CDM

Minimization and NFAs

Klaus Sutner

Carnegie Mellon University

26-minimization 2017/12/15 23:19

1. Partition Refinement

- Minimizing NFAs

- Hardness

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 ρ 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

$$\Psi_X(\rho) = \rho \cap (X, \overline{X})$$

Suppose A is ρ-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:

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

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 τQ: 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 as 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
    - $D^+ = D \cap \tau B$
    - $D^- = D \cap \overline{\tau B}$
    - $D^{++} = D \cap \tau(A - B)$
    - $D^{+-} = D \cap \overline{\tau(A - B)}$
  - update P, S accordingly

Note that if we add all blocks in P missing in S to S we get a partition S' such that $P \sqsubseteq S'$. 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).

Let ρ be the coarsest stable refinement. Then $\rho \sqsubseteq P$ 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'.

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:

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

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

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

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

One can show that one splitting round is

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

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

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

2. 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.

$R = a^+b + (aa)^+c$

In the minimal DFA, the states correspond to the left quotients of R:

1	$a^+b + (aa)^+c$
2	$a^*b + a(aa)^*c$
3	$a^*b + (aa)^*c$
4	ε

The NFA has a state with behavior a^*b and another with behavior $(aa)^*c$.

This suggests a wild idea: how about trying to use regular subsets of quotients, rather than just the quotients themselves?

The universal NFA $\mathfrak{U}$ 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 ε.

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

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

More generally, any computation in $\mathfrak{U}$

$$\pi : L_0 \xrightarrow{a_1} L_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} L_n$$

can be lifted to a computation

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

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 $\mathfrak{U}$ has behavior K.

Theorem

Let $\mathfrak{U}$ be the universal NFA for R and $\mathcal{A}$ any NFA such that $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathfrak{U})$. Then there is an epimorphism $f : \mathcal{A} \to \mathfrak{U}$.

Proof. Define

$$f(p) = \bigcap \{\, u^{-1}R \mid \mathcal{A} : I \xrightarrow{u} p \,\}$$

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

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

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

□

Theorem

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

Proof.

Note that $p = [\![p]\!]_{\mathfrak{U}} = [\![f(p)]\!]_{\mathcal{A}}$, so f must be injective.

□

- Partition Refinement
- Minimizing NFAs

3. Hardness

Suppose $\mathcal{C}$ and $\mathcal{D}$ are two classes of finite state machine acceptors.

Problem:	$\mathcal{C}$-$\mathcal{D}$ **Minimization**
Instance:	A machine $\mathcal{A}$ in class $\mathcal{C}$.
Solution:	An equivalent machine $\mathcal{A}'$ in class $\mathcal{D}$ of minimal state complexity.

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

Problem:	$\mathcal{C}$-$\mathcal{D}$ **Minimization**
Instance:	A machine $\mathcal{A}$ in class $\mathcal{C}$, a bound B.
Question:	Is there an equivalent machine $\mathcal{A}'$ in class $\mathcal{D}$ of state complexity at most B?

Theorem

DFA-DFA Minimization is polynomial time.

Theorem

NFA-DFA Minimization is PSPACE*-complete.*

Theorem

NFA-NFA Minimization is PSPACE*-complete.*

Theorem (Stockmeyer, Meyer 1973)

NFA Universality is PSPACE*-complete.*

Proof.

Suppose $\mathcal{A}$ is an NFA. Since PSPACE is closed under complement we can instead check that $\mathcal{L}(\mathcal{A}) \neq \Sigma^\star$. Note that a witness $x \in \Sigma^\star - \mathcal{L}(\mathcal{A})$ may have exponential length. However, since $\mathrm{PSPACE} = \mathrm{NPSPACE}$, we can simply guess x letter by letter and verify that x is duly not accepted by $\mathcal{A}$. This is clearly in linear space.

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

$$a_1 a_2 \ldots a_l \, p \, b_1 b_2 \ldots b_r$$

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

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

$$\Sigma' = \{ \underset{\rightarrow}{a}, \underset{\leftarrow}{a} \mid a \in \Sigma \}$$

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

$$\#\underset{\rightarrow}{a}\underset{\rightarrow}{a} \ldots \underset{\rightarrow}{a}\, p\, \underset{\leftarrow}{b}\underset{\leftarrow}{b} \ldots \underset{\leftarrow}{b}$$

A whole computation looks like

$$W = \#I_0 \# I_1 \ldots \# I_m \$$$

For the proof we construct an NFA $\mathcal{A}$ of size polynomial in n that accepts all words over Γ 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

$$(\# \underset{\rightarrow}{\Sigma}^\star \, Q \, \underset{\leftarrow}{\Sigma}^\star)^\star \, \$$$

Thus, a correct string may contain a factor $\underset{\rightarrow}{a}\, p$ or $p \, \underset{\leftarrow}{a}$, but neither $\underset{\leftarrow}{a}\, p$ nor $p \, \underset{\rightarrow}{a}$. Similarly, it may contain blocks $\underset{\rightarrow\rightarrow}{ab}$ and $\underset{\leftarrow\leftarrow}{ab}$, but not $\underset{\rightarrow\leftarrow}{ab}$ or $\underset{\leftarrow\rightarrow}{ab}$.

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

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

$$\# \ldots \underbrace{a\, p\, b \ldots \# \ldots}_{p(n)+1} a\, q\, b' \ldots$$

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

It is a labor of love to check that one can indeed build an NFA $\mathcal{A} = \bigcup \mathcal{A}_i$ of size polynomial in n that accepts exactly all strings that fail to represent an accepting computation on x.

But then T accepts x iff $\mathcal{L}(\mathcal{A}) \neq \Gamma^\star$ and PSPACE-hardness follows.

□

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

Here is a version of the Emptiness Problem using multiple DFAs:

Problem:	**DFA Intersection**
Instance:	A list $\mathcal{A}_1, \ldots, \mathcal{A}_n$ of DFAs.
Question:	Is $\bigcap \mathcal{L}(\mathcal{A}_i)$ empty?

Note that n, the number of machines, is not fixed. Of course, we can check Emptiness on the product machine $\mathcal{A} = \prod \mathcal{A}_i$. The Emptiness algorithm is linear, but it is linear in the size of $\mathcal{A}$, which is itself exponential. And, there is no universal fix for this:

Theorem (Kozen 1977)

The DFA Intersection Problem is PSPACE*-complete.*

It might be tempting to re-purpose the argument for NFA Universality. The idea there was to construct a collection of NFAs $\mathcal{A}_i$ such that

$$\Gamma^\star = \bigcup \mathcal{L}(\mathcal{A}_i) \iff T \text{ does not accept.}$$

It follows that $\bigcap(\Gamma^\star - \mathcal{L}(\mathcal{A}_i)) = \emptyset$ if the Turing machine does accept.

Hence it would suffice to construct a polynomial size DFA for $\Gamma^\star - \mathcal{L}(\mathcal{A}_i)$.

Alas, there appears to be no way to convert all the NFAs from the previous argument to DFAs without encountering exponential blowup. And we do not know how to complement without determinization.

The problem is clearly in PSPACE.

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

$$\#a_1 a_2 \ldots a_l \, p \, b_1 b_2 \ldots b_r$$

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 $I_{2i} \to I_{2i+1}$ is correct, and another family that checks $I_{2i+1} \to I_{2i+2}$. We will only deal with the first case.

Here is machine $\mathcal{A}_k$, where $0 \leq k \leq p(n) - 3$.

- It reads $\#\Sigma^k xyz$ and remembers x, y and z;
- It then skips $p(n) - k - 3$ letters, reads a $\#$ and skips k more letters.
- Then the machine reads the next three letters x', y', z' and checks that they are compatible with the transition relation of the Turing machine.
- Lastly, it skips another $p(n) - k - 3$ letters, and either reads a $\#$ and starts all over, or reads $\$$ and accepts.

For example, if $x = p$, $y = a$ and $\delta(p, a) = (b, q, +1)$ then $x' = b$, $y' = q$ and $z' = z$. Of course, there may be multiple possibilities.

$\square$

The conversion of a nondeterministic automaton $\mathcal{A}$ to an equivalent deterministic $\mathcal{A}'$ relies on the standard Rabin-Scott power automaton construction. We are only interested in the case when $\mathcal{A}'$ is accessible.

Even so, we only have the exponential bound

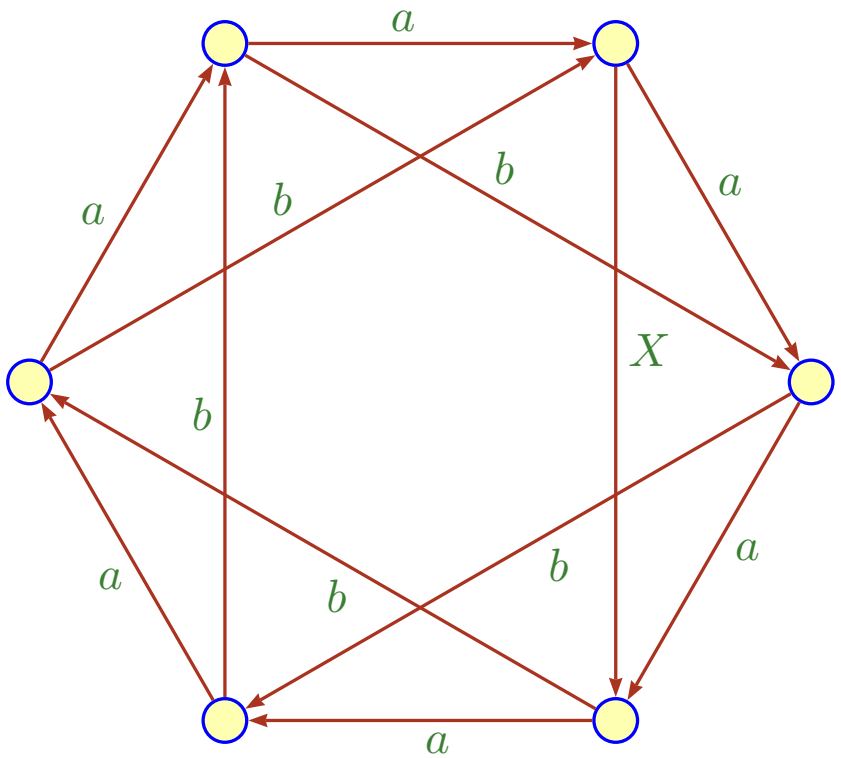
$$\mathsf{sc}(\mathcal{A}') \leq 2^{\mathsf{sc}(\mathcal{A})}$$

Unfortunately, this bound is tight in general. It remains tight when one considers only semiautomata where $Q = F = I$, even when they are nearly deterministic and co-deterministic.

Here is a 6-state semiautomaton.

$X = b$: machine is deterministic and co-deterministic, so the power automaton has size 1.

$X = a$: the power automaton has maximal size 2^6.



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.

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 $\mathfrak{P}(\mathcal{A})$?

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 $x \in \Sigma^\star$ (one symbol at a time) and verify that $I \xrightarrow{x} P$ in $\mathcal{A}$.

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 NSPACE from NL.

In practice, succinct representations of finite state machine by, say, Boolean Decision Diagrams are quite important.

Write $\pi(\mathcal{A})$ for the size of the accessible part of the power automaton of $\mathcal{A}$. So we would like to solve the following function problem:

Problem:	**Power Automaton Size**
Instance:	An NFA $\mathcal{A}$.
Solution:	Compute $\pi(\mathcal{A})$.

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

Problem:	**Power Automaton Size Bound**
Instance:	An NFA $\mathcal{A}$, a bound β.
Question:	Is $\pi(\mathcal{A}) < \beta$?

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

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 $\pi(\mathcal{A}) \geq \beta$.

The proof uses the DFA Intersection emptiness problem: let $\mathcal{A}_1, \ldots, \mathcal{A}_r$ be DFAs over some alphabet Σ. 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 $\mathcal{A}_i$. 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 $n_i + 2^{n_i} \leq 2^{p_i}$, for $i = 1, \ldots, r$.

We construct new machines $\mathcal{A}'_i$ by attaching a pointed cycle labeled b of length p_i to $\mathcal{A}_i$.

Each cycle node except the base-point has a self-loop labeled by c. The unique final state of $\mathcal{A}_i$ has a cone labeled b to the whole cycle C_i.

Lastly, we attach a self-loop labeled a to the initial state of $\mathcal{A}$.

The semiautomaton $\mathcal{A} = \bigcup \mathcal{A}'_i \cup C$ 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 Σ 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 $\mathcal{A}$.

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

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

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

For $X \subseteq [r]$ let $\alpha(X) = \prod_{i \in X} 2^{p_i}$.

For $x \in \Sigma^\star$ let

$$A(x) = \{\, i \in [r] \mid \mathcal{A}_i \text{ accepts } x \,\}$$

and $\alpha(x) = \alpha(A(x))$.

Let us say that an input string is proper if it is of the form

$$\Sigma^\star \, a \, \Sigma^\star \, b \, \Gamma^\star.$$

Consider proper inputs $w = a\,x\,b$, $x \in \Sigma^\star$.

One can check that one can generate $m\,\alpha(x)$ states from $Q \cdot axb$.

Hence, all proper inputs $uaxbv$ produce at least $m\,\alpha(x)$ states.

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

$\Sigma^\star$	$\prod 2^{n_i}$
$\Sigma^\star\, a\, \Sigma^\star$	$m \prod n_i$
$\Sigma^\star\, a\, \Sigma^\star (a+c)\Gamma^\star$	$m \prod 2^{p_i}$
$\Sigma^\star\, (b+c)\Gamma^\star$	$m \prod 2^{p_i}$

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

$$\gamma = 3\alpha([r]) + m\big(1 + \sum_x \alpha(x)\big),$$

where the summation is over suitably chosen factors $x \in \Sigma^\star$ in proper inputs.

Now consider the bound $\beta = m \cdot \alpha([r])$.

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

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

$$\begin{aligned}
\gamma/\beta &\leq \frac{3\alpha([r]) + m(1 + \sum_x \alpha(x))}{m \cdot \alpha([r])} \\
&= 3/m + \sum_x \alpha(x)/\alpha([r]) + 1/\alpha([r]) \\
&< 1/2 + \sum_i 2^{-p_i} + 2^{-\sum p_i} < 1.
\end{aligned}$$

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