So far, all our partition refinement algorithms require transition functions rather than relations.

Naturally, the questions arises how to deal with nondeterministic transitions.

Here is the key difference between the two situations.

Suppose we have some equivalence relation $\rho$ on $Q$ and sets $B \subseteq A \subseteq Q$.
To simplify notation, let $\overline{X} = Q - X$ for any $X \subseteq Q$. As before, we use the refinement operator $\Psi_X(\rho) = \rho \cap (X, X)$.

Suppose $A$ is $\rho$-stable. In the function case we have $\Psi_B \circ \Psi_A^{-B} = \Psi_B$ and one refinement step suffices.

But not in the general case: given a block $D$ we can have up to three slices:

$D^+ = D \cap \tau B$
$D^- = D \cap \overline{\tau B}$
$D^{++} = D \cap \tau (A - B)$
$D^{+-} = D \cap \tau (A - B)$

But note that $D^-$ is not split by $\tau (A - B)$ (since $A$ is stable).

Thus the challenge is to compute these pieces quickly.

One can slightly simplify matters by first splitting with respect to $\tau Q$: this removes states with out-degree 0 from discussion.

Letting $m = |\tau|$ and $n = |Q|$ we then have $m \geq n - 1$. Actually, since we are not interested in the function case we may safely assume $m \geq n$.

The algorithm maintains a partition $P$ and a split list $S$. The entries of $S$ are lists of blocks of $P$ of length at least 2. Each block in $P$ may occur at most once in $S$.

$P$ is initialized by the given coloring, and $S = ((P_1, \ldots, P_r))$. 

initialize $P$ and $S$

while $S$ not empty do

extract $(X_1, \ldots, X_r)$ from $S$

assume wlog $|X_1| \leq |X_2|$

let $B = X_1$ and $A = \bigcup X$

if $r \geq 3$

then add $(X_2, \ldots, X_r)$ to $S$

foreach block $D$ do

if $D$ splits

then compute

$D^+ = D \cap \tau B$
$D^- = D \cap \overline{\tau B}$
$D^{++} = D \cap \tau (A - B)$
$D^{+-} = D \cap \tau (A - B)$

update $P$, $S$ accordingly
Comment 7

Note that if we add all blocks in $P$ missing in $S$ to $S$ we get a partition $S'$ such that $P \subseteq S'$. The algorithm stops when $P = S'$.

The "update" part is meant to
- split blocks already in $S$ by updating the corresponding list, and
- place newly split blocks into $S$ (as a list of length 2 or 3).

Correctness 8

Let $\rho$ be the coarsest stable refinement. Then $\rho \subseteq P$ is a loop invariant of the algorithm.

To see that upon completion $\rho = P$ note that after each round $P$ is stable with respect to the sets in $S'$.

Running Time 9

Because of Hopcroft's trick, every state is used to split at most $\log n$ times.

As before, the critical part is perform the actual splitting work without recomputation. To this end keep counters for all $X = \bigcup X_i$ in $S$:

$$\#_X \tau = |p \tau \cap X|$$

Thus $\#_X \tau$ is the number of transitions with source $p$ and target in $X$.

Initially $\#_Q \tau$ is just the out-degree of $p$.

Splitting 10

- Compute $\tau B$ and $\#_B \tau$ for $p \in \tau B$.
- Split blocks wrt $\tau B$.
- Compute $\Delta = \tau B - \tau (A - B)$. To this end, add $p$ to $\Delta$ whenever $\#_B \tau = \#_A \tau$.
- Split $D^+$ via $\Delta$.

One can show that one splitting round is

$$O(|B| + \sum_{p \in B} |\tau p|),$$

leading to a total running time of $O(m \log n)$.

Results 11

Theorem (Paige, Tarjan 1987)

Refinement with respect to a single relation can be done in $O(m \log n)$ steps.

One can adapt the Paige-Tarjan algorithm to the case of multiple relations; alas, the logarithm factor cannot be maintained.

Theorem (Fernandez 1989)

Refinement with respect to a family of relation can be done in $O(mn)$ steps.

Partition Refinement

Minimizing NFAs

Hardness
Note that for any regular language there trivially must be an NFA that minimized the number of states.

Alas, as simple examples show, one cannot hope to transfer the uniqueness results from DFA minimization to the nondeterministic realm.

Theorem
Let $U$ be the universal NFA for $R$ and $A$ any NFA such that $L(A) \subseteq L(U)$. Then there is an epimorphism $\pi: L_0 \xrightarrow{a_1} L_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} L_n$ can be lifted to a computation

$$\pi: L_0 \xrightarrow{a_1} K_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} K_n$$

where $L_s \subseteq K_s = (a_1 \ldots a_s)^{-1} L_0$.

This has the immediate consequence

Proposition
The universal NFA for $R$ accepts $R$. More precisely, state $K$ in $U$ has behavior $K$.

The universal NFA $U$ for a language $R$ is defined as follows:

States All non-empty intersections of left quotients.
Transitions $K \xrightarrow{a} L$ iff $L \subseteq a^{-1} K$.
Initial All states $K \subseteq R$.
Final All states containing $\varepsilon$.

Note that the universal NFA contains a copy of the minimal DFA for $R$: the part generated by initial state $K$ and using only the “maximal” transitions

$$K \xrightarrow{a} L \iff L = a^{-1} K$$

Note that for any regular language there trivially must be an NFA that minimized the number of states.

Alas, as simple examples show, one cannot hope to transfer the uniqueness results from DFA minimization to the nondeterministic realm.

Properties
Let $U$ be the universal NFA for $R$ and $A$ any NFA such that $L(A) \subseteq L(U)$. Then there is an epimorphism $f: A \rightarrow U$.

Proof. Define $f(p) = \bigcap \{ u^{-1} R \mid A : I \xrightarrow{a} p \}$

Suppose $p \xrightarrow{a} q$ is a transition in $A$. Then

$$f(q) = a^{-1} f(p) \cap \bigcap u^{-1} R$$

where $w$ ranges over all words $I \xrightarrow{w} q$ that are not captured by the first term.

Theorem
Let $U$ be the universal NFA for $R$ and $A$ any NFA such that $L(A) \subseteq L(U)$. Suppose there is an epimorphism $f: U \rightarrow A$. Then $f$ is an isomorphism.

Proof.
Note that $p = [p]_U = [f(p)]_A$, so $f$ must be injective.

$\Box$
Suppose \( C \) and \( D \) are two classes of finite state machine acceptors.

**Problem:** \( C \)-\( D \) Minimization

**Instance:** A machine \( A \) in class \( C \).

**Solution:** An equivalent machine \( A' \) in class \( D \) of minimal state complexity.

As usual, it is more convenient to consider the decision version of the problem:

**Problem:** \( C \)-\( D \) Minimization

**Instance:** A machine \( A \) in class \( C \), a bound \( B \).

**Question:** Is there an equivalent machine \( A' \) in class \( D \) of state complexity at most \( B \)?

### Hardness of Minimization

**Theorem**

DFA-DFA Minimization is polynomial time.

**Theorem**

NFA-DFA Minimization is PSPACE-complete.

**Theorem**

NFA-NFA Minimization is PSPACE-complete.

For hardness, suppose \( T \) is a nondeterministic Turing machine with polynomial space complexity \( p(n) \). Let \( x \) be some input of size \( n \). We may safely assume that the instantaneous descriptions (IDs) of \( T \) in all computations on \( x \) have size exactly \( p(n) \) and can be written as:

\[
 a_1 a_2 \ldots a_1 p b_1 b_2 \ldots b_r
\]

Here the \( a_i \) and \( b_i \) are symbols from some tape alphabet \( \Sigma \), and \( p \) is a state in \( Q \) (assumed to be disjoint from \( \Sigma \)).

For our purposes, it is slightly easier to work with two directed copies of the tape alphabet:

\[
 \Sigma' = \{ \#, a \mid a \in \Sigma \}
\]

An ID is coded as a word over the alphabet \( \Gamma = \Sigma' \cup Q \cup \{\#, \$\} \), where \( \# \) and \( \$ \) are new separator symbols, and has the form:

\[
 \# a_1 \ldots a_1 p \# b_1 b_2 \ldots b_r
\]

A whole computation looks like:

\[
 W = \# I_0 \# I_1 \ldots \# I_n \$
\]

For the proof we construct an NFA \( A \) of size polynomial in \( n \) that accepts all words over \( \Gamma \) iff \( T \) does not accept \( x \).
There are two basic reasons why $W$ may not code an accepting computation:

- $W$ is syntactically wrong, it fails to code a computation.
- $W$ does code a computation, but it is not accepting.

Testing syntactical correctness is fairly straightforward: we match against the pattern

$$\# \sum^* Q \sum^* \#$$

Thus, a correct string may contain a factor $ap$ or $pI$, but neither $a$p nor $pI$. Similarly, it may contain blocks $ab$ and $gh$, but not $gh$ or $gh$.

In fact, we can check in addition that all $#$ symbols are at distance $p(n)$ (and likewise for the endmarker $\$).

It is a labor of love to check that one can indeed build an NFA $A = \bigcup A_i$ of size polynomial in $n$ that accepts exactly all strings that do not represent an accepting computation on $x$.

But then $T$ accepts $x$ iff $L(A) \neq I^*$ and PSPACE-hardness follows.

It follows immediately that the NFA-DFA and NFA-DFA minimization problems are PSPACE-complete.

To check syntactically correct strings that do not represent accepting computations one has to test the disjunction of three properties:

- The first ID in $W$ is not the initial ID for input $x$.
- The last ID in $W$ is not an accepting ID.
- Two consecutive IDs in $W$ do not represent one step in a computation.

The first two are straightforward. For the last, we have to rule out e.g. factors of the form

$$\# \ldots apb \ldots \# aq'\ldots$$

whenever there is no transition $(p, b) \rightarrow (q, b', 0)$ in $T$. Note that this is a strictly local property; we only need to focus on three consecutive letters at distance $p(n) + 1$.

The problem is clearly in PSPACE.

As before, let $T$ be a Turing machine with polynomial space bound and consider IDs encoded as strings

$$\#a_1 a_2 \ldots a_i p b_1 b_2 \ldots b_i$$

A whole computation has the form

$$\#I_0 \#I_1 \ldots \#I_{m-1} \#$$

and we may safely assume that the length of each ID is exactly $p(n)$ and that $m$ is even.

The idea is to construct a family of DFAs that make sure that $I_{2i} \rightarrow I_{2i+1}$ is correct, and another family that checks $I_{2i+1} \rightarrow I_{2i+2}$. We will only deal with the first case.
Blow-Up Example 2

Here is a 6-state semiautomaton.

\[ X = b : \text{machine is deterministic and co-deterministic, so the power automaton has size 1.} \]

\[ X = a : \text{the power automaton has maximal size } 2^6. \]

Determinization and Reachability

We can dress up the determinization process as a decision problem:

Problem: Power Automaton Reachability

Instance: An NFA \( \mathcal{A} \), a state set \( P \subseteq Q \).

Question: Is \( P \) a state in \( \mathcal{P}(\mathcal{A}) \)?

A deterministic algorithm like DFS requires potentially exponential space (to keep track of marked nodes).

But the problem can easily be solved nondeterministically by an LBA: guess a witness string \( x \in \Sigma^* \) (one symbol at a time) and verify that \( I \xrightarrow{\epsilon} P \) in \( \mathcal{A} \).

Tip of an Iceberg

The example generalizes to a whole group of circulant machines on \( n \) states with diagram \( C(n; 1, 2) \).

Start with a labeling where the edges with stride 1 are labeled \( a \) and the edges with stride 2 are labeled \( b \).

Then change exactly one of these edge labels: the resulting nondeterministic machines have power automata of size \( 2^n \) and the power automata are already reduced.

Exercise

Full blow-up means that for any subset \( P \subseteq [n] \) there is some word \( x \) such that \( \delta([n], x) = P \). Determine such a word \( x \).

Exercise

Prove that full blow-up occurs for all these NFA.

The Digraph

The graph in question is determined by the NFA \( \mathcal{A} \): its nodes are subsets of \( Q \) and edges are determined by

\[ (P, P') \in E \iff \exists a \in \Sigma (P \cdot a = P'). \]

Note that this is a succinct representation of the graph: as a consequence, the counting problem now shoots up to \( \text{NSPACE} \) from \( \text{NL} \).

In practice, succinct representations of finite state machine by, say, Boolean Decision Diagrams are quite important.
Write $\pi(A)$ for the size of the accessible part of the power automaton of $A$. So we would like to solve the following function problem:

**Problem:** Power Automaton Size

**Instance:** An NFA $A$.

**Solution:** Compute $\pi(A)$.

As usual, it is convenient to consider a decision version:

**Problem:** Power Automaton Size Bound

**Instance:** An NFA $A$, a bound $\beta$.

**Question:** Is $\pi(A) < \beta$?

If there were a fast algorithm for this problem, we could use it to avoid trying to determinize NFAs where blow-up occurs. Alas, getting a bound on the size of the power automaton is just as hard as constructing the whole machine.

The complement of the Power Automaton Size problem can be solved nondeterministically by an LBA: the LBA will

- guess $\beta$-many subsets $P \subseteq Q$, say, in lexicographic order, and
- guess a string $x \in \Sigma^*$ and verify that $I \xrightarrow{x} P$ in $A$.

By Immerman-Szelepsényi, the Power Automaton Size problem itself is also in linear nondeterministic space.

In fact, one can change the query to $\leq \beta$ by $\geq \beta$, $\beta < \beta$ and so forth, without affecting complexity.

The version that is most convenient for the hardness argument is $\pi(A) \geq \beta$.

The proof uses the DFA Intersection emptiness problem: let $A_1, \ldots, A_r$ be DFAs over some alphabet $\Sigma$. From Kozen’s proof, we may assume that the DFAs are unitary.

Let $\Gamma = \Sigma \cup \{a, b, c\}$, $a, b, c$ three new symbols.

Let $n_i$ be the size of $A_i$. Assume $r \geq 4$ and that $n_i \leq n_{i+1}$. Let $p_i$ be the least prime larger than $7$, $r$, $n_i$, $p_{i-1}$, and such that $n_i + 2^{p_i} \leq 2^{n_i}$, for $i = 1, \ldots, r$.

We construct new machines $A_i'$ by attaching a pointed cycle labeled $b$ of length $p_i$ to $A_i$.

Each cycle node except the base-point has a self-loop labeled by $c$. The unique final state of $A_i'$ has a cone labeled $b$ to the whole cycle $C_i$.

Lastly, we attach a self-loop labeled $a$ to the initial state of $A_i$.

The semiautomaton $A = \bigcup A_i' \cup C$ is the union of all the modified DFAs plus an extra pointed cycle $C$:

- $C$ has length $m = 7$ and is $b$ labeled; each node has a self-loop labeled by $\Sigma$ and $c$.
- The loop at the base-point is in addition labeled $a$. There are no transitions from or to $C$ from any other part of $A$.

The state set of $A$ is $Q = \bigcup Q_i \cup \bigcup C \cup C$. Write $Q'$ for $\bigcup Q_i$ and $C'$ for $\bigcup C$.

Terminology: elements of $Q$ are points, subsets are states.

We have to count the number of states in $\mathcal{P}(A)$ that are reachable from $Q$.

Non-proper words are of one of the following mutually exclusive forms, together with bounds on the number of corresponding states:

- $\Sigma^* \quad m \prod 2^{n_i}$
- $\Sigma^* a \Sigma^* \quad m \prod n_i$
- $\Sigma^* a \Sigma^*(a + c) \Gamma^* \quad m \prod 2^{n_i}$
- $\Sigma^* (b + c) \Gamma^* \quad m \prod 2^{n_i}$

Hence the number of states reachable from $Q$ is bounded from above by

$$\gamma = 3\alpha(|r|) + m(1 + \sum x \alpha(x)),$$

where the summation is over suitably chosen factors $x \in \Sigma^*$ in proper inputs.
Now consider the bound $\beta = m \cdot \alpha([r])$.

If the Kozen automata have non-empty intersection, at least $\beta$ states are reachable, as required.

If the intersection is empty, then $\mathcal{A}(x)$ has cardinality at most $r - 1$ and we have

$$\frac{\gamma}{\beta} \leq \frac{3\alpha([r]) + m(1 + \sum x \alpha(x))}{m \cdot \alpha([r])}$$

$$= \frac{3}{m} + \sum x \frac{\alpha(x)}{\alpha([r])} + \frac{1}{\alpha([r])}$$

$$< \frac{1}{2} + \sum_i 2^{-p_i} + 2^{-\sum_i p_i} < 1.$$ 

Thus, $\gamma < \beta$, and we are done.