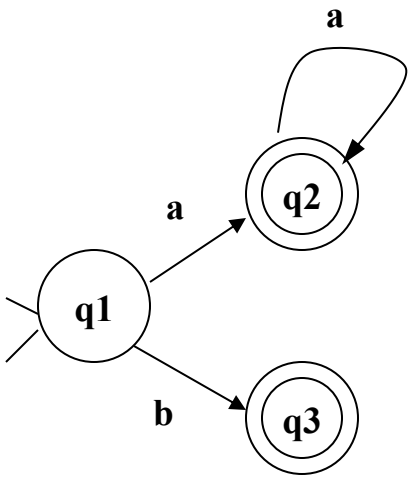
- Q is a finite set of states {q0, q1, q2, …, qn}
- F is a subset of final (accepting) state of Q
- q0 is a single start state in Q
- S is an input/output alphabet
- δ: Q x S $\rightarrow$Q maps the current input and current state into the next state

Notation FSM diagrams

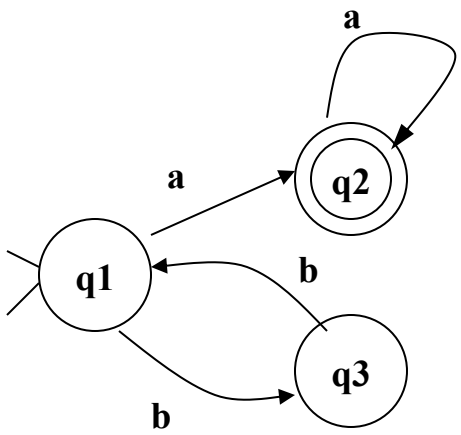
Sample finite-state machines



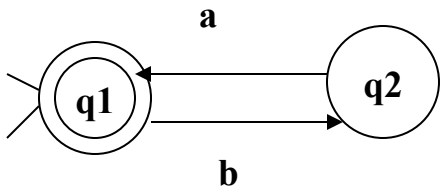
(a) accepts
a, aa, aaa, ...



(b) accepts b or
a, aa, aaa, aaaa, ...



(c) accepts
bba, bbaa, …, or
a, aa, aaa, aaaa, ...



(d) accepts
ba, baba, ...

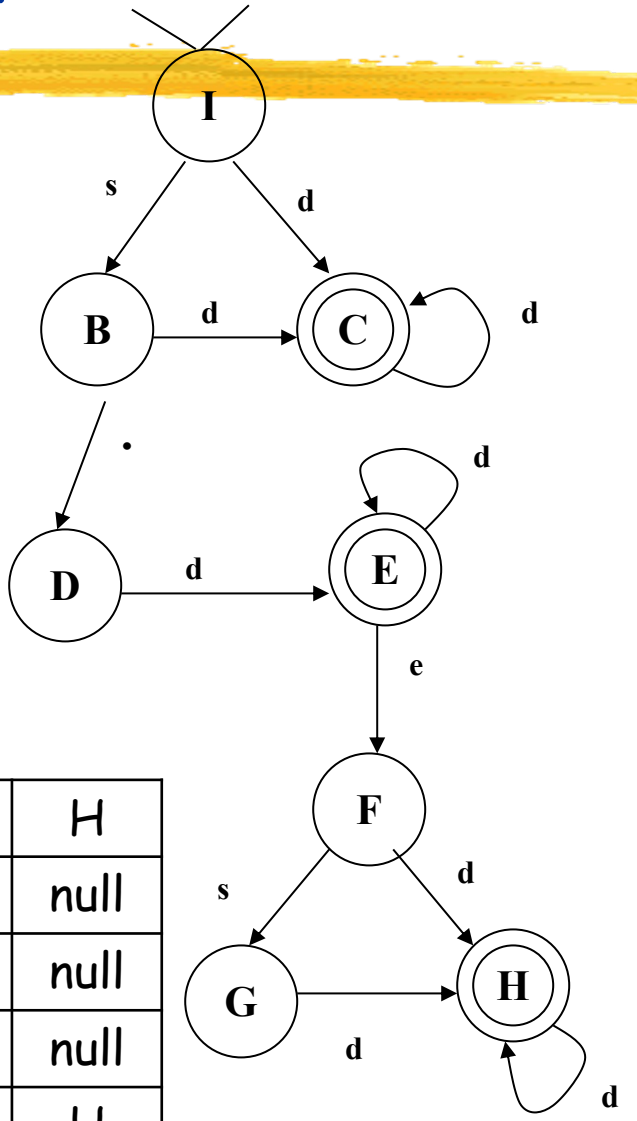
State table

The behaviors of FSMs can be represented in a table form. For example, a complete representation of the behavior of machine (b) is given in the following table.

State \ Input	a	b
q1	q2	q3
q2	q2	Φ
q3	Φ	Φ

Example: Use FSM to give the specification of numeric constants as accepted by the Pascal programming language

Inputs: S = { s, ., e, d}
States: Q={I, B, C, D, E, F, G, H}
Final states: F={C, E, H}
Initial state: q0=I



State Table

	I	B	C	D	E	F	G	H
s	B	null	null	null	null	G	null	null
.	null	D	null	null	null	null	null	null
e	null	null	null	null	F	null	null	null
d	C	C	C	E	E	H	H	H

Related References

- Introduction to the Design and Analysis of Algorithms, A. Levitin, 3rd Ed., Pearson 2011.
  - Chapters 6.3