Set Partitioning

Dr. John McDonough

Spoken Language Systems
Saarland University

June 4, 2009
In this lecture, we consider how to determine the *intersection* of two languages.

We also consider an algorithm for *set partitioning* that can also be used to minimize a weighted-finite state automaton.

**Coverage:** Aho, Hopcroft, Ullman (1974), Section 4.13.
Recall that we defined an equivalence relation $xR_Ly$ for a language $L$ when either $xz$ and $yz$ belong to $L$ or both do not belong.

The *index* is the number of equivalence classes in a language $L$.

An equivalence relation $R_L$ whereby $xzR_Lyz$ follows from $xR_Ly$ is known as right invariant.
The following statements are equivalent:

1. The set $L \subseteq \Sigma^*$ is accepted by a finite-state automaton.
2. $L$ is the union of equivalence classes of a right invariant equivalence relation with finite index.
3. The equivalence relation can be defined as follows: $xR_Ly$ is holds if and only if $xz$ is in $L$ when $yz$ is in $L$. Then $L$ has a finite index.
Consider a set $S$ and an initial partition $\pi$ of $S$ into disjoint blocks $\{B_1, B_2, \ldots, B_p\}$.

There is also given a function $f$ on $S$.

The task is to find the coarsest partition $\pi' = \{E_1, E_2, \ldots, E_q\}$ such that
1. $\pi'$ is consistent with $\pi$ (that is, each $E_i$ is a subset of some $B_j$, and,
2. $a$ and $b$ in $E_i$ implies $f(a)$ and $f(b)$ are in some $E_j$.

We then call $\pi'$ the coarsest partition of $S$ compatible with $\pi$ and $f$. 
Naive Solution

- Let $B_i$ be a block.
- Examine $f(a)$ for each $a$ in $B_i$.
- $B_i$ is partitioned so that $a$ and $b$ are in the same block if and only if $f(a)$ and $f(b)$ are in the same block.
- This process is iterated until no further refinements are possible.
Example

- Let $S = \{1, 2, \ldots, n\}$, and let $B_1 = \{1, 2, \ldots, n-1\}$, $B_2 = \{n\}$ be the original partition.
- Define the function $f$ on $S$ as

$$f(i) \triangleq \begin{cases} 
  i + 1, & \text{for } 1 \leq i < n \\
  n, & \text{for } i = n.
\end{cases}$$

- On the first iteration, $B_1$ is partitioned into $\{1, 2, \ldots, n-2\}$ and $\{n-1\}$.
- This iteration requires $n - 1$ steps because each element in $B_1$ must be examined.
- On the next iteration, we partition $\{1, 2, \ldots, n-2\}$ into $\{1, 2, \ldots, n-3\}$ and $\{n-2\}$. 
Running Time of the Naive Solution

- A total of \(n - 2\) such iterations are required, whereby the \(i\)th iteration requires \(n - i\) steps, for a total of

\[
\sum_{i=1}^{n-2} \frac{n(n-1)}{2} - 1
\]

steps.

- The problem with the naive solution is that refining each block requires \(O(n)\) steps, even if only a single element is removed.

- We would like to develop an algorithm whereby refining a block into two subblocks requires time proportional to the smaller subblock.

- This will result in a \(O(n \log n)\) algorithm.
For each $B \subseteq S$, let $f^{-1}(B) = \{ b | f(b) \in B \}$.

The naive algorithm partitions a block $B_i$ by the values of $f(a)$ for $a \in B_i$.

Instead, let us partition with respect to $B_i$ those blocks $B_j$ which contain at least one element in $f^{-1}(B_i)$ and one element not in $f^{-1}(B_i)$.

That is, each $B_j$ is partitioned into the sets $\{ b | b \in B_j \text{ and } f(b) \in B_i \}$, and $\{ b | b \in B_j \text{ and } f(b) \notin B_i \}$. 
Once we have partitioned with respect to $B_i$, we need not partition again with respect to $B_i$ unless $B_i$ is itself split.

If initially $f(b) \in B_i$ for each element $b \in B_j$, and $B_i$ is split into $B'_i$ and $B''_i$, then we can partition $B_j$ with respect to either $B'_i$ or $B''_i$.

This follows because $\{b| b \in B_i \text{ and } f(b) \in B'_i\}$ is the same as $B_i - \{b| b \in B_i \text{ and } f(b) \in B''_i\}$. 
Conventional Automaton

Let define a conventional automaton without weights.

**Definition (finite-state machine)**

A FSM is a 5-tuple $A = (\Sigma, Q, E, i, F)$ consisting of
- an *alphabet* $\Sigma$,
- a finite set of states $Q$,
- a finite set of *transitions* $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$,
- a *initial state* $i \in Q$,
- and a set of *end states* $F \subseteq Q$.

A transition $e = (p[e], l[e], n[e]) \in E$ consists of
- a previous state $p[e] \in Q$,
- a next state $n[e] \in Q$,
- a label $l[e] \in \Sigma$,

A final state $q \in F$ may have an associated label $a \in \Sigma$. 
Problem Statement

Consider a FSM with the set of states $Q$.

We wish to partition $Q$ into subsets $M = \{Q_i\}$ such that for some $i$.

We seek the coarsest partition $\{Q_i\}$ of $Q$, which is by definition the partition with fewest elements, that satisfies (1).

Let $\nu$ be a partition of $Q$ and let $f$ be a function mapping $Q \times \Sigma$ to $Q$. In the present case, $f$ is defined implicitly through the transitions $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$.

For each $Q_i \in \nu$ define the sets

$$\text{symbol}(Q_i) = \{ a \in \Sigma : \exists e = (p, a, n) \in E, n \in Q_i, p \in Q \}$$
Pseudocode

Pseudocode for the partitioning algorithm is shown below:

```
00  def partition():
01    Q₀ ← Q − F
02    Q₁ ← F
03    push Q₀ on S
04    push Q₁ on S
05    n ← 1
06    while |S| > 0:
07      pop P from S
08      for a in symbol(P):
09        for Q_j such that Q_j ∩ f⁻¹(P, a) ≠ ∅ and Q_j ⊆ f⁻¹(P, a):
10          n += 1
11          Q_n ← Q_j ∩ f⁻¹(P, a)
12          Q_j ← Q_j − Q_n
13          if Q_j ∈ S:
14            push Q_n on S
15          else:
16            if |Q_n| < |Q_j|:
17              push Q_n on S
18            else:
19              push Q_j on S
```
We will say the set \( T \subseteq Q \) is \textit{safe} for \( \nu \) if for every \( B \in \nu \), either
\[
B \subseteq f^{-1}(T, a) \quad \text{or} \quad B \cap f^{-1}(T, a) = \emptyset \quad \forall \ a \in \Sigma.
\]

The key of the algorithm is the partitioning of \( Q_j \) in Lines 12–13, which ensures that there are no transitions of the form
\[
e_1 = (p_1, a, n_1) \quad \text{and} \quad e_2 = (p_2, a, n_2),
\]
where either \( p_1, p_2 \in Q_j \) or \( p_1, p_2 \in Q_n \), for which (1) does not hold.

Hence, Lines 12–13 ensure that \( P \) is safe for the resulting partition, inasmuch as if \( Q_j \cap f^{-1}(P, a) \neq \emptyset \) for some \( Q_j \), then either \( Q_j \subseteq f^{-1}(P, a) \), or else \( Q_j \) is split into two blocks in Lines 12–13, the first of which is a subset of \( f^{-1}(P, a) \), and the second of which is disjoint from that subset.

For reasons of efficiency, the smaller of \( Q_j \) and \( Q_n \) is placed on \( S \) in Lines 17–20, unless \( Q_j \) is already on \( S \), in which case \( Q_n \) is placed on \( S \) in Lines 14–15 regardless of whether or not
\[
|Q_n| < |Q_j|.
\]
Aho et al (1974) proved the following lemma. **Lemma (set partitioning):** After the algorithm in Listing ?? terminates, every block $Q_i$ in the resulting partition $\nu'$ is safe for the partition $\nu'$. 