Minimization of Finite Automata

Introduction

Canonical DFA (Minimum State DFA)

- For each state q_A in Q_A and q_B in Q_B, there must exist $\sigma \in \Sigma^*$ such that,
  - Without such σ, we could remove q_A or q_b to find a smaller automaton
  - q_A will correspond with q_B
- $|Q_A| = |Q_B|$ and DFA A and B are isomorphic

Two minimizing algorithms

1. John Hopcroft (Hot cross buns)
2. Janusz (John) Brzozowski

Outline

- Hopcroft's Algorithm
- Brzozowski's Algorithm
- Comparison

Hopcroft's Algorithm

Fixed Point: the fixed point of a function is a value where the function's input and output are equal

- For f: $X \rightarrow X$, the fixpoint is some value $x \in X$ where $f(x) = x$

Hopcroft's Algorithm Outline:

- Input: DFA M
- Output: Minimal DFA M', such that $L(M') = L(M)$
- Algorithm: Repeatedly refine partitions until reaching a fixpoint:
  1. Partition states initially into accept F and non-accept Q / F
  2. Repeatedly refine partitions:

     2.1. If partition p contains states that transition to different successor partitions on symbol s.

     2.2. Split p into new sub-partitions where all states in each sub-partition transition to the same successor partition on s
  3. Repeat the refinement until reaching the fixpoint (no further refinements possible)

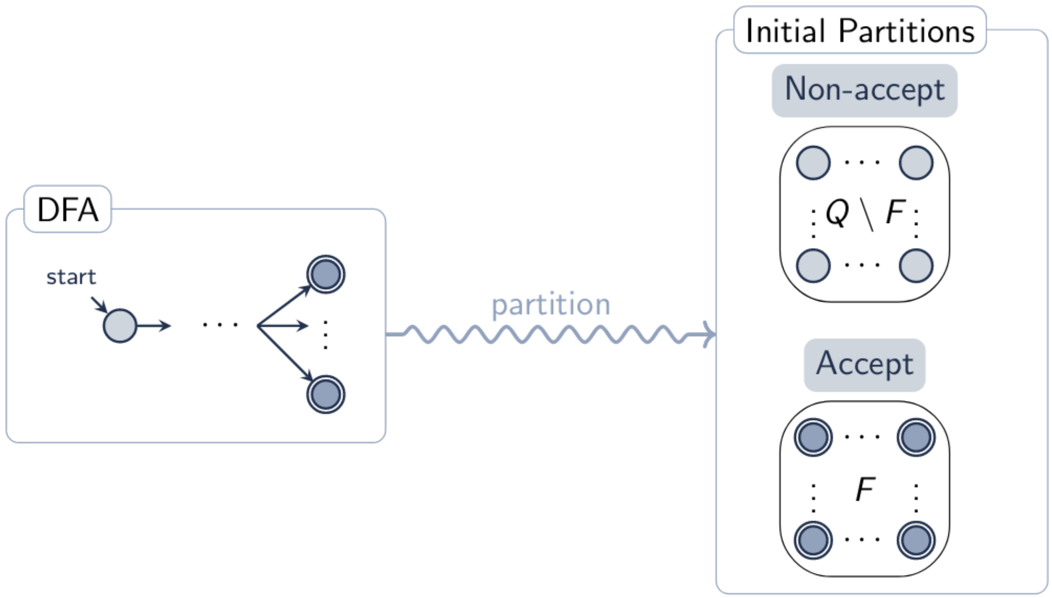

Figure 1: Hot Cross Buns: Initialization step

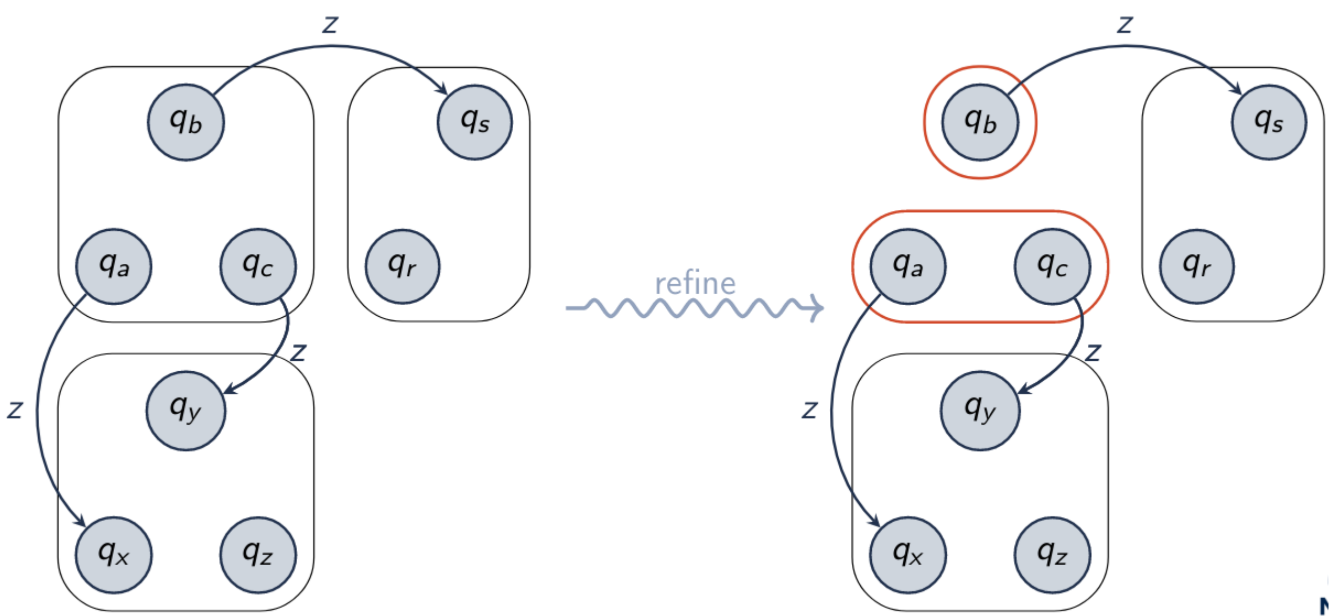

Figure 2: Hot Cross Buns: Refinement Step

b was split from a and c because they pointed to different partitions

Hopcroft's Algorithm (Detailed)

Algorithm 1: Hopcroft's Algorithm

```
Input: Q, Σ, E, s, F ;                  // FA states, tokens, edges, start, accept
Output: Q', Σ, E', s', F' ;             // Minimum DFA states, tokens, edges, start, accept
Q' ← {F, Q \ F}; // Initial Partitioning
W ← F; // Work list
while W do
    q' ← pop(W) ;
    forall z ∈ Σ do
        x ← { p ∈ Q | ∃ (p -z-> r) ∈ E,  r ∈ q' }; // z-predecessor states of partition q'
        if x then
            Q* = ∅;
            forall  y ∈ Q' do
                i = y ∩ x; // Subset of partition y transitioning on z to q'
                j = y \ x; // Subset of partition y transitioning on z to q̄'
                if  i ∧ j then
                    Q* ← Q* ∪ i ∪ j; // Replace partition y with i and j
                    if  y ∈ W then  W ← (W \ y) ∪ i ∪ j ;
                    else if  |i| < |j| then  W ← W ∪ i ;
                    else  W ← W ∪ j ;
                else  Q* ← Q* ∪ y ; // Don't split y
            Q' ← Q* ;
```

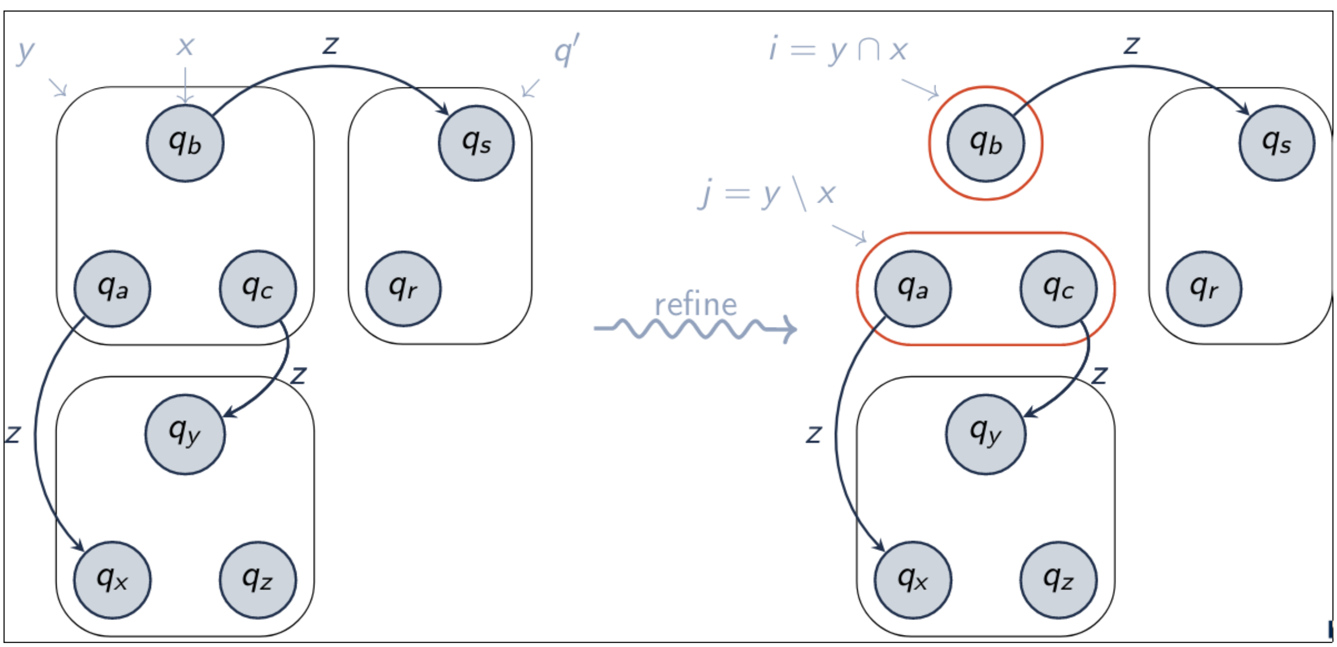

Figure 3: Illustration of components from the detailed algorithm

Brzozowski's Algorithm

Algorithm 2: Brzozowski's Algorithm

Input: A; `// FA`

Output: A'; `// Minimum state DFA`

$A' \leftarrow$ `nfa-to-dfa(reverse(nfa-to-dfa(reverse(`A`))))`;

(caveat: may need to eliminate a redundant start state)

Hopcroft's VS Brzozowski's

	Hopcroft	Brzozowski
Input	DFA	DFA or NFA
Worst-case runtime	O(k n ln n)	exponential (P(Q))
Average-case runtime	O(k n ln n)	"pretty good"

n = |Q| (number of states) k = |Σ| (size of alphabet)