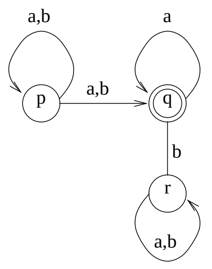
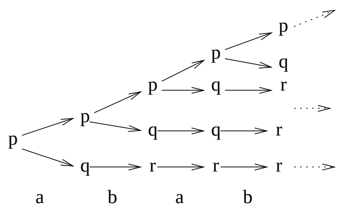
Lecture 11: Safra's Determinization Construction

In this lecture we shall see a construction due to S. Safra that turns a nondeterministic Büchi automaton of size n into an equivalent deterministic Rabin automaton of size $2^{O(nlogn)}$. As an immediate corollary we see that we can complement a nondeterministic Büchi automaton into a deterministic Streett automaton of size $2^{O(nlogn)}$.

Safra's construction is one of the masterpieces of this area. There are a number of articles that try to demystify and explain the construction [2, 4]. They try and explain why simpler ideas do not work and lead you to Safra's construction. At the risk of obfuscating matters I shall not take this route and instead try to arrive at Safra's construcion from an analysis of the run-trees of nondeterministic Büchi automata. The attempt here is to explain the special structure of the state space constructed as something that arises naturally from the structure of accepting runs (rather than as a complicated structure obtained by a sequence of refinements of simpler ideas that do not work).

A nondeterministic automaton has many runs on an input word. The deterministic automaton must simulate all those runs (or all the "relevant" ones) in a single one. The set of all runs of a NBA A on a word w would be an infinite tree in which each path is a run of A on w. (This is similar to the run of a universal alternating automaton, i.e. one in which all the states are $\wedge$ states. Of course, the acceptance criterion is different. We only demand that at least one path through this tree is accepting.) Here is NBA A_f and its run on the word $ababa\ldots$.



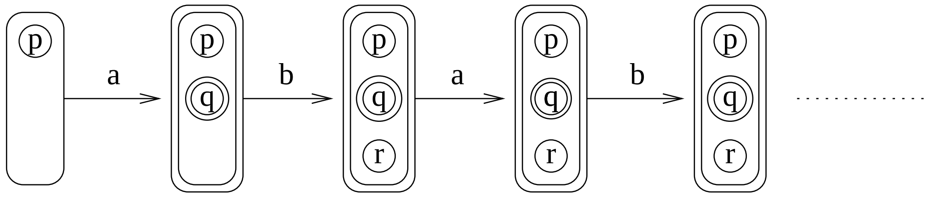


We could deterministically simulate this automaton level by level. That is pretty much all that we can do in a deterministic simulation. But the number of nodes in a level is not bounded. So, we need approximate and maintain some bounded amount of information. In the so called "power-set" construction, we approximate a level by the set of states that label some vertex at the level. Thus, we have an automaton of size 2^n. Next, we need to decide on the acceptance condition.

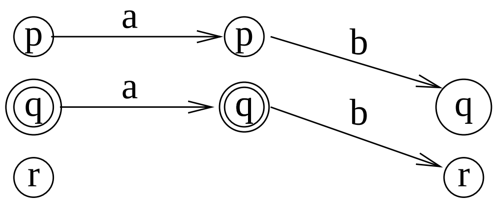
Demanding that that infinitely many levels contain final states is too generous. In the above run, the accepting state F appears in every level in the run tree, however there is no path in this tree that visits F infinitely often (and hence $ababab\ldots$ is not accepted by the above automaton). Thus the powerset automaton

$$A_p = (2^Q, \Sigma, \delta_p, \{s\}, \{X \mid X \cap F \neq \emptyset\}) \text{ with } \delta_p(X,a) = \{q \mid \exists p \in X.\ q \in \delta(p,a)\}$$

accepts more words than A. The power-set automaton constructed from A_f accepts $abab\ldots$ though A does not. Here is an accepting run:



For any two sets X, Y of states from Q and word u, let us write $X \xrightarrow{u}_g Y$ if for each $q \in Y$ there is a $q' \in X$ such that there is a run from q' to q on u that visits some state in F. For example, in the automaton A_f, $\{p,q,r\} \xrightarrow{ab}_g \{q,r\}$.



We now describe a different acceptance criterion for the powerset automaton. Let $S_0 \xrightarrow{a_1} S_1 \xrightarrow{a_2} \ldots$ be the run of the powerset automaton on $a_1a_2\ldots$. We accept provided we can decompose this run as $S_0 \xrightarrow{x_1} S_{i_1} \xrightarrow{x_2} S_{i_2} \ldots$ such that for each $q \in S_{i_{k+1}}$, there is a $q' \in S_{i_k}$ such that $q' \xrightarrow{x_{k+1}}_g q$.

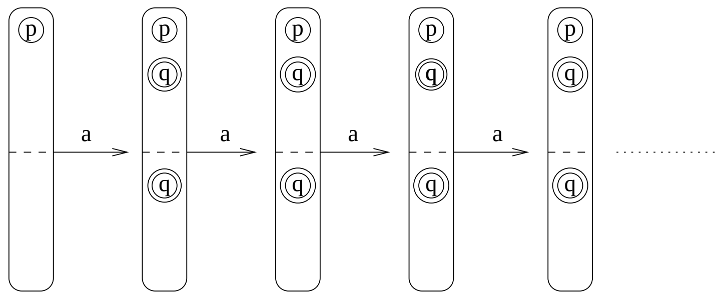
We claim that acceptance under this criterion guarantees that A accepts w. We can see this as follows: Construct a DAG whose nodes at level k are the states in S_{i_k}. An edge is drawn from a node at q' at level i to a node at level $i+1$ if and only if $q' \xrightarrow{x_{i+1}}_g q$. Label this edge by such a good run from q' to q. This finitely branching DAG has infinitely many nodes reachable from the vertices at level 1. Thus, there must be an infinite path through this DAG. The labels along the edges of this path gives an accepting run for A on w.

Can we translate this acceptance condition as a Büchi acceptance condition (by blowing up the state space if necessary?) The idea is similar to the one used to translate ABAs into NBAs. Let $A_m = (Q_m, \Sigma, \delta_m, (\{s\}, \emptyset), \{(X,X) \mid X \subseteq Q\}$ where

$$\begin{array}{lcl} Q_m & = & \{(X,Y) \mid Y \subseteq X \subseteq Q\} \\ \delta_m((X,Y),a) & = & (\delta_p(X,a), \delta_p(Y,a) \cup (\delta(X,a) \cap F)) \quad \text{if } X \neq Y \\ \delta_m((X,X),a) & = & (\delta_p(X,a), \delta_p(X,a) \cap F) \end{array}$$

We simulate the powerset automaton at one level. In the other level, we keep track of the subset of states that have been reached via paths that visited F. When these two sets are the same, we reset the second set and proceed. Thus we have a deterministic automaton A_m with $L(A_m) \subseteq L(A)$. In [2], the automaton A_m is called the *marked powerset* automaton.

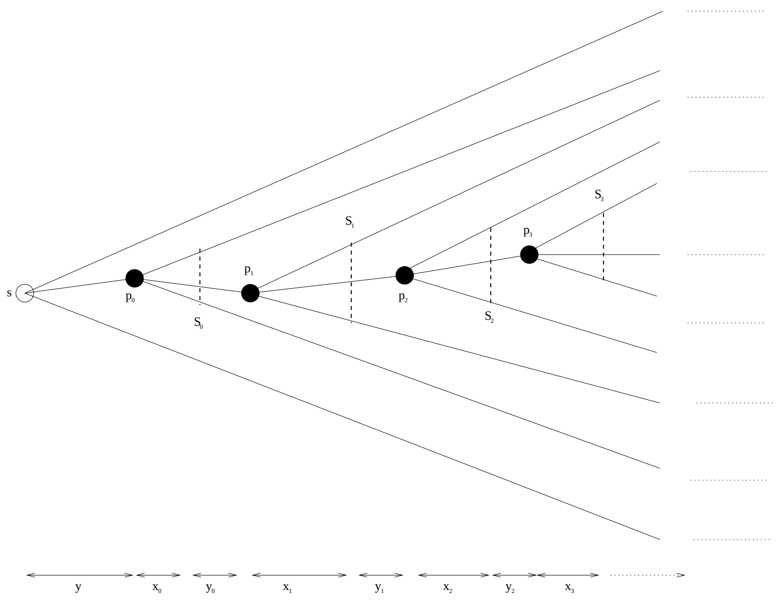
Quite clearly, $L(A_m)$ will not in general be the same as $L(A)$. For instance, the marked powerset automaton constructed from A_f does not accept $aaa\ldots$ which is accepted by A_f.



In general, if there is a dead state d(a nonaccepting state with a self-loop on all letters which is further not reachable from any accepting state) is reachable via some prefix of w then A_m will not accept w. This is because the state d appears in every level of the runtree beyond a finite prefix on every word and there is NO path in the automaton to reach d via an accepting state. How do we get around this? The naive idea would be somehow modify the construction of A_m to drop some elements of Q from the current state (so that any dead-states are dropped.) But that is what a nondeterministic automaton does! It simply drops all the states except the one that appears on the accepting path. So we need a better idea.

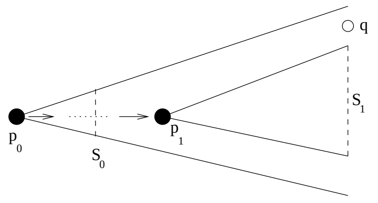
It turns out that what we need to do is to start the automaton $L(A_m)$ not right at the beginning, but instead start it after some initial prefix has been read. In some sense, we wait for all the useless paths to branch off and run the A_m on a "core" of the runtree. Of course, we have to do all this deterministically.

Consider the following figure. It describes a runtree of an NBA A on a word $w = a_1a_2\ldots$ which is accepted by A. In the figure, we have marked one accepting path in this runtree, and indicated the states of F that appear along this path by darkened circles.

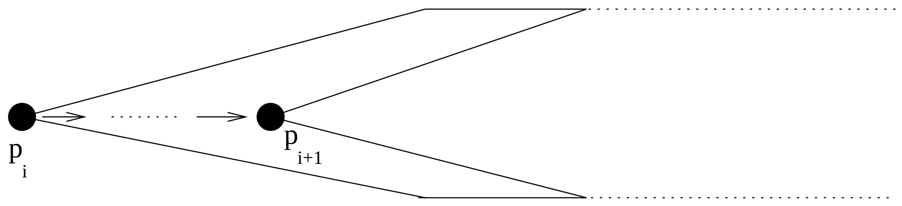


Here $p_0, p_1, \ldots$ are the states from F reached along the marked accepting run. We have placed slices $S_0, S_1, \ldots$ on the sub-runtree rooted at $p_0, p_1, \ldots$. S_0 is the set of nodes reached from p_0 on reading the segment x_0, S_1 is the set of nodes reached from p_1 on reading the segment x_1 and so on. $w = yx_0y_0x_1y_1\ldots$ (The segment y_i is the suffix obtained when we drop x_i from the word that takes p_i to p_{i+1}.)

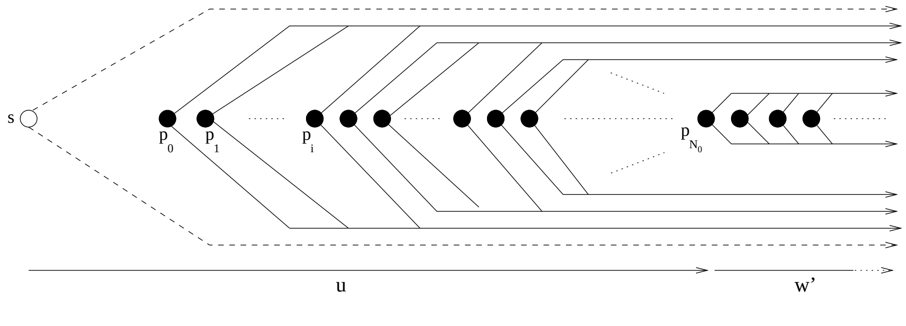
Notice that $S_i \xrightarrow{y_ix_{i+1}}_g S_{i+1}$. This might lead us to believe that we simply have to start the automaton A_m at p_0. Unfortunately, S_1 need not be set of states visited by the automaton A started at state p_0 (on $x_0y_0x_1$), i.e. $\delta_p(p_0, x_0y_0x_1)$ could be a strict superset of $S_1 = \delta_p(p_1, x_1)$. As indicated in the figure below, the state q may appear in $\delta_p(p_0, x_0y_0x_1)$ but not in S_1.



But then is $\delta_p(p_1, x_1y_1x_2)$ the same as $\delta_p(p_2, x_2)$ (i.e. can we start the A_m automaton at p_2?) Well, once again $\delta_p(p_1, x_1y_1x_2)$ could be a strict superset of $\delta(p_2, x_2)$. Note that if $\delta_p(p_i, x_iy_iu) = \delta_p(p_{i+1}, u)$ for some u then $\delta_p(p_i, x_iy_iuv) = \delta_p(p_{i+1}, uv)$ for all extensions v.



Now, let us say that p_i and p_{i+1} *merge* if there is an u such that $\delta_p(p_i, x_iy_iu) = \delta_p(p_{i+1}, u)$. We can easily extend this notion of merging to p_i and p_j for any pair of occurances of accepting states (along the accepting path under consideration.) If p_i merges with p_{i+1} and p_{i+1} merges with p_{i+2} then clearly p_i and p_{i+2} also merge. It is easy to check that merges is an equivalence relation that divides $p_0, p_1 \ldots$ into segments $(p_0, \ldots p_i)$, $(p_{i+1} \ldots p_j)$, ... Also note that if p_i and p_{i+1} do not merge, then for any extension u of x_iy_i, $\delta_p(p_i, x_iy_iu)$ is a strict superset of $\delta_p(p_{i+1}, u)$. But the set $\delta(p_i, u)$ is a subset of Q and hence finite. Thus, merge is an equivalence relation of finite index. Thus there is an equivalence class of merge consisting of $\{p_j \mid j \geq N_0\}$ for some N_0.



Suppose, that $w = uw'$ where $s \xrightarrow{u} p_{N_0}$, then we claim that the marked subset automaton started at state $p \in F$ labelling p_{N_0} accepts the word w'. That is, the automaton $A_m^p =$

$(2^Q \times 2^Q, \Sigma, \delta_m, (\{p\}, \emptyset), \{(X,X) \mid X \subseteq Q\}$ where δ_m is as before, accepts the word w'. Let (X_1, X_2) be the state reached after some prefix v of w'. Hence there is some j run has gone past p_j but not p_{j+1}. The states p_{N_0} and p_{j+1} must merge at some prefix w'_1 of w'. At that point the state of A^p_m would be (Y, Y) for some Y. Thus, the automaton A^p_m visits the accepting set infinitely often on w'.

Thus, our deterministic automaton must (deterministically!) guess a path in the runtree and a F labelled position in this path (corresponding to P_{N_0}) and run the marked subset automaton (which is a deterministic automaton) A^p_m from there on. Thus, in a manner quite reminiscent of the proof of McNaughton's theorem, we have shown that nondeterminism is needed only in a finite prefix of the run. The question is how to get rid of the initial nondeterminism.

As in the proof of McNaughton's theorem, the idea would be to "run all possible copies" instead of guessing which copy to run and merge copies whenever possible. That is, at each point in the run if the word read so far leads s to some final state p we fork off a copy of A^p_m at this point. We also merge copies that are at the same state. Let us examine the correspondence with the construction in Lecture 8 closely:

1. The simple powerset automaton A_p will play the role of A_U. Its role is to identify the prefixes at which copies of the deterministic Büchi automata are to be forked off.

2. The role of the deterministic Büchi automaton A_V will be played by a collection of deterministic Büchi automata $\{A^p_m \mid p \in F\}$. After reading a finite word w, if the automaton A_p has reached the state X then, for each $p \in X \cap F$ a copy of A^p_m is to be forked off.

3. Observe that if $p \neq p'$ then A^p_m and $A^{p'}_m$ differ only in the start state. That is, the state space of all the A^p_m is that of A_m. Thus the operation of merging different copies still makes sense. In particular, if two different copies are at the same state we merge them together.

4. The number of states in A^p_m is 2^{2n}. At each step, we could fork upto $|F|$ copies (depending on which states in F appear in the current state). To ensure that if a copy at slot j merges (with a lower numbered copy) down then slot j is empty (i.e. it carries the state $\perp$) in the next state, it suffices to keep $2^{2n} + |F|$ slots. This ensures that any slot that becomes vacant (because it has merged with some other copy) stays vacant at least for one move.

5. A run is accepting precisely when there is some slot j which is $\perp$ only finitely often (so that the copy j simulates some A^p_m on some suffix of w) and automaton at slot j visits the final state infinitely often (i.e. the state in the jth slot is of the form (X, X) infinitely often).

This leads us to the following construction of the (double exponential sized) deterministic automaton A^d accepting $L(A)$. Let $K = 2^{2n} + |F|$.

$$\begin{aligned} Q_d &= 2^Q \times (2^Q \times 2^Q \cup \{\perp\})^K \\ s_d &= (\{s\}, \perp, \perp, \ldots \perp) \end{aligned}$$

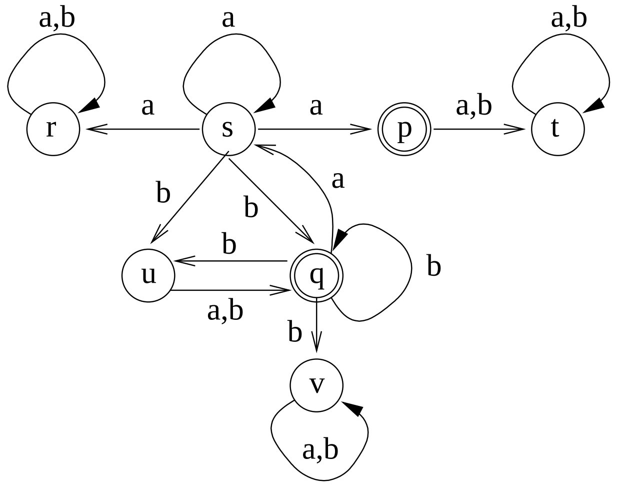
Let $(S, W_1, W_2, \ldots W_K)$ be a state. To compute $\delta_d((S, W_1, W_2 \ldots, W_K), a)$, first compute the tuple $V = (\delta_p(S, a), \delta_m(W_1, a), \delta_m(W_2, a) \ldots \delta_m(W_K, a))$ with the understanding that $\delta_m(\perp, a) = \perp$. Suppose $\{p_1, p_2, \ldots, p_l\} = \delta(S, a) \cap F$. We need to fork a copy of $A_m^{p_i}$ for each i. Pick the l lowest numbered slots in V that are occupied by $\perp$ and replace them by $(\{p_1\}, \emptyset), (\{p_2\}, \emptyset) \ldots (\{p_l\}, \emptyset)$ respectively. Now, if two or more slots are occupied by the same state of A_m, then replace all but the lowest numbered such slot by $\perp$. This V' is $\delta_d((S, W_1, W_2, \ldots W_K), a)$.

Finally, the acceptance condition is a Rabin acceptance condition consisting of K pairs (F_i, I_i). F_i contains all the tuples where the ith coordinate is $\perp$. I_i contains all the tuples where the ith coordinate is an accepting state of A_m, that is it is of the form (X, X) for some X. It is quite easy to check, using the same ideas as in the proof in Lecture 8, that this automaton accepts a word precisely when it is accepted by $L(A)$.

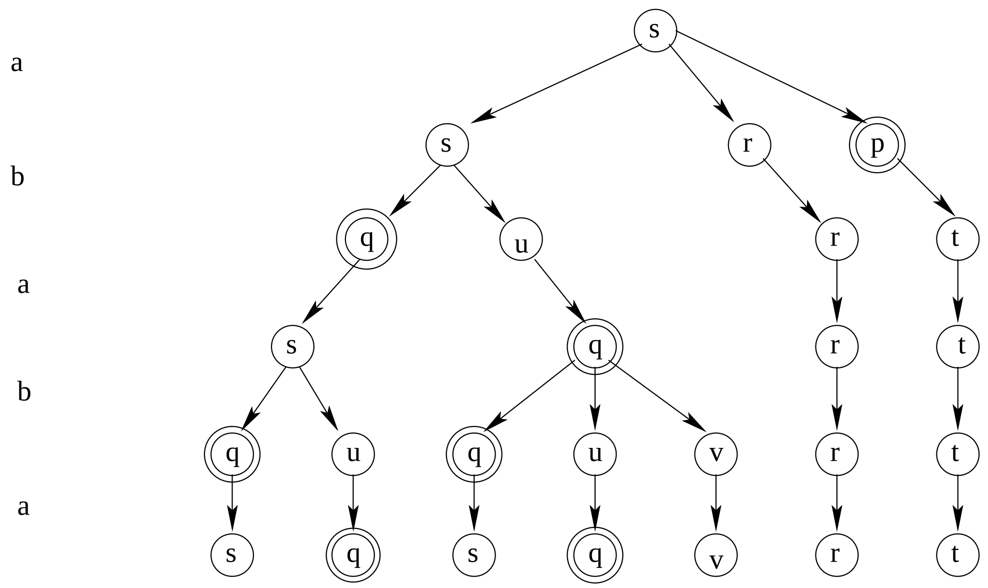
How to transform this double exponential sized automaton to a single exponential sized automaton is the topic of the next section.

1 Safra's Construction

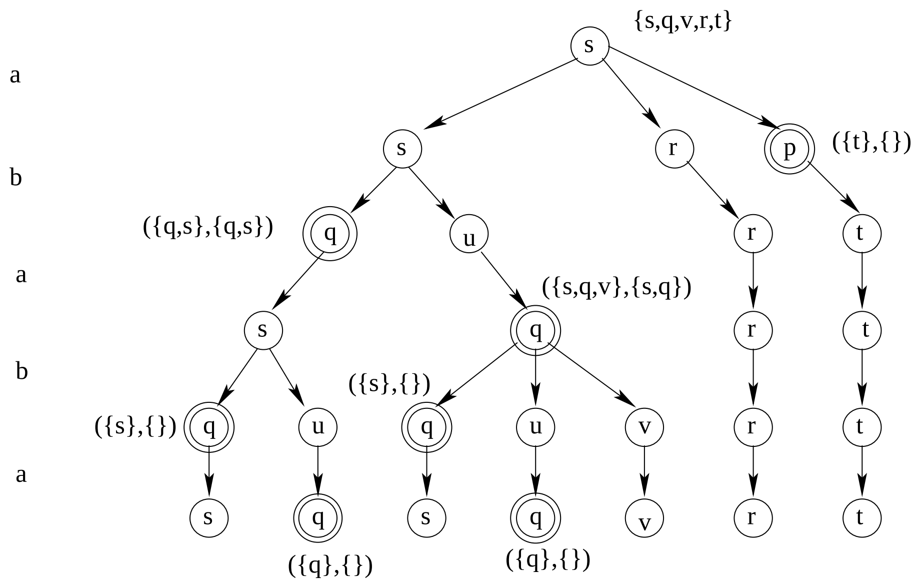
Safra's construction incorporates the structure of the runtree into the state space of the deterministic automaton. We shall illustrate this using the following complex automaton which accepts the set of all words with infinitely many bs:



Here is a prefix of the runtree of this automaton on the input $abab\ldots$.



There are seven occurances of accepting states in the run on the word *ababa*. As an when these states are encountered copies of A_m is forked out. The state reached by these seven copies is marked next to these states. Moreover the state reached by complete powerset automaton is marked next to the root.

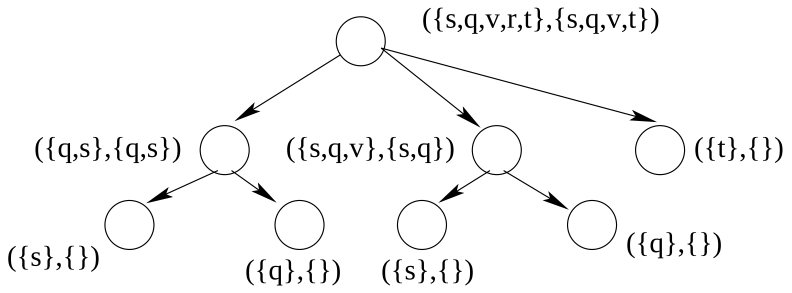


In the autmaton A^d this would have been represented by a tuple

$$(\{s,q,v,r,t\},(\{q,s\},\{q,s\}),(\{t\},\emptyset),(\{s,q,v\},\{s,q\}),(\{s\},\emptyset),(\{q\},\emptyset))$$

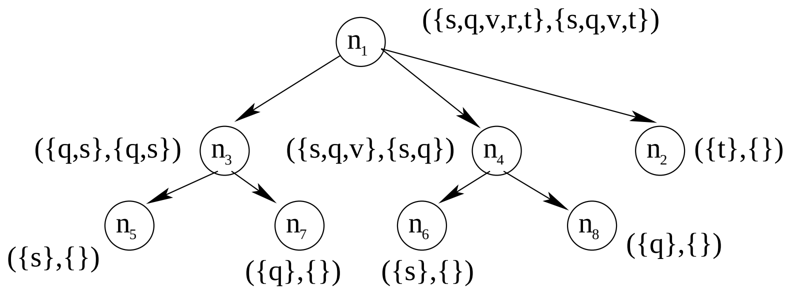
(with a large number of $\perp$s interspersed which we have omitted to avoid clutter).

Safra's construction stores more information. It records relationships between these various copies. For example, the three copies corresponding to the p at level 2, and the qs at level 3 and 4 are working on independent subtrees. There are two copies working in the subtree corresponding to the copy started w.r.t. the q at level 3 and so on. It would record these dependencies and store the state as a tree.



It also uses a copy of A_m instead of A_p for the automaton that forks out deterministic automata. This allows for a uniform treatment of all the nodes in the tree. (We have not eliminated duplicate copies here. We will get down to that soon.)

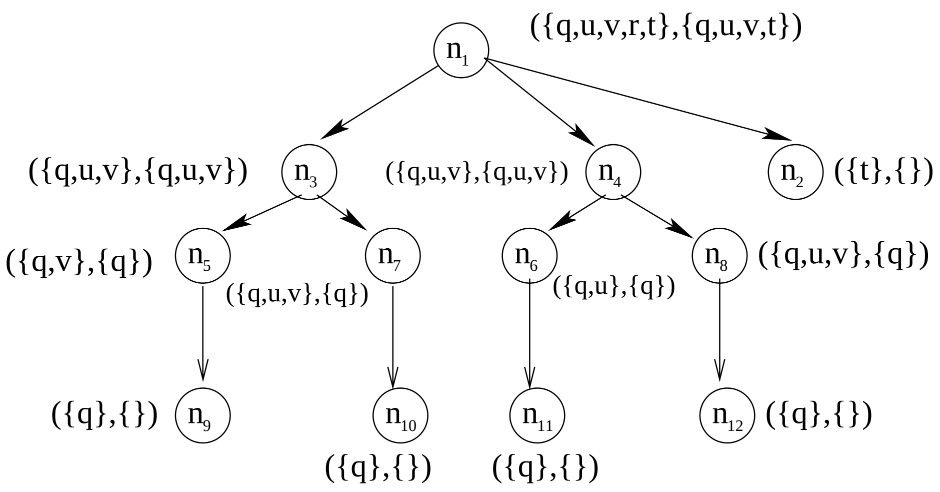
In A_d we maintained the state as a tuple and so the index i allows us to refer to a particular copy of A_m. To do this here, we assume that there is a an infinite ordered set of nodes $n_1 < n_2 \ldots$ and each time we add a node in the tree, we pick the next node from this sequence and use it. For example, with this the above tree would look like:



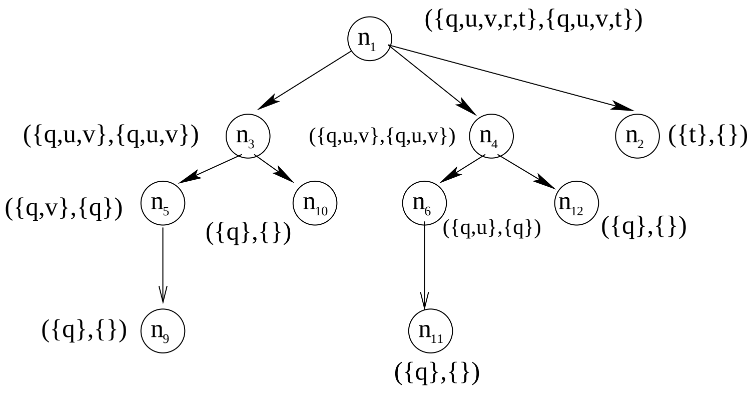
Notice that the nodes are numbered in the order in which the nodes were introduced into the tree (We assume some fixed ordering on Q so when multiple children are added to a node in a single step, the order is determined by the ordering on Q). To understand the behaviour of this automaton let us see what happens to this state when the next input letter, say a is read:

1. First, we simulate this move in each of the copies of A_m (one per node in the tree).

2. Then, we need to introduce new copies of A_m for each state in $f \in F$ that is reached at this state. Where do we introduce these copies? Notice that each such f appears along some set of paths starting at the root (since the set of states labelling a parent of a node is always a superset of that labelling a child). For each such path, find the highest level such node on that path and add a a child labelled $(\{f\}, \emptyset)$. In this example, it turns out that all these highest level nodes are leaves and so we add children to these leaves. But this need not always be the case. For the above automaton this gives:



3. If the sets labelling a node and its child are of the form (X, Y_1) and (X, Y_2) this indicates that the copies of A_m started at the node and its child have merged and so we simply delete the child and transfer all its children to the parent. In the above example, doing this we get:



Thus, unlike in A_d we do not actively merge any pair of copies that have reached the same state but only parent-child pairs that have reached the same state. Notice that this ensures that along any path in the tree the number of states in the first component of the node labels decreases strictly. Thus any path in such a tree is of length bounded by $|Q|$. On the other hand, there is no bound on the number of children of a given node.

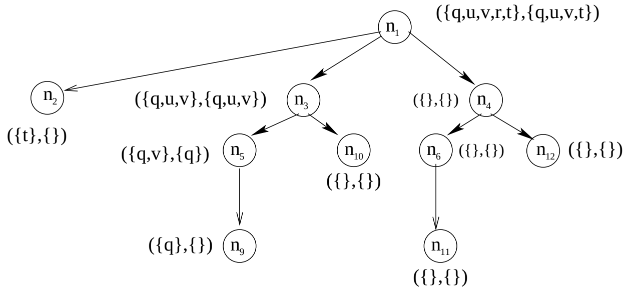
(**Observation 1:** Note that whenever a child of n merges with n, n must be labelled by a state of the form (X, X), i.e. an accepting state of A_m. This is because if a node labelled (X, Y) has children with labels $(X_1, Y_1) \ldots (X_k, Y_k)$ then $Y = X_1 \cup X_2 \ldots X_k$.)

The initial state of the automaton is the tree with a single node n_1 with the state $(\{s\}, \emptyset)$. The transition relation described above guarantees that in any reachable state, if the node n_i is the parent of n_j and the labels of n_i and n_j are (X_i, Y_i) and (X_j, Y_j) respectively, then

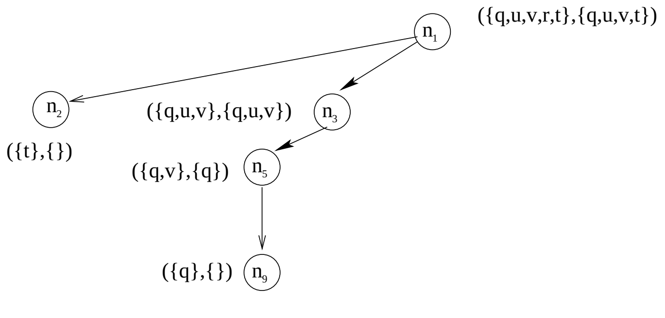
X_j is a strict subset of X_i. We take our set of states to be trees over $n_1, n_2 \ldots$ labelled by states of A_m that further satisfy this property. Thus the trees constituting states have depth bounded by $|Q|$. However, since we have an infinite set of labels the automaton we have constructed has infinitely many states.

Finally, we have to define the acceptance condition. It accepts a word w, if there is a node n_i such that along the unique infinite run on w n_i appears in all the states in some infinite suffix and moreover n_i is labelled by an accepting state of A_m infinitely often. The correctness of this construction is quite obvious: It simply maintains the state of A_d in a more complicated form (and with some redundancy) and so its correctness follows from the correctness of A_d.

So it seems like we have taken a step in the wrong direction: From a simple double exponential automaton to a complex infinite state automaton. But one brilliant observation of Safra takes us all the way to a $2^{O(nlogn)}$ construction. His observation is the following: If $q \in Q$ appears in the sets labelling multiple paths in the tree (i.e. a state of the Safra automaton), then we can drop it from all but one! (By this we mean that if a node n_i is not along the chosen path for q and its label is (X, Y) then we may replace this label by $(X \setminus \{q\}, Y \setminus \{q\})$). This choice of which path is to keep track of q has to be made carefully. Children of any node are ordered from left to right in the order in which they were inserted into the tree. Given two paths starting at the root, the one that moves left at the point of their branching is said to be to the left of the other. Safra's idea is to keep the state q in the left most path from the root (along which q appears). With this idea, the above state becomes:



We can delete nodes labelled $(\{\}, \{\})$ since they play no further role in the run. Thus the state reached on *ababa* would actually look like:



The modified automaton A_s, has as its set of states trees over the set nodes $n_1, n_2 \ldots$ labelled by states of A_m in such a way for any $q \in Q$, there is at the most one path (starting at the root) whose labels contain q. The transition relation is computed as above with the following additional step:

> 4. If q appears along labels along multiple paths starting at the root, delete it from all but the left-most path.

The acceptance criterion continues to be the same.

Why does this modified automaton A_s accept the same language as A (and A_d)? Every word accepted by A_s is accepted by A. This is because, dropping some states from the sets labelling the tree corresponds to running copies of A_m in such a way that occasionally we drop some states from the sets. This does not increase the family of words accepted as shown by the following exercise.

Exercise: Let A'_m be the automaton defined as follows:

$$A_m = (Q_m, \Sigma, \delta'_m, (\{s\}, \emptyset), \{(X, X) \mid X \subseteq Q\}$$

where

$$\begin{array}{lcll} Q_m & = & \{(X,Y) \mid Y \subseteq X \subseteq Q\} & \\ \delta'_m((X,Y),a) & = & \{(\delta_p(X,a) \setminus Z, \delta_p(Y,a) \cup \delta(X,a) \cap F \setminus Z) \mid Z \subseteq Q\} & \text{if } X \neq Y \\ \delta'_m((X,X),a) & = & \{(\delta_p(X,a) \setminus Z, \delta_p(X,a) \cap F \setminus Z \mid Z \subseteq Q\}) & \end{array}$$

Show that $L(A'_m) \subseteq L(A_m)$.
Hint: Simply show that if $(X, X \cap F) \xrightarrow{u} (Z, Z)$ in this automaton then for each $z \in Z$ there is some $x \in X$ such that $x \xrightarrow{u}_g z$.

So we are left with showing that any word accepted by A is also accepted by the modified automaton A_s. Let $\rho = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \xrightarrow{\cdots}$ be an accepting run. Let $T_0 \xrightarrow{a_1} T_1 \xrightarrow{a_2} \ldots$ be the corresponding run in A_s. We associate a sequence of nodes $N_1, N_2 \ldots$, $N_i \in T_i$ with the state q_i of ρ. In T_i, set N_i to be the node at the highest level number containing q_i (This node need not be a leaf.) Notice that at each step one of the following things may happen:

1. N_{i+1} is a child of N_i. This happens if $q_{i+1} \in F$ and the path from root N_{i+1} is the left-most path with q_{i+1}.

2. N_{i+1} is an ancestor of N_i. This happens of N_i and several of its ancestors have the same label as N_{i+1} and they all get merged with N_{i+1}.

3. N_{i+1} is reached via a “left jump”.

Observation 2: If the associated node reaches a node n from its descendents (by this we mean $N_{i+1} = n$ and it is reached using case 2 above) then n must be labelled by an accepting state of A_m. This follows from Observation 1 and the condition for merging states.
Observation 3: By Observation 1, we do not need to keep the second component of the

labels of the trees as this can be computed from the first component of the children. (Safra's construction uses this optimization.)

First we observe that if n_0 appears as N_i for infinitely many i then the node n_0 is labelled by an accepting state of A_m infinitely often. This is because, each time the path visits an $q_j \in F$, the node N_j would move to a child of n_0. But it returns to n_0 in the future. But this happens only when some child of n_0 merges with it. But, by Observation 2, n_0 must be labelled by an accepting state whenever this happens. Otherwise, there is a point j such that $N_i \neq n_0$ for all $i \geq j$. At the last point, say N_i, when the associated node moved from n_0 it must have moved to a child of n_0. Moreover there are only finitely many children of the root and in particular only finitely many to the left of N_i. So whenever the associated node returns to level 2 (i.e. the level of children of the root) there are only finitely many choices (it cannot ever reach a child of the root that appears to the right of N_i without visiting the root, but we have already moved beyond the last visit to the root). Thus, the only way to get to any of these is either by a merge from below or by a left jump. But left jumps to nodes at level 2 can happen only finite number of times (since there are only finitely many of these vertices that lie to the left of N_i at this level). Thus, there is point beyond which no vertex at level 2 is reached by a left jump. Beyond this point either some vertex at level 2 is visited infinitely often via merges from below, which (by Observation 2) guarantees that this node is labelled by accepting states of A_m infinitely often or else there is a point beyond which the run never returns to level 2. We may then repeat this argument with vertices at level 3 and so on. But since the depth of the tree is bounded by N it follows that some node is reached infinitely often by merges from below which guarantees that this node is labelled by accepting states of A_m infinitely often. Thus the run is an accepting run of the automaton A_s.

Now we turn this into a finite state automaton. First of all observe that the first component of the label of any child of n is a subset of the first component of the label of n. Further the first components of the children are all disjoint. Thus if we focus just on the first component, we start at the root and as we go down, we keep partioning the set into finer and finer parts. Thus, the total number of nodes in the tree is bounded by $2 * n$ where $|Q| = n$. At each step, we will add at the most $|F| \leq n$ new nodes. Thus, if we have a set of $3n + 1$ nodes (following the ideas in Lecture 8 or the construction of A_d above) we are guaranteed that any node that is dropped does not appear in the following state. With this we get a finite state automaton, with a Rabin accepting condition with $3n + 1$ pairs. In the ith pair (F_i, I_i), F_i consists of all the trees where n_i does not appear and I_i consists of all the trees where n_i is labelled by an accepting state of A_m.

What is the size of this automaton? How many trees of the kind described above can be formed of size $2 * n$ from a set of $3n + 1$ distinct nodes. By Cayley's theorem the number of trees over $O(n)$ nodes is of the order of $2^{O(nlogn)}$. We have ordered trees (i.e. different ordering of the children of any node yields differnet trees). From each unordered tree we may generate upto $n!$ different labelled trees. Thus the total number of ordered trees continues to be of the order of $2^{O(nlogn)}$. Given such a tree how many different ways can we label the nodes with subsets of Q satisfying the "partioning" property? Note that each $q \in Q$ appears (if at all) along one unique path. Thus, if we determine the tail (i.e. the highest numbered node) for each $q \in Q$, the labelling of the tree is uniquely determined. The number of functions from Q to the set of nodes is n^{3n+1}. Thus the total number of Safra trees is $O(2^{(O(nlogin)})$.

Our presentation of Safra trees is somewhat different from the one used by Safra in his paper. This is because of our attempt to arrive from an analysis of the run-tree. We shall outline the original construction briefly at the beginning of the next lecture.

References

[1] Christof Löding: *Optimal Bounds for Transformations of ω-automata*, Proceedings of the International Conference on Foundations of Software Techonology and Theoretical Computer Science (FSTTCS) 1999, Springer Lecture Notes in Computer Science 1738, 1999.

[2] Madhavan Mukund: *Finite Automata on Infinite Words*, Internal Report, SPIC Science Foundation, TCS-96-2, 1996. Available at http://www.cmi.ac.in/ madhavan/papers/ps/tcs-96-2.ps.gz

[3] S. Safra: *On the complexity of ω-automata*, Proceedings of the 29th FOCS, 1988.

[4] Wolfgang Thomas: *Languages, automata, and logic* In the Handbook of Formal Languages, volume III, pages 389-455. Springer, New York, 1997.