An Introduction/Disclaimer!

These are notes from a course I taught a motley group of 1st, 2nd and 3rd year B.Sc. students, Ph.D students and a few of my colleagues (I gave a few preparatory lectures to the first year students on regular languages, which is not part of this course.) in early 2006. These lectures constituted the first part of a three part course and deals with automata, logics, algebras and games over words. The second and third parts, if they ever see the light of day, would deal with trees/graphs and nonregular structures.

The course consisted of 17 lectures of about 1.5 hours each. The notes for the last 4 lectures are still in preparation. These 4 lectures dealt with LTL, its translation to alternating automata and a detailed proof of the expressive completeness of LTL w.r.t FO. (I have some slides from two tutorial lectures I gave at Guwahati in the summer of 2006 that cover part of this material and I have that made available here as a poor substitute.)

These notes are still in draft form and there are possibly a number of errors. I must also apologize for the absence of an appropriate listing of references in these notes. I hope to fix this lacuna sometime soon.

K. Narayan Kumar

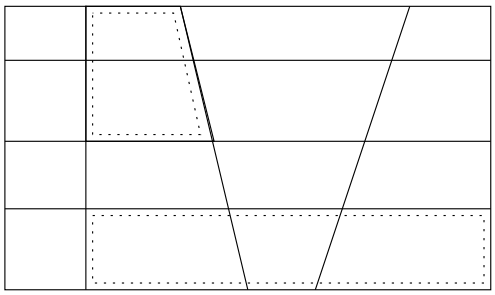
Lecture 1: Regular Languages and Monoids

We shall assume familiarity with the definitions and basic results regarding regular languages and finite automata (as presented in Hopcroft and Ullman [**?**] or Kozen [**?**]) and begin by recalling their connections to Myhill-Nerode relations.

1 Myhill-Nerode Characterization

An equivalence relation $\sim$ over Σ^*

- is a *right congruence* if $x \sim y$ implies $xz \sim yz$ for every $x, y, z \in \Sigma^*$
- is *of finite index* if $\Sigma^*/\sim$ is finite.
- *saturates* a language L if $x \sim y \;\Rightarrow\; (x \in L \text{ iff } y \in L)$. Or equivalently, L is the union of some of the equivalence classes of $\sim$, or equivalently for each $x \in \Sigma^*$, $[x]_\sim \cap L = \emptyset$ or $[x]_\sim \subseteq L$. This is illustrated by the following diagram. The entire rectangle corresponds to Σ^* and the individual regions inside are the equivalence classes under $\sim$ and the regions enclosed by the dotted lines are those that are contained in L. Note that every region is either entirely contained in L or is disjoint from L.



Theorem 1 *(Myhill-Nerode) A language L is regular if and only if there is a right congruence $\sim$ of finite index, that saturates L.*

From any finite automaton $A = (Q, \Sigma, \delta, s, F)$ recognising L it is easy to construct a right congruence $\sim_A$ of the desired kind.

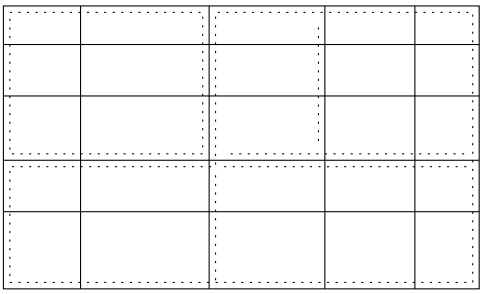
$$x \sim_A y \quad \Longleftrightarrow \quad \delta(s, x) = \delta(s, y)$$

For the converse, an automaton recognising L can be constructed as $A_\sim = (\Sigma^*/\sim, \Sigma, \delta, [\epsilon], F)$ where, $F = \{[x]_\sim \mid x \in L\}$ and $\delta([x]_\sim, a) = [xa]_\sim$.

With every language L (regular or otherwise) one can associate the "coarsest" right congruence that saturates L ($\sim_L$) as follows:

$$x \sim_L y \quad \Longleftrightarrow \quad \forall z.(xz \in L \iff yz \in L)$$

Quite evidently this relation is a right congruence that saturates L. It is also the coarsest because, if $\sim$ is any other right congruence that saturates L and $x \sim y$ then $x \sim_L y$ — suppose not, then there is an z such that (w.l.o.g.) $xz \in L$ and $yz \notin L$, contradicting the right congruent property of $\sim$. Thus, not only does $\sim_L$ saturate L, but further if $\sim$ is any right congruence saturing L then the equivalences classes of $\sim$ can be coalesced to form the equivalence classes of $\sim_L$.



In the above diagram the regions enclosed by the solid lines are the equivalence classes induced by $\sim$ and they are all entirely contained inside the equivalence classes induced by $\sim_L$ (the regions enclosed by the dotted lines).

If the language L is regular then $\sim_L$ is of finite index (since we can start with any finite index relation saturating L and coalesce its states to obtain $\sim_L$). The automaton obtained from $\sim_L$ is the *minimal* automaton for L.

An equivalence relation $\equiv$ is said to be a *congruence* if $x \equiv y$ implies $uxv \equiv uyv$ for all $u, v, x, y \in \Sigma^*$. It is quite easy to show that a language is regular if and only if there is congruence of finite index that saturates L. The construction of the automaton recognising L from the congruence is identical to the one described above. For the other direction, starting with an automaton $A = (Q, \Sigma, \delta, s, F)$ with no unreachable states, define $x \equiv_A y$ iff for all $q \in Q$, $\delta(q, x) = \delta(q, y)$. (Check that this relation is indeed a congruence and that it saturates L.)

One can also define a canonical congruence for each language L, given by $x \equiv_L y$ if and only if for all $u, v \in \Sigma^*$ $uxv \in L$ if and only if $uyv \in L$. Understandably, this is the "coarsest" congruence saturating L (verify this), so that starting with any other congruence saturating L, one may obtain this congruence by coalescing some of the equivalence classes together.

Exercise: Verify that if A is the minimal automaton for L then $\equiv_A$ is $\equiv_L$.

2 Monoids

A monoid is set M along with a associative binary operation . and a special element $e \in M$ which acts as the identity element w.r.t to .. We write $(M, ., e)$ to describe a monoid, but very often we shall write M instead. $(\mathbb{N}, +, 0)$ is a monoid and so is $(\Sigma^*, ., \epsilon)$ where . is the concatenation operation. These monoids are "infinite" as the underlying set is an infinite set. An example of a finite monoid is $(\mathbb{Z}_n, +, 0)$. Another class of finite monoids comes from

functions over a finite set. Let S be a set and let F be the set of functions from S to S and let Id_S be the identity function. Define $f \circ g$ to be the composition of g with f, i.e., $f \circ g(x) = g(f(x))$. Then $(F, \circ, Id_S)$ forms a monoid.

Given a finite automaton $A = (Q, \Sigma, \delta, s, F)$, the set of functions on Q defines a finite monoid. But there is a second and significantly more interesting monoid that one can associate with A. Let $M_A = (\{\hat{\delta}_x \mid x \in \Sigma^*\}, \circ, \hat{\delta}_\epsilon = Id_Q)$ where, $\hat{\delta}_x$ is the function from Q to Q defined by $\hat{\delta}_x(q) = \hat{\delta}(q, x)$. This monoid consists of those functions over Q that are defined as transition functions of words over Σ^*. Thus, it forms a *submonoid* of the set of functions over Q (any subset of a monoid containing the identity and closed w.r.t. the operation of the monoid is called a submonoid). This monoid associated with the automaton A is called the *transition monoid* of A and will play a critical role in the following developments.

A (homo)morphism from a monoid $(M, ., e)$ to a monoid $(N, *, f)$ is a function $h : M \longrightarrow N$ such that $h(x.y) = h(x) * h(y)$ and $h(e) = f$. For example, len : $\Sigma^* \longrightarrow \mathbb{N}$ with $\text{len}(x) = |x|$ is a morphism. The monoid $(\Sigma^*, ., \epsilon)$ is also called as the *free monoid* over Σ because, given any monoid $(N, *, f)$ and a function $f : \Sigma \longrightarrow N$, we can define a morphism $\hat{f}$ from $(\Sigma^*, ., \epsilon)$ to $(N, *, f)$ such that $\hat{f}(a) = f(a)$ for each $a \in \Sigma$ (the definition of $\hat{f}$ is quite obvious).

2.1 Monoids as Recognizers

We shall use monoids as recognizers of languages. Given a monoid $(M, ., e)$, a subset X of M and a morphism h from Σ^* to M, *the language defined by X w.r.t. to the morphism h* is $h^{-1}(X)$. We say that a language L is recognised by a monoid M if there is a morphism h and $X \subseteq M$ such that $L = h^{-1}(X)$. The interesting case is when M is a finite monoid.

Theorem 2 *L is a regular language if and only if it is recognised by some finite monoid.*

Proof: Suppose L is recognised by the monoid M via the morphism h and the subset X. Define the automaton $A_M = (M, \Sigma, \delta, e, X)$ where $\delta(m, a) = m.h(a)$. Then, $\hat{\delta}(m, a_1 a_2 \ldots a_n) = m.h(a_1).h(a_2). \ldots h(a_n)$ and therefore $\hat{\delta}(e, a_1 a_2 \ldots a_n) = e.h(a_1).h(a_2) \ldots h(a_n) = h(a_1 a_2 \ldots a_n)$. Thus, $L(A_M) = \{x \mid h(x) \in X\} = L(A_M) = L$.

For the converse, let A be any automaton recogising L. Consider the transition monoid $M_A = (\{\hat{\delta}_x \mid x \in \Sigma^*\}, \circ, Id_Q)$ and the morphism h from Σ^* to M_A defined by $h(x) = \hat{\delta}_x$. The pre-image under h of $X = \{\hat{\delta}_x \mid \hat{\delta}(s, x) \in F\}$ is easily seen to be L. Thus, A is recognised by a finite monoid. ■

2.2 The Syntactic Monoid

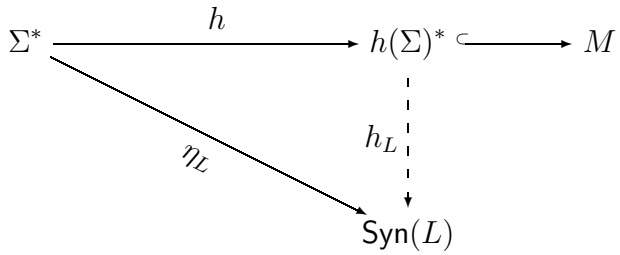
With each regular language L we can associate a canonical (in a manner to be explained soon) monoid that recognizes L. We associate a monoid structure on $\Sigma^*/\equiv_L$ by $[x]_{\equiv_L}.[y]_{\equiv_L} = [xy]_{\equiv_L}$. It is easy to check that with this operation $\Sigma^*/\equiv_L$ forms a monoid with $[\epsilon]_{\equiv_L}$ as the identity. The natural morphism η_L defined by $\eta_L(x) = [x]_{\equiv_L}$ recognises L as the pre-image of $X = \{[x]_{\equiv_L} \mid x \in L\}$. This monoid, denoted $\mathsf{Syn}(L)$, is called the *syntactic monoid* of L.

Exercise: Show by an example that $\sim_L$ is not a congruence in general. Thus, there is no monoid structure on $\Sigma^*/\sim_L$.

Exercise: What is the syntactic monoid of the language $(aa)^*$?

This monoid is canonical because, first of all, this is the smallest monoid that recognises L, and more importantly, η_L factors via every homomorphism (to any monoid M) that recognises L. This is the import of the following theorem

Theorem 3 *Let L be a regular language and suppose that L is recognised by the M via the morphism h. Then there is a morphism h_L from $h(M)$ (where $h(M)$ is the submonoid of M consisting of all the elements in the image of h) to $\mathsf{Syn}(L)$ such that $\eta_L = h \circ h_L$.*



Proof: Note that $\equiv_h$ defined by $x \equiv_h y$ if and only if $h(x) = h(y)$ is a congruence that saturates L: If $h(x) = h(y)$ then $h(uxv) = h(u)h(x)h(v) = h(u)h(y)h(v) = h(uyv)$. Thus, $\equiv_h$ is a congruence. Further, if $x \equiv_h y$ and $h(x) \in X$ then $h(y) \in X$. Hence it also saturates L. Thus, $\equiv_h$ is refined by $\equiv_L$ (i.e. each equivalence class of $\equiv_h$ is completely contained in some equivalence class of $\equiv$.)

Note that $\equiv_{\eta_L}$ is the same as $\equiv_L$. Hence, we may define the function h_L from $h(M)$ to $\mathsf{Syn}(L)$ as $h_L(h(x)) = \eta_L(x)$. This function is well-defined since we know that $h(x) = h(y)$ implies $\eta_L(x) = \eta_L(y)$. Clearly this map h_L is a morphism and by construction $h_L \circ h(x) = \eta_L(x)$. ■

Exercise: Prove that the syntactic monoid of a regular language L is isomorphic to the transition monoid of the minimal automaton for L.

We say that a monoid M *divides* a monoid N (written $M \prec N$) if M is the homomorphic image of a submonoid of N. In this language, the above theorem can be restated as

Theorem 4 *A monoid M recognises a regular language L only if $\mathsf{Syn}(L) \prec M$.*

We shall return to the study of regular languages via monoids after a couple of lectures. We shall see how we can use the structure of syntactic monoids to characterise subclasses of regular languages.

References

[1] John E. Hopcroft and Jeffrey D. Ullman: *Introduction to automata theory, languages and computation*, Addison-Wesley, 1979.

[2] Dexter Kozen: *Automata and Computability*, Springer-Verlag, 1997.

Lecture 2: Languages via Logical Formulae

Languages can be defined (and this is perhaps the most natural way to define them) as words satisfying a particular property. For example, the set of words satisfying each of the following properties is a regular language:

1. Every occurance of an a is eventually followed by a b.
2. There is exactly one a.
3. The first letter is an a.
4. The last letter is a a.
5. Every occurance of a is immediately followed by a b.
6. There are even number of a's.

However, the set of words which satisfy the property *there are equal number of as and bs* is not a regular language. So what identifies properties that define regular languages?

1 First order logic of words

A natural formal language to describe properties such as the ones described above is the *first order logic over words*. This logic is quite easy to understand and we shall illustrate this logic with a few examples. For instance, consider the formula,

$$\forall x.\ (a(x) \Rightarrow \exists y.\ ((y > x) \wedge b(x))$$

Here, the variables x, y etc. refer to positions in the word. The formula $a(x)$ asserts that the letter at position x is a. The quantifiers have the usual meaning. The formula $y > x$ is true if the position y appears somewhere to the right of the position x. Thus, a word w satisfies the above formula only if for any position (x) with the letter a, there is some position to its right (y) with the letter b. This captures in the language of first order logic the first property listed above.

The second property can be expressed as

$$\forall x.\ \forall y.\ (a(x) \wedge a(y)) \Rightarrow x = y$$

Consider the formula $\text{First}(x) \stackrel{\triangle}{=} \forall y.(x = y) \vee (x < y)$. This formula evaluates to true at a position x if and only if it is the first position in the word. With this, the third property can be expressed as

$$\forall x.(a(x) \Rightarrow \text{First}(x))$$

Consider the formula $(x < y) \wedge \forall z.(x < z) \Rightarrow (y \leq z)$ (where $y \leq z$ is $(y = z) \vee (y < z)$.) It asserts that the position y is the position immediately to the right of the position x. We

shall write $(y = x + 1)$ to denote this formula and using this we write the fourth property above as

$$\forall x.a(x) \Rightarrow \exists y.(y = x + 1) \wedge a(y)$$

As it turns out, the fifth property, which does describe a regular language, has no equivalent in the first order logic of words. This is one of the results that we shall prove. But for the moment, we shall turn to extending the first order logic of words so that all regular languages can be described.

2 Monadic logic over words

In first order logic, the only kind of variables we have are those that represent positions in the word. In *monadic logic*, in addition to the position variables, we are also allowed to use variables that represent sets of positions. We shall use $X, Y, \ldots$ to denote variables that range over sets of positions. We are also allowed to quantify over such variables. For example

$$\forall X.\forall x.(x \in X) \Rightarrow a(x)$$

asserts that in every subset of positions, every position it contains has the letter a. Clearly this is true of a word if and only if every position in the word has the letter a. The following formula

$$\forall X.\exists x.(x \in X) \wedge a(x)$$

is not true of any word!! This is because, when X is the empty set of positions $\exists x.(x \in X) \wedge a(x)$ cannot be satisfied.

Once we have quantification over sets, we can describe the set of even length words as follows:

The set of all positions can be divided into two sets X and Y such that

1. The first position is in X
2. The last position is in Y
3. for each position x if x is in X then the next position (if it exists) lies in Y
4. for each position x if x is in Y then the next position (if it exists) is in X

The third and fourth properities ensure that the positions of the word are alternately in X and Y and the first and second assertions ensure that there are even number of alternations and thus the word must have even number of positions! The monadic formula expressing the above properties is:

$$\begin{array}{ll} \exists X.\exists Y. & (\forall x.(x \in X) \iff (x \notin Y) \\ & \wedge\forall x.\mathrm{First}(x) \Rightarrow (x \in X) \\ & \wedge\forall x.\mathrm{Last}(x) \Rightarrow (x \in Y) \\ & \wedge\forall x.\forall y.\ ((x \in X) \wedge (y = x + 1)) \Rightarrow (y \in Y) \\ & \wedge\forall x.\forall y.\ ((x \in Y) \wedge (y = x + 1)) \Rightarrow (y \in X)) \end{array}$$

The first line asserts that the set of all positions is divided into two sets X and Y and the following four lines describe the four properties mentioned above.

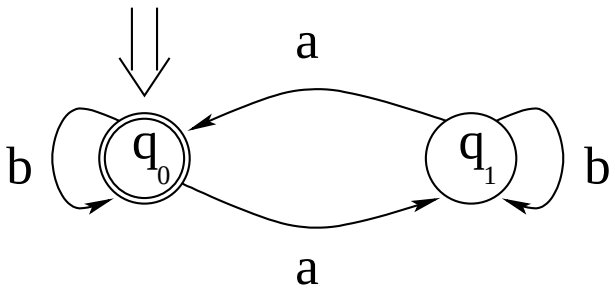
This extended logic which includes the use of set variables (and $x \in X$) is called the second-order monadic logic of words (or second-order monadic logic of order). We shall often write MSO to denote this logic. It is also called S1S (or second-order monadic logic of one successor).

3 Monadic logic and regular languages

Büchi and Elgot showed that the class of languages definable using formulas in monadic logic over words is precisely the class of regular languages.

Theorem 1 *(Büchi/Elgot) A language L is regular if and only if it can be described using a formula in MSO.*

The translation from a finite automaton accepting a regular language to a MSO formula describing the same language is the easier direction. Before we give the details, we shall illustrate the ideas with a simple example. Consider the following automaton with 2 states.



Let us examine a run of this automaton on some word, say *ababaa*.

$$q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_1 \xrightarrow{a} q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_0$$

Observe that a run on a word of length n involves a sequence of states of length $n+1$. For the moment if we omit the last state reached on a input of length n, we have a sequence of of states of length n. Thus, we can think of the run (without the last state) as a labelling of the positions of the word with states (the state reached before the letter at that position is read). Can we then write out a formula that "describes" the run of the automaton on any given word?

Here is an outline of how to do this: The trick is to use one second order variable X_q for each $q \in Q$, and use X_q to pick out the positions that are labelled by state q. Let the states of the automaton be $q_0, q_1, \ldots q_k$. (Thus, for the above example we use two variables X_{q_0} and X_{q_1}.) We then assert that the set of positions can be decomposed into $|Q|$ sets, $X_{q_0}, X_{q_1}, \ldots X_{q_k}$ such that

1. The first position belongs to X_{q_0} (since q_0 is the start state of the automaton).

2. If x and $x+1$ are positions in the given word and the letter at position x is some a then, x and $x+1$ belong to sets X_p and X_q such that $\delta(p,a) = q$.

3. If x is the last position in the word w, the letter at this position is a and x belongs to X_q then $\delta(q,a) \in F$.

For the above automaton these assertions can be expressed in MSO as follows:

$$\begin{array}{ll} \exists X_{q_0}.\exists X_{q_1}. & (\forall x.(x \in X_{q_0}) \iff (x \notin X_{q_1}) \\ & \wedge \forall x.\mathrm{First}(x) \Rightarrow (x \in X_{q_0}) \\ & \wedge \forall x.\forall y.\ ((x \in X_{q_0}) \wedge a(x) \wedge (y = x+1)) \Rightarrow (y \in X_{q_1}) \\ & \wedge \forall x.\forall y.\ ((x \in X_{q_0}) \wedge b(x) \wedge (y = x+1)) \Rightarrow (y \in X_{q_0}) \\ & \wedge \forall x.\forall y.\ ((x \in X_{q_1}) \wedge a(x) \wedge (y = x+1)) \Rightarrow (y \in X_{q_0}) \\ & \wedge \forall x.\forall y.\ ((x \in X_{q_1}) \wedge b(x) \wedge (y = x+1)) \Rightarrow (y \in X_{q_1}) \\ & \wedge \forall x.(\mathrm{Last}(x) \Rightarrow ((x \in X_{q_0} \wedge b(x)) \vee (x \in X_{q_1} \wedge a(x))) \end{array}$$

The first line says each position is labelled either with q_0 or q_1 (to represent the state reached before leading the letter at that position). The second line asserts that state corresponding to the first position is the start state. The subsequent four lines encode the four possible transitions of the automaton. The last line ensures that the state assigned to the last position is such that the transition from this state on the letter at the last position takes the run into a final state.

In general given a finite automaton $A = (\{q_0, \ldots, q_k\}, \Sigma, \delta, q_0, F)$, the following formula asserts that " there is an accepting run for A over the given word". Thus the models of this formula are precisely the words in the language accepted by A.

$$\begin{array}{ll} \exists X_{q_0}.\exists X_{q_1} \ldots \exists X_{q_k}. & \forall x.\ \bigvee_{0 \le i \le k} x \in X_{q_i} \\ & \forall x.\ \bigwedge_{i \neq j} (x \in X_i) \Rightarrow \neg(x \in X_j) \\ & \forall x.\mathrm{First}(x) \Rightarrow x \in X_{q_0} \\ & \bigwedge_{\delta(p,a)=q} \forall x.((x \in X_p) \wedge a(x) \wedge (y = x+1)) \Rightarrow (y \in X_q) \\ & \forall x.\mathrm{Last}(x) \Rightarrow (\bigvee_{\delta(p,a) \in F}((x \in X_p) \wedge a(x))) \end{array}$$

The first and second lines assert that the set of positions is decomposed into a collection of sets, one for each state in Q. The the next three lines assert the 3 properties mentioned above and thus a word satisfies this property if and only if it is accepted by the automaton A. This completes a sketch of the argument showing that every regular language can be described by a MSO formula.

Notes: In this lecture and the next our presentation follows that of Straubing [1].

References

[1] Howard Straubing: *Finite Automata, Formal Logic and Circuit Complexity*, Birkhäuser, 1994.

Lecture 3: MSO to Regular Languages

To describe the translation from MSO formulas to regular languages one has to be a bit more formal! All the examples we used in the previous class were *sentences* i.e., every variable that occured in the formula occured within the scope of a quantifier. (A variable that is tied to quantifier is called a *bound* variable. Every variable in a sentence is a bound variable.) Given a sentence ϕ, any word w either satisfies ϕ or does not.

However, in order to reason about sentences, one has to reason about subformulas of sentences and these need not be sentences. As a matter of fact, subformulas of sentences are usually NOT sentences.

For example, consider formulas $\text{First}(x)$ and $y = x + 1$ that were used repeatedly in the previous lecture. The former has x as a *free variable* while the latter has x and y as free variables. A free variable is one that is not "captured" by a quantifier. It does not make sense to ask if w satisfies $\text{First}(x)$. Instead, one has to give a word w and a position i in the word w and then one may ask if $\text{First}(i)$ is true. Similarly to evaluate $y = x + 1$, one needs values (i.e. positions) for the variables x and y before we can verify its truth.

For the moment let us restrict overselves to first order formulas. Then, to meaningfully discuss the truth or falsity of a formula with k free variables, we need a word along with assignment of positions to the k variables. For example, the formula $\phi = (x < y) \wedge a(x) \wedge b(y)$ is true of *bacabc* with x assigned position 2 and y assigned position 5. We represent such a word with assignments for x and y as a word decorated with the variables x and y as follows:

b	a	c	a	b	c
	x			y	

On the other hand the formula ϕ is not true of *bacabc* if x and y are assigned position 5.

b	a	c	a	b	c
				x	
				y	

Notice that these decorated words can themselves be thought of as words over the alphabet $\Sigma \times 2^V$ where V is the set of (free) variables. For instance, the two decorated models correspond to the words $(b, \emptyset)(a, \{x\})(c, \emptyset)(a, \emptyset)(b, \{y\})(c, \emptyset)$ and $(b, \emptyset)(a, \emptyset)(c, \emptyset)(a, \emptyset)(b, \{x, y\})(c, \emptyset)$ respectively. Often we shall a for $(a, \emptyset)$ and write $b(a, \{x\})ca(b, \{y\})c$ and $baca(b, \{x, y\})c$ instead.

For the purposes of this lecture let us fix the basic alphabet to be Σ. Following Straubing [2], given a set V of variables we define the set of V-words to be words over the alphabet $\Sigma \times 2^V$ to be those that describe a word over Σ and indicate the positions of all the variables V in the word. Formally, a V-word is a word $(a_1, U_1)(a_2, U_2) \ldots (a_k, U_k)$ where

1. $U_i \cap U_j = \emptyset$ for $i \neq j$.
2. $\bigcup_{1 \leq i \leq k} U_i = V$.

Thus, a V-word associates a unique position of the underlying word, over the alphabet Σ, with each variable in V.

Given a formula ϕ with all of its free variables (free(ϕ)) coming from V and a V-word w we can define whether w satisfies ϕ in the obvious way (A formal definition is given in the appendix). Thus, if free(ϕ) $\subseteq V$ then ϕ defines a language of V-words. (In particular, when free(ϕ) is empty, the ϕ defines a language over the set of words over Σ, the set of all words that satisfy ϕ.)

How do we extend this to MSO formulas? Notice that in order to evaluate a formula with a free second order variable X we need to associate a set of positions with X. We could use the decorating technique and simply decorate each position that belongs to the set associated with X by X. Of course, there might be no positions decorated with X, indicating that X is the empty set.

For instance, consider the formula $a(x) \wedge (x \in X) \wedge (y \in X)$. The following decorated word, where x is assigned position 2, y is assigned position 5 and X is assigned the set $\{2, 3, 5\}$ of positions, satisfies this formula.

b	a	c	a	b	c
	x			y	
	X	X			X

On the other hand,

b	a	c	a	b	c
	x			y	
		X			X

does not.

Extending the idea used for FO, we define (V, W)-words to be words over the alphabet $\Sigma \times 2^V \times 2^W$ that describe positions for the variables in V and sets of positions for the variables in W. Formally, a word $(a_1, U_1, W_1)(a_2, U_2, W_2) \ldots (a_k, U_k, W_k)$ is a (V, W)-word if it satisfies:

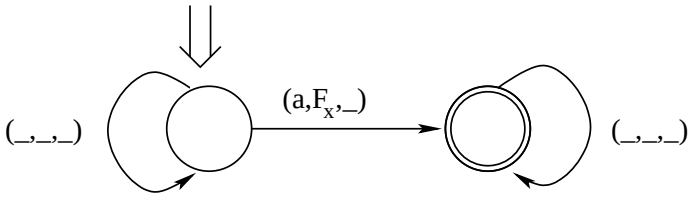
1. $U_i \cap U_j = \emptyset$ for $i \neq j$.
2. $\bigcup_{1 \leq i \leq k} U_i = V$.

Thus, given a formula ϕ whose set of first order free variables free$_1(\phi)$ is contained in V and whose set of second order free variables free$_2(\phi)$ is contained in W, and a (V, W)-word w we can define, in the obvious way, whether w satisfies ϕ (a formal definition is given in the appendix). Thus, such a formula, defines a language of (V, W)-words.

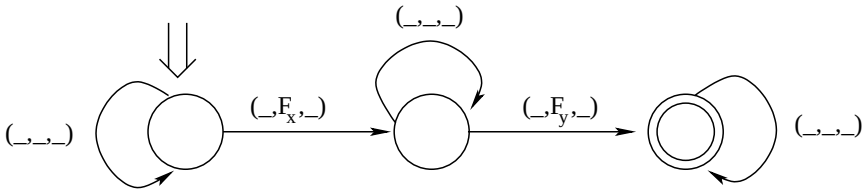
We shall show, by induction on the structure of the formula ϕ that the language (V, W)-words defined by ϕ for any V, W with free$_1(\phi) \subseteq V$ and free$_2(\phi) \subseteq W$ is a regular language over $\Sigma \times 2^V \times 2^W$.

It is quite easy to write down a finite automaton that accepts the language of all (V, W)-words. Since regular languages are closed under intersection, in what follows we will assume that only valid (V, W)-words are considered as valid input.

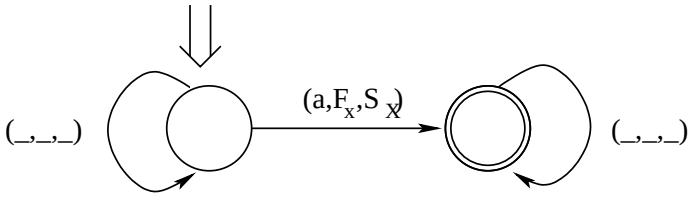
For the basis, we consider the atomic formulas. There are three choices $a(x)$, $x < y$ and $x \in X$. Here is an automaton that accepts (V, W)-words that satisfy $a(x)$.



where a _ stands for "any" and F_x is any subset of V that contains x. Similarly, an automaton that accepts $x < y$ is the following:



where F_y is any subset of V that contains y. Finally, here is an automaton that accepts any (V, W)-word that satisfies $x \in X$,



where S_X is any subset of W that contains X. Note, that the correctness of these three automata relies on the fact that the input consists only of (V, W)-words, but we can always ensure this by taking the product of this automaton with any finite automaton accepting the set of (V, W)-words.

For the induction step, we need to consider the various choices of logical operators. If $\phi = \alpha \wedge \beta$ then, by induction hypothesis we have automata for the languages accepted by α and β and we know that finite automata are closed under language intersection. Similarly if $\phi = \neg\alpha$, we can complement the automaton recognising the set of (V, W)-words satisfying α (and intersect it with the set of valid (V, W)-words). The other operators like $\vee$ and $\Rightarrow$ can be expressed using $\wedge$ and $\neg$. Thus we are left with the quantifiers. Note that $\forall x.\phi(x)$ is $\neg(\exists x.\neg\phi(x))$ and thus it suffices to consider the first order and second order existential quantifiers.

Suppose $\text{free}_1(\exists x.\phi) \subseteq V$ and $\text{free}_2(\exists x.\phi) \subseteq W$. Then, $\text{free}_1(\phi) \subseteq V \cup \{x\}$ and $\text{free}_2(\phi) \subseteq W$. Therefore, by the induction hypothesis, the set of $(V \cup \{x\}, W)$-words that satisfy ϕ is a regular language. Further, it is quite easy to see that if

$$\begin{array}{cccccc} a_1 & a_2 & \ldots & a_i & \ldots & a_k \\ F_1 & F_2 & \ldots & F_i \cup \{x\} & \ldots & F_k \\ S_1 & S_2 & \ldots & S_i & \ldots & S_k \end{array}$$

satisfies ϕ then

$$\begin{array}{cccccc} a_1 & a_2 & \ldots & a_i & \ldots & a_k \\ F_1 & F_2 & \ldots & F_i & \ldots & F_k \\ S_1 & S_2 & \ldots & S_i & \ldots & S_k \end{array}$$

satisfied $\exists x.\phi$ (simply choose x to be the position i). Conversely, when

$$\begin{array}{cccc} a_1 & a_2 & \ldots & a_k \\ F_1 & F_2 & \ldots & F_k \\ S_1 & S_2 & \ldots & S_k \end{array}$$

satisfies $\exists x.\phi$ then, then there is a choice of position, say i, for x such that ϕ is satisfied w.r.t. to this assignment and thus,

$$\begin{array}{cccccc} a_1 & a_2 & \ldots & a_i & \ldots & a_k \\ F_1 & F_2 & \ldots & F_i \cup \{x\} & \ldots & F_k \\ S_1 & S_2 & \ldots & S_i & \ldots & S_k \end{array}$$

satisfies ϕ. Thus, set of (V, W)-words that satisfy $\exists x.\phi$ are just the images of the set of $(V \cup \{x\}, W)$-words that satisfy ϕ under the homomorphism h defined by $h((a, F, S)) = (a, F \setminus \{x\}, S)$. Since, regular languages are closed under homomorphic images, we conclude that the set of (V, W)-words satisfying $\exists x.\phi$ is a regular language.

The proof in case of $\exists X.\phi$ is almost identical. In this case, the set of (V, W)-words that satisfy $\exists X.\phi$ are just the images of the set of $(V, W \cup \{X\})$-words that satisfy ϕ under the homomorphism h defined by $h((a, F, S)) = (a, F, S \setminus \{x\})$. Thus the set of words satisfying $\exists X.\phi$ forms a regular language.

Thus we have proved that whenever $\text{free}_1(\phi) \subseteq V$ and $\text{free}_2(\phi) \subseteq W$, the set of (V, W)-words that satisfy ϕ is a regular language over $\Sigma \times 2^V \times 2^W$. Thus, if ϕ is a sentence then the set of $(\emptyset, \emptyset)$-words that satisfy ϕ is a regular language over $\Sigma \times 2^\emptyset \times 2^\emptyset$ (and this is the same as the language over Σ, via the bijection that sends $(a, \emptyset, \emptyset)$ to a).

Thus, we have established both directions of Büchi's theorem.

1 Stratification of FO formulas

We now turn our attention to showing that the language of words with even number of as is not definable in the first-order logic of words. We define the quantifier depth of a f.o. formula ϕ as follows: if ϕ is quantifier-free than $\mathsf{qd}(\phi) = 0$. Otherwise, $\mathsf{qd}(\phi \wedge \phi') = \mathsf{maximum}(\mathsf{qd}(\phi, \phi'))$, $\mathsf{qd}(\neg\phi) = \mathsf{qd}(\phi)$ and $\mathsf{qd}(\exists x.\phi) = \mathsf{qd}(\phi) + 1$.

Now, if we fix a finite set F of variables, there are only finitely many quantifier-free formulas over F upto logical equivalence. Simply rewrite the formula into its equivalent disjunctive normal form (i.e. a formula of the form $P_1 \vee P_2 \vee \ldots P_k$ where each $P_i = A_1 \wedge A_2 \wedge \ldots A_{k_i}$ is a conjunction of literals (i.e. each A_i is either a atomic formula or the negation of an atomic formula) and use the fact that $\phi \wedge \phi = \phi$ and $\phi \vee \phi = \phi$.

Next, observe that $\mathsf{qd}(i+1)$ formulas are just boolean combinations of formulas with quantifier depth $<= i$ and formulas of the form $\exists x.\phi$ where $\mathsf{qd}(\phi) \leq i$. Thus, if the number of formulas (upto logical equivalence) of quantifier depth i or less is finite then the number of formulas with quantifier depth $i+1$ or less is also finite.

The previous two paragraphs (when formalise appropriately!) yields the following theorem.

Theorem 1 *For any i there are only finitely many formulas of quantifier depth i or less (upto logical equivalence).*

Thus, we can stratify first order definable languages via the quantifier depth necessary to define a language. One method to show that a particular to language is not first order definable is to show that for each k, no sentence of quantifier depth k can define the language. This is the route we shall take in order to show that evenness is not first order definable.

This leads us to the natural question: How do we show that a language is not definable via sentences of quantifier depth k? Well, this is done by finding two words w and w', one in the language and another outside the language and show that these cannot be *distinguished* by sentences of quantifier depth k or less. That is, we show that for each sentence ϕ of quantifier depth k or less, either both w and w' satisfy ϕ or neither satisfies ϕ.

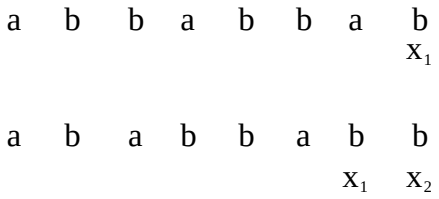
Thus we move on to the following question: Given two words w and w' how do we decide whether there is a sentence of quantifer depth k (or less) that distinguishes w from w'? It is here that *Ehrenfeucht-Fraisse games* play their role. Given w,w' and k we set up a game between two players 0 and 1 (the *cynic* and the *believer*) such that w is distinguishable from w' by some sentence of quantifier depth k or less if and only the player 0 has a winning strategy in the game.

1.1 Ehrenfeucht-Fraisse Games

Let w and w' be two V-words and let k be some positive integer. There are k-rounds in the game. In each round, say round i, player 0 (who is trying to show that these two words are distinguishable) picks one of the two words and a position in that word and labels it with a new variable x_i. Player 1 must then pick the other word (the one not picked by player 0), and label one of its positions with x_i. Thus, at the end of k rounds we have two $V \cup \{x_1, \ldots x_k\}$-words. Player 0 wins the game if there is some quantifier-free formula (over $V \cup \{x_1, \ldots, x_k\}$) that distinguishes these two words. Otherwise player 1 wins the game. Notice that this forces player 1 to try and duplicate player 0's moves as closely as possible so that the labellings are indistinguisable via atomic propositions.

We say that two V-words w and w' are *k-equivalent* if player 1 has a winning strategy to win the k round game on w and w'. We write $w \equiv_k w'$ to indicate this. On the logical side, we say that two V-words w and w' are *k-indistinguishable* if no quantifier depth k formula with free variables in the set V can distinguish between these two words. We write $w \sim_k w'$ to indicate this.

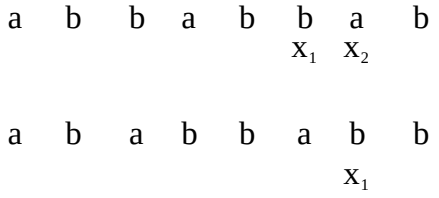
Here is a 2 round game played on the words $w = abbabbab$ and $w' = ababbabb$. Player 0 picks w' and labels position 7 with x_1.



Now, player 1 must pick some position, with a b, and to represent the "equivalent" in w of position 7 in w'. But this is doomed to fail.

If player 1 picks position 8 then in the second round player 0 would pick position 8 in w' and this leaves us at the following situation: Now, no matter where player 1 places x_2 it would violate atomic formula $x_1 < x_2$ satisfied by w'.

On the other hand, if player 1 picks any position other than 8, then player 0 would pick w in the second round and label position 7 with x_2. Here is the result (where player 1 played position 6 in the first round):



Once again, no matter where player 1 places x_2 it would violate the formula $a(x_2) \wedge (x_1 < x_2)$ satisfied by w. Here is a formlula of quantifier depth 2 that distinguishes these two words: $\exists x_1.\ b(x_1) \wedge (\exists x_2.\ x_1 < x_2) \wedge \forall x_2.(x_2 > x_1) \Rightarrow \neg a(x_2)$. The word w' satisfies this formula with x_1 instantiated as position 7. Further note that $\exists x_2.\ x_1 < x_2$ translates the strategy against player 1 playing position 8 in round 1 and $\forall x_2.(x_2 > x_1) \Rightarrow \neg a(x_2)$ translates the strategy against player 1 playing any other position in round 1. This ability to translate a k round winning strategy to a distinguishing formula of quantfier depth k is not a coincidence.

Lemma 2 *Let w and w' be two V-words such that player 0 has a winning strategy in the k round game. Then, there is a formula ϕ (with free variables in V) with quantifier depth bounded by k that is satisfied by w and not by w'.*

Proof: Let $w = (a_1, V_1)(a_2, V_2)\ldots(a_m, V_m)$ and $w' = (a'_1, V'_1)(a'_2, V'_2)\ldots(a'_n, V'_n)$. The proof proceeds by induction on k. If $k = 0$ then, by definition there is a quantifier-free formula that distinguishes w and w' and this serves as the base case. Suppose the results holds if the number of rounds is less than k.

Now, consider the winning strategy for player 0 that wins the k round game. Suppose this move picks the position i in word w and labels it with variable x. Therefore, any position j in w' as the choice for player 1's move is a loosing move (i.e. player 0 can continue the game so as to win it.) This is equivalent to saying that player 0 has a winning strategy in the $k-1$ round game played on the words $u = (a_1, V_1)\ldots(a_i, V_i \cup \{x\})\ldots(a_m, V_m)$ and $u'_j = (a'_1, V'_1)\ldots(a'_j, V'_j \cup \{x\})\ldots(a'_n, V'_n)$ for each j. Thus, by the induction hypothesis there is a formula ϕ_j, with quantifier depth bounded by $k-1$, such that $u \models \phi_j$ and $u'_j \not\models \phi_j$.

Thus, $w \models \exists x.\ \bigwedge_{1 \leq j \leq n} \phi_j$ (Simply set x to be i). On the other hand $w' \not\models \exists x.\ \bigwedge_{1 \leq j \leq n} \phi_j$. Thus, we have constructed a formula of quantifier depth bounded by k that is satisfied by w and not by w'. ■

Notes: Our presentation has followed the notation used in Straubing [2]. Another book that presents these results is Pippinger [1].

References

[1] Nick Pippenger: *Theories of Computability*, Cambridge University Press, 1997.

[2] Howard Straubing: *Finite Automata, Formal Logic and Circuit Complexity*, Birkhäuser, 1994.

Appendix:

Given a word $w = a_1 a_2 \ldots a_n$ a (V, W)-valuation over w is a function σ that maps V to $\{1, 2, \ldots n\}$ and W to $2^{\{1,2,\ldots,n\}}$. Given a word $w = a_1 a_2 \ldots a_n$ and a (V, W)-valuation σ with $\text{free}_1(\phi) \subseteq V$ and $\text{free}_2(\phi) \subseteq W$, we define when $(a_1 a_2 \ldots a_n, \sigma)$ satisfies a formula ϕ, written $a_1 a_2 \ldots a_n, \sigma \models \phi$, as follows:

$$
\begin{array}{lcll}
a_1 a_2 \ldots a_n, \sigma & \models & a(x) & \text{if } a_{\sigma(x)} = a \\
a_1 a_2 \ldots a_n, \sigma & \models & x < y & \text{if } \sigma(x) < \sigma(y) \\
a_1 a_2 \ldots a_n, \sigma & \models & x \in X & \text{if } \sigma(x) \in \sigma(X) \\
a_1 a_2 \ldots a_n \sigma & \models & \phi \wedge \phi' & \text{if } (a_1 a_2 \ldots a_n, \sigma \models \phi) \text{ and } (a_1 a_2 \ldots a_n, \sigma \models \phi') \\
a_1 a_2 \ldots a_n, \sigma & \models & \neg\phi & \text{if } (a_1 a_2 \ldots a_n, \sigma \not\models \phi) \\
a_1 a_2 \ldots a_n, \sigma & \models & \exists x.\phi & \text{if there is an } i \in \{1, 2, 3, \ldots n\} \text{ such } a_1 a_2 \ldots a_n, \sigma[x : i] \models \phi \\
a_1 a_2 \ldots a_n, \sigma & \models & \exists X.\phi & \text{if there is } S \subseteq \{1, 2, 3, \ldots n\} \text{ such } a_1 a_2 \ldots a_n, \sigma[X : S] \models \phi
\end{array}
$$

where $\sigma[v : y](u) = \sigma(u)$ if $u \neq v$ and $\sigma[v : y](v) = y$.

Given a (V, W)-word $(a_1, F_1, S_1)(a_2, F_2, S_2) \ldots (a_n, F_n, S_n)$ we can construct a word–valuation pair (w, σ) by setting $w = a_1 a_2 \ldots a_n$ and $\sigma(x) = i$ if $x \in F_i$ for any $x \in V$ and $\sigma(X) = \{i \mid X \in S_i\}$ for any $X \in W$. It is easy to check that this is a bijective correspondance between (V, W)-words and word-valuation pairs. We say that a (V, W)-word satisfies a formula ϕ if and only if the corresponding word–valuation pair satisfies the formula ϕ.

Lecture 4: EF games and first order definability

At the end of the last lecture we showed that a winning strategy for player 0 on the k round game played on words w and w' guarantees the existence of a formula with quantifier depth bounded by k that distinguishes w and w'. Do distinguishing formulas lead to winning strategies?

Consider the words $w = abab$ and $w' = baba$. One formula that distinguishes these two words is $\phi = \forall x.\ (a(x) \Rightarrow \exists y.\ (y > x))$. This formula is satisfied by w and not by w'. Here is how we synthesise a winning strategy for player 0 from this formula: Since w' does not satisfy ϕ, there is an i so that with $x = i$ the formula $a(x) \Rightarrow \exists y.\ (y > x)$ is not satisfied. Player 0 picks the word w' and picks this position i (4) giving:

Now, player 1 will place his x_1 against one of the two a's in w. Say, he picks the a at position 3 to give:

Now, w with $x = 3$ satisfies $\exists y.\ (y > x)$ while w' with $x = 4$ does not. Player 0 then picks the witness for this in word w, i.e. position 4 and places his x_2 there.

And now every move that player 1 makes will be losing.

The strategy construction using a distinguishing formula proceeds in the following way:

1. If the distinguishing formula is a quantifier free formula then clearly player 0 wins even the 0 round game.

2. If the distinguishing formula is of the form $\neg\phi$ then ϕ is also a distinguishing formula and we use that instead.

3. If the distinguishing formula is of the form $\phi_1 \wedge \phi_2$ then at least one of ϕ_1 or ϕ_2 is also a distinguishing formula and we use that instead.

4. If the distinguishing formula is $\exists x.\phi(x)$ then in one of the words w or w' there is a position i such that $\phi(x)$ is true when x is assigned the position i. Player 0 picks this word and the position i as his move. In the other word no matter which position we assign to x, $\phi(x)$ is not satisfied. Thus, no matter how player 1 responds to this move, resulting pair of words will be distinguished by $\phi(x)$, a formula of lower quantifier depth.

This is a winning strategy as player 0 has arranged things so that the words formed after each move are distinguished by formulas of lower quantifier depth. Thus after k rounds, where k is the quantifier depth of the formula distinguishing the original words, he is left with a quantifier free formula that distinguishes the two words. This gives us the following theorem.

Theorem 1 *Two V-words are distinguished by a formula of quantifier depth k if and only if player 0 has a winning strategy in the k round EF game associated with these words. In particular, if two words over Σ are distinguished by quantifier depth k sentences if and only if player 0 has a winning strategy in the k round EF game associated with these words.*

Proof: The direction from formulas to winning strategies is described above. The other direction was proved in the last class. ■

0.1 Evenness is not first-order definable

We now show that the words a^m and a^{m+1} are not distinguishable by quantifier depth k sentences if $m \geq 2^k$. We do this by showing that player 1 has a winning strategy in the k round game defined by these words. The proof proceeds by induction. When $k = 0$ we have the words a^i and a^{i+1}, $i \geq 1$ are clearly indistinguishable by atomic formulas (over the empty set of variables!).

Let us suppose that player 1 has a winning strategy in the r round game over (a^m, a^{m+1}), for all $r < k$ and $m \geq 2^r$. Consider the k round game over the words (a^m, a^{m+1}) with $m \geq 2^k$.

The move by player 1 would divide one of the words (the word that he picks) into three parts so that it looks like $a^s\ a\ a^t$ (where $s + 1 + t = m$ or $s + 1 + t = m + 1$, depending on which word was picked). But note that either s or t is at least 2^{k-1}.

Suppose $t \geq s$ then player 1 picks position $s + 1$ in the other word and plays that as his response. This ensures that w.r.t. the first variable placed on both words, the words to the left are identical and the words to the right are both of length $\geq 2^k$. After the first move the words look like $a^s(a, x)a^t$ and $a^s(a, x)a^{t'}$ where $|t - t'| = 1$, $t, t' \geq 2^{k-1}$. Thus, by the induction hypothesis, player 1 has a winning strategy on the $k - 1$ round game on $(a^t, a^{t'})$. From now on, whenever player 0 picks a position among the initial $s + 1$ positions in one word, player 1 simply duplicates the move in the other word. If player 0 picks a position in a^t (or $a^{t'}$) then player 1 responds using his strategy on the game $(a^t, a^{t'})$. It is not difficult to check that this is a winning strategy for player 1.

The construction of the strategy when $s \geq t$ proceeds similarly. Thus we have established that player 1 has a winning strategy in the k round EF game played on the words a^{2^k} and a^{2^k+1}.

We have just shown that player 1 has a winning strategy in this game and thus player 0 cannot have a winning strategy in this game. Thus, there is no quantifier depth k formula that can distinguish the words a^{2^k} and $a^{2^{k-1}}$.

Theorem 2 *The language $\{a^{2n} | n \geq 0\}$ is not definable in the first-order logic of words.*

Proof: Suppose ϕ is a formula that defines this language. Let the quantifier depth of ϕ be k. Then, either a^{2^k} and a^{2^k+1} are both in $L(\phi)$ or neither is in $L(\phi)$. This contradicts the definition of ϕ. ■

0.2 EF games for MSO

We can extend EF games to allow "second order" moves: In such a move, player 0 picks a subset of positions in one of the two words and labels them all by a second order variable X. In reponse player 1 must pick a subset of positions in the other word and label them all with X. As usual, after the k rounds are played, the winner is determined by whether the two resulting models satisfy the same set of quantifier free formulas or not (or equivalently by atomic formulas or not).

It is quite easy to establish that two words are distinguishable by MSO formulas of quantifier depth k if and only if player 0 has a winning strategy in the k round second-order EF game (where both first order and second order moves are allowed) played over these words.

An Application: We shall use this characterization to give an alternative proof of the fact that MSO formulas define regular languages: Let us write $w \sim_k w'$ to denote that player 1 has the winning strategy in the k round game on (w, w') (Equivalently, w and w' are indistinguishable via quantifier depth k formulas). Clealy $\sim_k$ is a equivalence relation on Σ^*. Moreover, a simple extension of theorem 1 from the previous lecture, shows that it is also of finite index.

We show that this relation is right invariant. Consider any pair of $w.z$ and $w'.z$. The winning strategy for player 1 is the following: when player 0 picks a position in z duplicate the move in the other word. Whenver player 0 picks positions in w or w' respond using the winning strategy on (w, w'). It is not difficult to check this is a winning strategy for player 1. Thus $w.z \sim_k w'.z$.

Finally observe that for any ϕ with quantifier depth k, $L(\phi) = \bigcup_{x \models \phi} [x]_{\sim_k}$. Hence, $\sim_k$ is a right congruence of finite index that saturates $L(\phi)$. Thus $L(\phi)$ is a regular language.

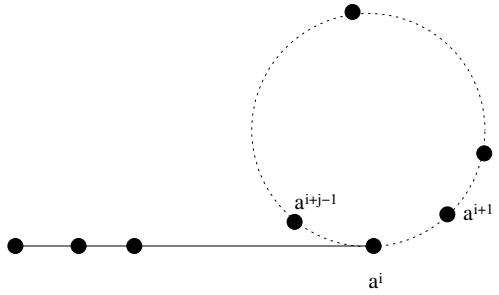
1 Aperiodic Monoids, Star-free sets and FO definability

We say that a monoid M is group-free if it does not contain a nontrivial (i.e. other than the 1 element group) subgroup (Note that we don't insist that the identity of the monoid appear in the group).

An element i in a monoid M is said to be an *idempotent* if $i.i = i$.

Lemma 3 *(Frobenius) Any finite cyclic semigroup contains idempotents.*

Proof: Consider a sequence $a, a^2, a^3, \ldots$ (where $a^i = a.a.a.\ldots.a(i \text{ times})$). There is some least i and least $j > i$ such that $a^i = a^{i+j} = a^i a^j$. Here, j called the *period* of the element a. Thus the sequence looks like the following lollipop:



Thus, $a^i = a^i a^j = a^i a^j a^j = \ldots = a^i (a^j)^n \ldots$. Our aim is to find a idempotent, i.e. an element of the form a^{i+k} such that $a^{i+k}.a^{i+k} = a^{i+k}$. This would clearly be true if $i + k$ is divisible by j (since $a^{i+k+i+k} = a^{i+mj+k} = a^{i+k}$). Thus, any $i + k$ with $i + k$ divisible by j is an idempotent. ■

This allows us to characterize group-free monoids as follows:

Lemma 4 *A monoid M is group-free if and only if there is an N such that for $a \in M$, $a^N = a^{N+1}$.*

Proof: Suppose, there is such a N and suppose G is a group contained in M. Let $a \in G$ and let a^{-1} be its inverse in G. Therefore $a^N = a^{N+1}$ implies $a^N.(a^{-1})^N = a.a^N.i(a^{-1})^N$. Thus $id_G = a$. Thus the group is trivial.

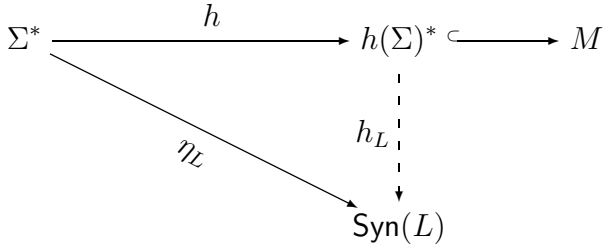
Suppose M is group-free. Pick any $a \in M$. By Lemma 3, $a^i = a^{i+j}$ for some j. If $j > 1$, then $\{a^i, a^{i+1}, \ldots, a^{i+j-1}\}$ forms a nontrivial subgroup of M. Thus $j = 1$. Thus $a^i = a^{i+1}$. Since M has only finitely many elements it follows that we can pick a N such that $a^N = a^{N+1}$ for all $a \in M$. ■

A monoid M is said to be *aperiodic* if there is a N such that $a^N = a^{N+1}$ for each $a \in M$. The above lemma shows that aperiodic monoids and group-free monoids are the same.

We say that a regular language is aperiodic if it is recognized by some aperiodic monoid. We next show that a regular language L is recognized by an aperiodic monoid if and only if its syntactic monoid is aperiodic.

Theorem 5 *Let L be recognized by a morphism $h : \Sigma^* \to M$ and suppose M is aperiodic. Then $\mathsf{Syn}(L)$ is aperiodic.*

Proof: Let N be such that $x^N = x^{N+1}$ for all $x \in M$. By Theorem 3 of lecture 1, η_L factors via h as follows:



η_L is a surjective map and thus h_L is also surjective. Thus if $y \in \mathsf{Syn}(L)$, $y = h_L(x)$. But $x^N = x^{N+1}$. Thus $h_L(x^N) = h_L(x^{N+1})$, giving $h_L(x)^N = h_L(x)^{N+1}$. ■

This shows that checking whether a language is aperiodic is decidable. We just need to check whether its syntactic monoid is aperiodic.

Exercise: Show that the syntactic monoid of the language $\{a^{2n} \mid n \geq 0\}$ is not aperiodic.

One of the corner stones of the study of regular languages is the result of M.P.Schutzenberger showing that languages recognized by aperiodic monoids are exactly the class of languages that are described by regular expressions using the operators + (union) . (concatenation) and - (complementation). McNaughton showed that the the class of languages that can be expressed using the first-order logic of words also coincides with this class. We shall establish these two results in this and the following lecture.

2 From Star-free languages to Aperiodic Monoids

We shall show that every regular language that can be described by some star-free regular expression is also recognized by an aperiodic monoid. This can be seen as follows:

1. It is trivial to specify aperiodic monoids recognizing the languages $\{a\}$, $\{\epsilon\}$ and $\emptyset$.
2. If L is accepted via the (aperiodic) monoid M, then so is $\overline{L}$.
3. If L_1 and L_2 are recognised using the morphisms h_1 and h_2 to the monoid M_1 and M_2 respectively so that $L_1 = h^{-1}(X_1)$ and $L_2 = h^{-1}(X_2)$ then $L_1 \cup L_2$ is recognized by the monoid $M_1 \times M_2$ via the morphism h with $h(a) = (h_1(a), h_2(a))$ as the pre-image of the set $X_1 \times M_2 \cup M_1 \times X_2$.
4. Finally, suppose L_1 and L_2 are aperiodic so that $x^N = x^{N+1}$ for each $x \in \mathsf{Syn}(L_1) \cup \mathsf{Syn}(L_2)$. Then, we claim that for any word $w \in \Sigma^*$, $uw^{2N}v \in L_1.L_2$ if and only if $uw^{2N+1}v \in L_1.L_2$ (Exercise). Thus, $[w^{2N}]_{L_1.L_2} = [w^{2N+1}]_{L_1.L_2}$. Therefore, $\mathsf{Syn}(L_1.L_2)$ is aperiodic.

Exercise: Show how to directly construct a monoid recognizing $L_1.L_2$ from monoids accepting L_1 and L_2.

Exercise: Show that every language that can be described via star-free regular expressions can also be expressed in the first-order logic of words.

Notes: Most of the results mentioned here are from [2] and [1].

References

[1] Nick Pippenger: *Theories of Computability*, Cambridge University Press, 1997.

[2] Howard Straubing: *Finite Automata, Formal Logic and Circuit Complexity*, Birkhäuser, 1994.

Lecture 5: Schutzenberger's Theorem

In this lecture we shall prove the theorem of Schutzenberger that relates languages recognized by aperiodic monoids with regular languages that can be described via star-free regular expressions. The proof described below is an adaptation of the proof given by Nick Pippenger in [1] (and Pippenger indicates that his proof follows the original presentation by Schutzenberger in [2]).

Theorem 1 *(Schutzenberger) Every aperiodic language can desribed via star-free expressions.*

The proof of this theorem proceeds by induction on the size of the monoid recognizing the language L. To start with we set out a few lemmas that help us to carry out the inductive argument. In this lecture every monoid we consider will be aperiodic.

Lemma 2 *For any x in an aperiod monoid M, if $x = pxq$ then $x = px$ and $x = qx$.*

Proof: Clearly $x = p^i x q^i$ for each i. But, since M is aperiodic, there is an N such that $p^N = p^{N+1}$. Thus, $x = p^N x q^N = p^{N+1} x q^N = px$. Similarly $x = xq$. ■

One easy consequence of this lemma is that if $e = ab$ then $e = a$ and $e = b$ for the identity e of an aperiodic monoid M.

We say that a subset I of M is an *ideal* if $IM \subseteq I$ and $MI \subseteq I$. Thus, an ideal is a subset that is closed w.r.t. multiplication (on both sides) by the elements of the monoid. Ideals are interesting subsets as one can define quotient monoids via ideals.

Definition 3 *Let M be an ideal and let I be an ideal of M. Then, there is a natural monoid M/I whose elements are $M - I \cup \{i\}$ and whose multiplication operation . is defined as follows:*

- $x.i = i.x = i.i = i$
- $x.y = i$, *if* $x.y \in I$
- $x.y$ *is the same as* $x.y$ *in* M *otherwise.*

It is easy to check M/I is a monoid. There is the obvious morphism η_I from M to M/I which is identity on the elements of $M - I$ and maps every element of I to i. Note that if a language L is recognized by a morphism h as $h^{-1}(I)$ in M then the same is recognised by $\eta \circ h$ to M/I as the pre-image of $\{i\}$. We shall simply write h to denote the composed map $\eta \circ h$ to M/I.

Thus, if I is an ideal of size more than 2 then any language recognized as the preimage of I can be recognised via a smaller monoid (M/I). More generally,

Lemma 4 *Let M be an finite aperiodic monoid, I be an ideal and let either $I \subseteq X$ or $X \cap I = \emptyset$. Then, any language L recognized as the preimage of X is recognized via the monoid M/I. In particular, if I has atleast 2 elements then L is recognized by a smaller aperiodic monoid.*

Definition 5 *With each element x of a monoid M we can associate a interesting ideal $F(x)$, called the forbidding ideal of x.*

$$F(x) = \{y \mid \forall p, q. pyq \neq x\}$$

It consists of all the elements that cannot "divide" x or cannot generate x via multiplication.

It is easy to check that $F(x)$ is an ideal for any $x \in M$.

Lemma 6 *Let h be a morphism from Σ^* to M. Then, $h^{-1}(x) = (\eta_{F(x)} \circ h)^{-1}(x)$. Thus, if $F(x)$ has at least 2 elements then the language recognized as $h^{-1}(x)$ can be recoginized using a smaller monoid.*

This follows from the fact that $x \notin F(x)$ and $F(x)$ is an ideal.

We are now in a position to describe the main ideas behind the proof. The proof, understandably, proceeds by induction on the size of the monoid M. If M is the trivial monoid, then the only languages recognised via M are $\emptyset$ and Σ^* and clearly both are star-free languages.

For the induction step, consider any language L recognized via some monoid M. Firstly, for any $X = \{x_1, x_2, \ldots, x_k\}$, $h^{-1}(X)$ is the union of the sets $h^{-1}(x_1)$, $h^{-1}(x_2)$, ..., $h^{-1}(x_k)$. Thus, it suffices to show that $h^{-1}(x)$ can be expressed as a star-free expression involving languages definable using aperiodic monoids smaller than M.

If $F(x)$ has at least two elements, this would just be an application of Lemma 6. Otherwise, we need to do a lot of hard work. The idea is to show that we can find a collection Y of elements in M such that $h^{-1}(x)$ can be described as a star-free expression involving $h^{-1}(y)$, $y \in Y$, and further, for each $y \in Y$, $F(y)$ strictly contains $F(x)$. Once we do this, notice that we can always focus our attention on $h^{-1}(x)$ for only those elements with $|F(x)| > 1$ and complete our proof using Lemma 6.

Observe that $F(e) = M \setminus \{e\}$ for an aperiodic monoid (this follows from Lemma 2). Thus, $h^{-1}(e)$ poses any problems only in the case that M has fewer than 2 elements and we leave that as an exercise and assume henceforth that we are only interested in $h^{-1}(x)$ for $x \neq e$.

As a first step in that direction, we show that languages recognised by any ideal of M, even those of size 1, can be reduced to star-free expressions involving languages definable via smaller monoids.

Lemma 7 *If $L = h^{-1}(I)$ for some ideal I in M then L can be expressed using star-free expressions involving languages which are recognized by smaller aperiodic monoids.*

Proof: If the ideal $I = \emptyset$ then $L = \emptyset$ and it can be described by the star-free expression $\emptyset$.

Otherwise $|I| \geq 1$. Let $A = \{a \mid h(a) \in I\}$. Clearly, for each $a \in A$, the expression $E_a = \overline{\emptyset}.a.\overline{\emptyset}$ defines a language L_a contained in L (since I is an ideal). Thus, we could focus our attention on writing star-free expressions to cover words in $L \setminus \bigcup_{a \in A} L_a$.

Pick any word w in L. Consider a minimal substring of u of w that is in L.

If $u = \epsilon$ then the identity of M is in I which means that $M = I$ and thus $L = \Sigma^*$. If $u = a$ for some $a \in \Sigma$ then $w \in L_a$ and we need to do nothing in this case.

Thus we only need to consider the case when $u = avb$ for some v. Let $h(v) = y$. Note that, by the minimality of u, none of y, $h(a)y$ or $yh(b)$ can be in I.

Note that, since I is an ideal, $h(w_1).a.y.b.h(w_2) \in I$ for each $w_1, w_2 \in \Sigma^*$. Thus, $\overline{\emptyset}.a.h^{-1}(y).b.\overline{\emptyset} \subseteq L$. Note that this language contains the word w. If we show that $h^{-1}(y)$ may be described via a smaller monoid, this would take us one step closer to a star-free expression for L, since we have now managed to cover the word w.

Next we show that $F(y)$ has at least two elements, thus establishing that $h^{-1}(y)$ can be accepted via a smaller monoid $(M/F(y))$. Since $y \notin I$ and I is an ideal, $I \subseteq F(y)$ and since I is nonempty, there is at least one element in $F(y) \cap I$. We now show that there is at least one other element in $F(y)$.

Consider $h(a)y$. If $h(a)y \notin F(y)$ then there must be p, q such that $y = ph(a)yq$. Thus $y = ph(a)y$ and multiplying both sides by $h(b)$ we get $yh(b) = ph(a)yh(b)$ which is in I since $h(a)yh(b) \in I$. But this contradicts the minimality of u (since then $vb \in L$). Thus, $h(a)y \in F(y) \setminus I$. Thus $F(y)$ has at least two elements.

Finally, even though there are infinitely many w's outside of $\bigcup_{a \in A} L_a$ in L, the monoid M is finite and so is the alphabet Σ and thus we only have finitely many choices for triples of the form (a, y, b). Thus, we can describle all of $L \setminus \bigcup_{a \in A} L_a$ as a finite union of languages of the form $\overline{\emptyset}.a.h^{-1}(y).b.\overline{\emptyset}$, with $|F(y)| > 1$. This completes the proof of this lemma.

■

The other key idea behind Schutzenberger's proof is the following technical lemma :

Lemma 8 $x = (xM \cap Mx) \setminus F(x)$

Proof: Let $y \in (xM \cap Mx) \setminus F(x)$. Thus $y = px$ and $y = xq$ and $x = rys$. Then, by Lemma 2, $y = xq = rysq$. Thus $y = ry$. Similarly, $y = px = prys$ and thus $y = ys$. Thus $y = rys = x$. ■

We say "language defined by x" to mean $h^{-1}(x)$. Next we show that $h^{-1}(x)$ can be expressed as a star-free expression involving languages defined by other elements for which the forbidding set is larger than $F(x)$.

Lemma 9 *Let $x \in M$, then there is a subset $Y \subseteq M$ such that, $\forall y \in Y$. $F(y)$ strictly contains $F(x)$ and $h^{-1}(x)$ can be expressed as a star-free expression involving $h^{-1}(y)$, $y \in Y$ and other languages definable via smaller aperiodic monoids.*

Proof: We know that $x = (xM \cap Mx) \setminus F(x)$. Note that $h^{-1}((xM \cap Mx) \setminus F(x)) = h^{-1}(xM \setminus F(x)) \cap h^{-1}(Mx \setminus F(x))$ and $h^{-1}(xM \setminus F(x)) = h^{-1}(xM) \setminus h^{-1}(F(x))$.

$F(x)$ is an ideal and so, using Lemma 7, $h^{-1}(F(x))$ can be expressed via smaller monoids. We show that for each $w \in h^{-1}(xM \setminus F(x))$ one can find a letter a and an element y such that $h^{-1}(y).a.\overline{\emptyset} \subseteq h^{-1}(xM \setminus F(x))$ where $F(y)$ has more elements than $F(x)$. This combined with the fact that $h^{-1}(F(x))$ is recognized by smaller monoids (by Lemma 7) gives the inductive argument.

Let $w \in h^{-1}(xM \setminus F(x))$. Let u be the shortest prefix of w such that $h(u) \in (xM \setminus F(x))$. If $u = \epsilon$, then then $e = xd$ for some $d \in M$ where e is the identity of M. This contradicts the aperiodicity of M.

Thus $u = va$ for some v such that $h(v) = y \notin (xM \setminus F(x))$. We claim that $h^{-1}(y).a.\overline{\emptyset} \subseteq h^{-1}(xM \setminus F(x)) \cup h^{-1}(F(x))$. In proof, $h(avw') = h(va)h(w') = xm.d = xm'$. Thus it is in xM and thus either in $(xM \setminus F(x))$ or in $F(x)$. Thus every word in $h^{-1}(y).a.\overline{\emptyset}$ is in $h^{-1}(xM \setminus F(x)) \cup h^{-1}(F(x))$.

Next we show that $F(y)$ has more elements than $F(x)$. First of all $y \notin F(x)$. Otherwise, $yh(a)$ will also be in $F(x)$ leading to a contradiction. Thus $F(x) \subseteq F(y)$. We now show that $yh(a)$ is also in $F(y)$. Suppose $yh(a)$ is not in $F(y)$. Then $y = pyh(a)q$. This means that $y = py$ and $y = yh(a)q$. Thus $y = xdq$ (since $yh(a) \in xM$) and thus $y \in xM$. But we already showed that $y \notin F(x)$ and thus $y \in xM \setminus F(x)$. This contradicts the minimality of u. Thus it must be the case that $yh(a) \in F(y)$ and thus $F(y)$ has at least one more element than $F(x)$.

Thus, we have picked an arbitrary element w of $h^{-1}(xM \setminus F(x))$ and shown that there is a star-free regular expression involving $h^{-1}(y)$ for some y with $F(y)$ strictly containing $F(x)$, that describes a language containing w and which is contained in $h^{-1}(xM \setminus F(x)) \cup h^{-1}(F(x))$. Since M and Σ are finite sets, it follows that we can write a star-free regular expression involving some elements $y_1, \ldots y_k$ of M, where $F(y_i)$ strictly contains $F(x)$ for each i, that describes a language containing $h^{-1}(xM \setminus F(x))$ and which is contained in $h^{-1}(xM \setminus F(x)) \cup h^{-1}(F(x))$. The proof follows from Lemma 7 and the fact that star-free expressions may use the boolean operations.

A similar argument works for $h^{-1}(Mx \setminus F(x))$ and this completes the proof of this lemma. ■

Finally, we can establish Schutzenberger's theorem.

Proof: As indicated earlier if M is the trivial monoid, it can only describe $\emptyset$ and Σ^* and both of these are star-free languages. For the induction step, since star-free languages are closed under union, it suffices to consider $h^{-1}(x)$ for some $x \in M$.

Applying Lemma 9 twice, we may conclude that there is a set $Y \subseteq M$ such that $F(y)$ contains two more elements than $F(x)$ for each $y \in Y$ and further $h^{-1}(x)$ can be expressed as a star-free expression involving $h^{-1}(y)$, $y \in Y$ and other languages definable using small aperiodic monoids. But then $h^{-1}(y)$ is recognizable via a smaller monoid $M/F(y)$ for each $y \in Y$. Thus $h^{-1}(x)$ can be described via a star-free expression that only involves languages definable via smaller aperiodic monoids and we may now apply the induction hypothesis to conclude that $h^{-1}(x)$ is an aperiodic language. ■

References

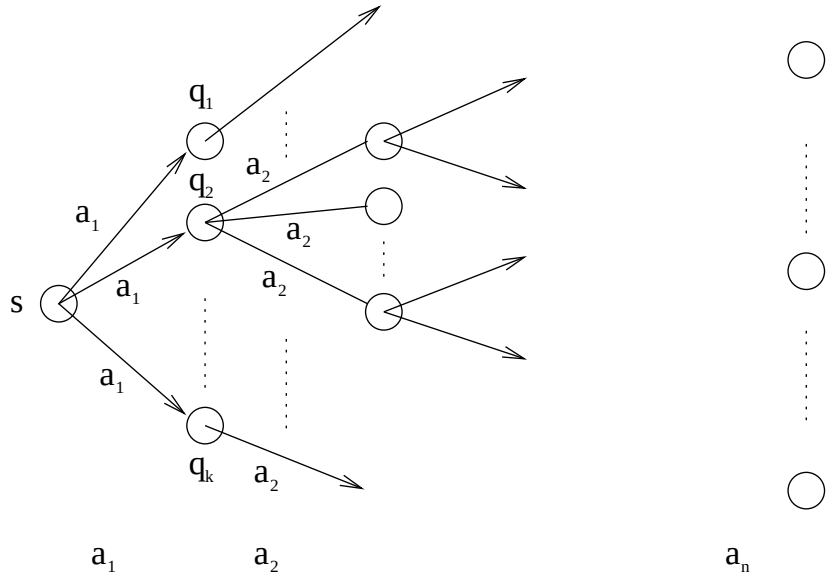
[1] Nick Pippenger: *Theories of Computability*, Cambridge University Press, 1997.

[2] M.P.Schutzenberger: "On Finite Monoids Having only Trivial Subgroups", *Information and Control* **8** (1965) 190-194.

Lecture 6: Alternating Automata

In this lecture we shall examine a new technique to prove that nondeterministic finite automata can be complemented. We have already seen two techniques, one via Myhill-Nerode classes and another via the powerset construction. In the powerset construction we show that a subclass of NFAs, namely deterministic automata, are closed under complmentation and equivalent in expressive power to NFAs. In this lecture we shall go the other way. We shall generalize NFAs to Alternating Finite Automata and show that this larger class, for which closure under complementation is easy to establish, is equal in expressive power to NFAs.

The motivation comes from the following idea. In order to check that a given NFA does not accept a word w, it suffices to do the following: Suppose $w = a_1 w'$ and $\delta(s, a_1) = \{q_1, q_2, \ldots, q_k\}$. Then it suffices to start k copies of the automaton, one from q_1, one from q_2 and so on and verify that none of these has an accepting run on w'. Of course, to verify that q_i has no accepting run on $w' = a_2 w''$ we shall do the same thing, i.e., start one copy for each state in $\delta(q_i, a_2)$ and verify that all these copies do not accept w'' and so on. In this process, we get a tree of depth $|w|$ where the edges at level i are all labelled by a_i, and w is not accepted iff every leaf node in this tree is labelled by a state from $Q \setminus F$.

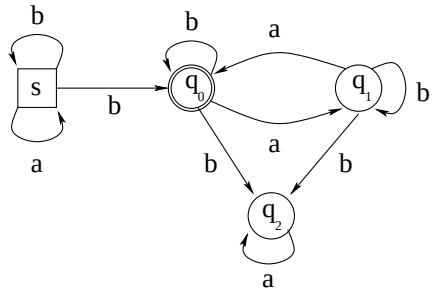


Here is a different way of looking at this idea: Usually, to check that the automaton accepts a word aw starting at a state q, we inductively proceed by evaluating $\mathsf{Acc}(q, w)$ where $\mathsf{Acc}(p, \epsilon)$ is true whenever $p \in F$ and $\mathsf{Acc}(q, aw) = \bigvee_{p \in \delta(q,a)} \mathsf{Acc}(p, w)$. This corresponds to an interpretation of the transition function δ as a logical or. On the other hand, the above construction interprets $\mathsf{Acc}(q, aw)$ as $\bigwedge_{p \in \delta(q,a)} \mathsf{Acc}(p, w)$ with $\mathsf{Acc}(p, \epsilon)$ being true whenever if $p \in Q \setminus F$. It interprets the transition function as a logical and.

Once we permit the interpretation of the transition function as a or in one automaton and as a and in another, we can go one step further and permit both of these choices coexist in the same automaton. This leads us to alternating automata.

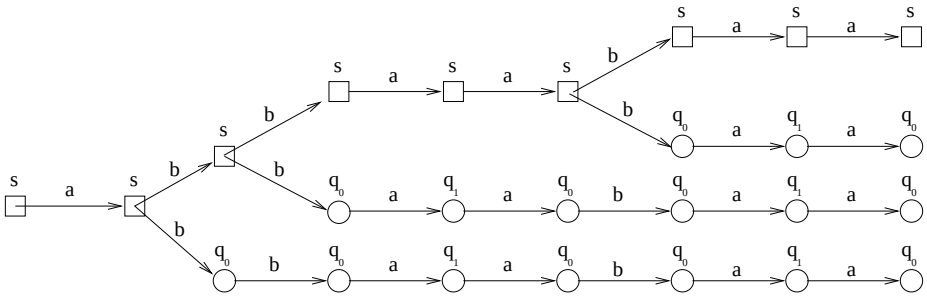
The set of states of an alternating automaton is divided into two sets $Q_\exists$ and $Q_\forall$. (The idea being that the transitions out of states in $Q_\exists$ are to be interpreted as or transitions and the transitions out of states in $Q_\forall$ are to be interpreted as and transitions.) Syntactically, this is the only difference between an NFA and an AFA. So, an AFA is of the form $(Q_\exists, Q_\forall, \Sigma, \delta, s, F)$ and we shall write Q for $Q_\exists \cup Q_\forall$.

Here is a an alternating automaton $(\{q_0, q_1, q_2\}, \{s\}, \{a, b\}, s, \delta, \{s, q_0\})$ where we have drawn the $\forall$ states as squares and the $\exists$ states as circles.

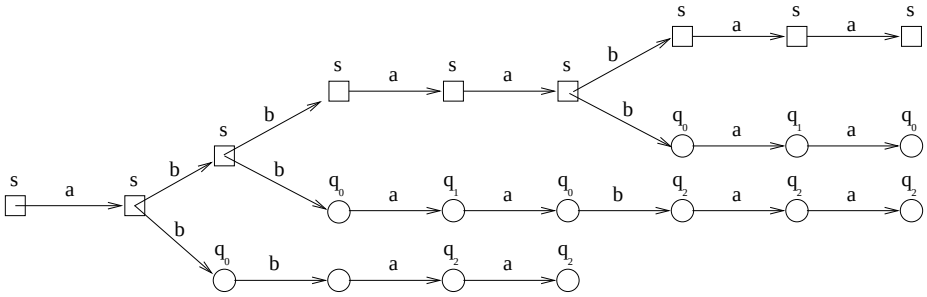


How does this automaton work? From the state s the automaton reads the entire input (since on both a as well as b there is a self-loop back to s), but since it is a $\forall$ state, on reading each b, it starts of an additional copy at state q_0 to read the rest of the input. It is quite easy to check that from state q_0 a word is accepted precisely when it has even number of a's. Thus, the language accepted by this automaton consists of all those words in which the number as to the right of each b is even.

Here is an accepting run of this automaton on the word *abbaabaa*. Notice, how the run branches in the state s on input b.



An alternating automaton can have multiple runs on a given input, because of the presence of nondeterminism. Here is another (partial) run of the above automaton on *abbaabaa*



The run is partial as along one of the paths the entire input is not read (since there is not transition on b at the state q_2.) Thus, this is not an accepting run. (Moreover, there is another path along which the entire input is read but the resulting state is not accepting.)

Formally a run of an alternating automaton $(Q_\exists, Q_\forall, \Sigma, \delta, s, F)$ on an input $a_1 a_2 \dots a_n$ is a Q labelled rooted tree where

1. The root is labelled by s.
2. If an interior node, at some level i, is labelled by some $q \in Q_\exists$ then it has single child which is labelled by a p where $p \in \delta(q, a_i)$.
3. If an interior node, at some level i, is labelled by some $q \in Q_\forall$ and $\delta(q, a) = \{q_1, \dots q_k\}$ then it has k children, labelled q_1, q_2 $\dots$ q_k.

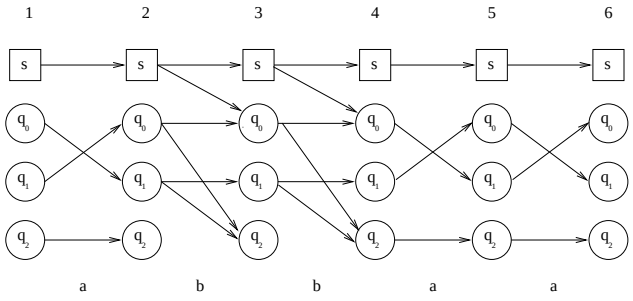
A run is accepting if all the leaf nodes are labelled by states in F.

In the next section we outline a notion of a game played on finite graphs (as a matter of fact finite DAGs suffices for this lecture) and relate it to alternating automata.

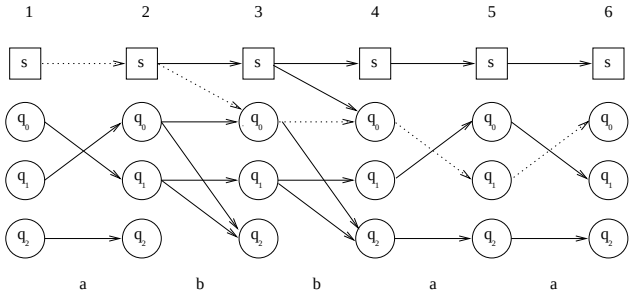
1 Games on Finite DAGs

With every alternating automaton $A = (Q_\exists, Q_\forall, \Sigma, \delta, s, F)$ and word $w = a_1 a_2 \dots a_n$, we can associate a directed (acyclic) graph with vertex set $Q \times \{1, 2, \dots, n+1\}$. Edges go from level i to level $i+1$. If $\delta(q, a_i) = \{q_1, \dots, q_k\}$ then there are edges from (q, i) to $(q_j, i+1)$ for $1 \leq j \leq k$. We shall refer to this game as $\mathcal{G}(A, w)$.

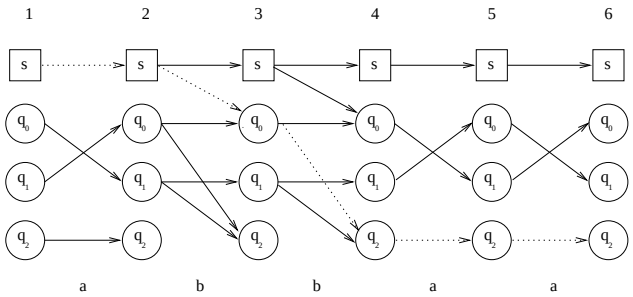
Here is the graph corresponding to the automaton described in the previous section and the word *abbaa*.



Given such a graph we define a game as follows: There are two players, *automaton* and *pathfinder*. The set of nodes is divided among the two players. The automaton owns all the nodes labelled with states from $Q_\exists$ and the pathfinder owns the nodes labelled by states from $Q_\forall$. A token is placed on the node $(s, 1)$ (i.e. the copy of the start state in the first level). In each round the player who owns the node with the token makes the move. He chooses one of the neighbours (via outgoing edges) and moves the token there. The play ends when the token reaches a vertex with no outgoing edges. The aim of the automaton is to ensure that the token reaches a node labelled by an accepting state in level $n + 1$. We say that the automaton wins the play if this happens (i.e. the token lies on a level $n + 1$ node labelled with a final state.) Otherwise, the pathfinder wins the play. For example, the following play (marked by dotted edges in the figure below) is winning for the automaton:



On the other hand, the following play is winning for the pathfinder.



Observe that we do NOT require that the players alternate in making moves. So much so that, one player may NOT make any moves at all during a play. For instance, if the pathfinder were to always move from s to s in the above game the automaton would never get an opportunity to make a move. However this would be very clever on part of the pathfinder since he would lose such a play!

A *strategy* f for a player (automaton/pathfinder) is a function that "examines" the entire play so far and decides what move to make. (Formally a strategy for the automaton is a function that whose domain is the set of sequences of the form $q_1 \longrightarrow q_2 \longrightarrow q_3 \ldots q_i$ with $q_i \in Q_\exists$ and for such an input the function returns a p where $p \in \delta(q_i, a_i)$. Similarly, a strategy for the pathfinder is a function whose domain is the set of sequences of the form $q_1 \longrightarrow q_2 \longrightarrow \ldots q_i$ with $q_i \in Q_\forall$ and it must return a $p \in \delta(q_i, a_i)$.) We say that a play is *consistent with strategy* f for the automaton (pathfinder) if all the moves made by the automaton (pathfinder) along this play are as suggested by the strategy f. We say that f is

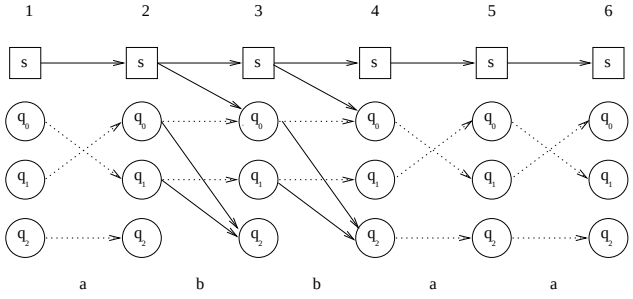
a winning strategy for the automaton (pathfinder) if every play consistent with f is winning for the automaton (pathfinder).

Claim 1: If the automaton has an accepting run on the given input word, then he has a winning strategy in the game associated with that word.

Proof: The idea is to do the following: As the play proceeds we trace a corresponding path through the accepting run (which is a tree) and use that in defining the next move. The path starts at the root (which is labelled by the start state) and the starting configuration in the game consists of the token the token being placed at the start state in level 1.

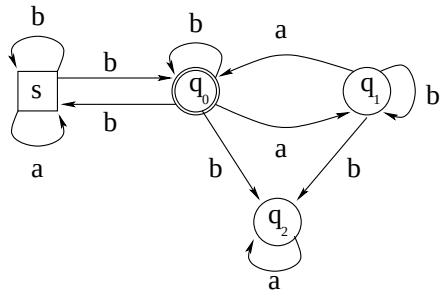
Inductively, assume that after i moves in the game the token is at a vertex labelled q in level $i+1$ and that corresponding path that we have traced in the accepting run ends a node at level $i+1$ labelled by the same state q. If q is a $Q_\exists$ state then there is unique child in the rune and suppose this is labelled p then our strategy recommends move $(q, i+1) \longrightarrow (p, i+2)$. If $q \in Q_\forall$, then it is the turn of the pathfinder to make a move. Suppose he moves the token to $(p, i+2)$ then we trace the edge from q at level $i+1$ to p at level $i+2$ in the accepting run (which must exist, since every state in $\delta(q, a_{i+1})$ appears as the label of a child of $(q, i+1)$.) Thus, following this strategy, after n rounds of moves the token will a node at level $n+1$ in the graph whose label is identical to that of the state (at level $n+1$) reached by the path traced in the accepting run. But every leaf node in the accepting run is labelled by a a state from F. Thus, this play ends in a node labelled by a state in F and is winning for the automaton. Since this argument works for all plays it follows that the strategy defined above is a winning strategy for the automaton. ■

A strategy f is said to be *positional* or *memoryless* if its value on $q_1 \longrightarrow q_2 \longrightarrow \ldots q_i$ depends only only q_i. That is, $f(q_1 \longrightarrow q_2 \longrightarrow \ldots q_{i-1} \longrightarrow q_i) = f(q'_1 \longrightarrow q'_2 \longrightarrow \ldots q'_{i-1} \longrightarrow q_i$ for any two runs ending at q_i. Thus, we may think of a positional winning strategy for the automaton (pathfinder) to be a function that sends each node that belongs to the automaton (pathfinder) to one of its neighbours. The following figure indicates a positional winning strategy (where the edges chosen by the strategy are marked by dots) for the automaton:

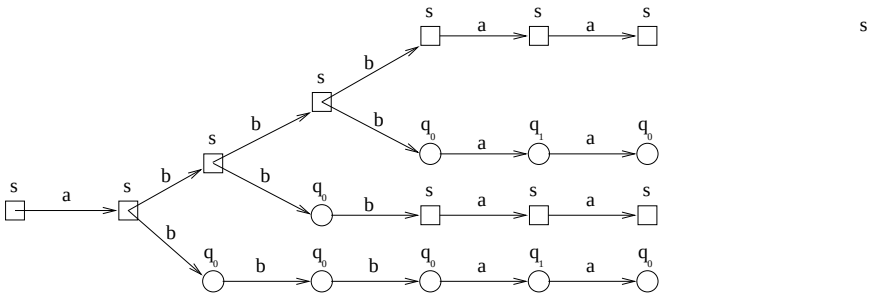


In the above game, an example of a non-positional winning strategy for the automaton is one that moves to q_0 on $s \longrightarrow s \longrightarrow q_0 \longrightarrow q_0$ and to q_2 on $s \longrightarrow s \longrightarrow s \longrightarrow q_0$. However such a strategy is not winning because, the pathfinder would play $s \longrightarrow s$ in the first round and $s \longrightarrow s$ in the second and $s \longrightarrow q_0$ in the third and now the automaton would use f to move to q_2 from where it cannot win the game.

Exercise: Consider the following automaton. Describe a non-positional and a positional winning strategy for the automaton in the game associated with the word *abbbaa*. Are there any non-positional strategies for the pathfinder in this game?



Will the strategy constructed in the proof of Claim 1 a positional strategy? Not always. It is possible that the accepting run may have the same state q appearing twice at the same level i and the state labelling the children of these two occurances may not be the same. If the paths from the root to these two occurances are $q_0 \longrightarrow q_1 \longrightarrow \ldots q_i$ and $q_0 \longrightarrow q_1' \xrightarrow{q'}_2 \longrightarrow \ldots \longrightarrow = q_i$ respectively, then the strategy induced by this run would play different moves on $q_0 \longrightarrow q_1 \longrightarrow q_2 \ldots q_i$ and $q_0 \longrightarrow q_1' \longrightarrow q_2' \ldots \longrightarrow q_i$ and thus it is not positional at q_i. As an illustration, consider the following accepting run of the automaton A_1 defined in the above exercise (on input *abbbaa*).



The strategy defined using this run yields q_0 on $s \longrightarrow s \longrightarrow s \longrightarrow q_0$ while it yields s on $s \longrightarrow s \longrightarrow q_0 \longrightarrow q_0$. Let us call an accepting run to be positional for if all occurances of a state $q \in Q_\exists$ in any fixed level i, the label of the unique child of each these occurances is the same.

It is quite easy to check that if we start with positional accepting run then, the strategy constructed in the proof of Claim 1 is a positional strategy for the automaton. Claim 1 together with this observation leads to the following lemma.

Lemma 1 *Let A be an alternating automaton and let w be a word. In the game associated with A and w, the automaton has a winning strategy whenver A accepts w. Moreover, if A has positional accepting run on w then the automaton has a positional winning strategy.*

What about the converse? Does the existence of a winning strategy for the automaton in the game $\mathcal{G}(A, w)$ guarantee that A accepts w? The answer turns to be yes with a rather simple proof.

Lemma 2 *Let A be an alternating automaton and let w be a word. If A accepts w then the automaton has a winning strategy in the game $\mathcal{G}(A, w)$. Moreover, if the automaton has a positional winning strategy then there is a positional accepting run on w.*

Proof: Let f be a winning strategy for the automaton. We define the run level by level. At level 1, there is a single node, the root and it is labelled by s. Suppose we have inductively constructed the run upto level i. Pick any node at level i and suppose it is labelled by the state q. Consider the path $s \longrightarrow q_1 \longrightarrow q_2 \ldots q$ from the root to this node. if q is in $Q_\forall$, and $\delta(q, a_i) = \{p_1, p_2, \ldots p_k\}$, then add k children to this node and label them as p_1, p_2, ..., p_k. Otherwise, create a single child with label $f(s \longrightarrow q_1 \xrightarrow{q}_2 \ldots \longrightarrow q)$.

Notice that every path in this tree corresponds to a play of the game that is consistent with the strategy f for the automaton. Therefore, the state labelling the last node must be in F. Thus every leaf in this tree is labelled by a state in F and hence it is an accepting run. It is trivial to check that if f is a positional winning strategy then the constructed run is positional. ■

Thus, we have established a strong relationship between A accepting the word w and the existence of a winning strategy for the automaton in the game $\mathcal{G}(A, w)$.

Theorem 3 *The automaton wins the game $\mathcal{G}(A, w)$ if and only if A accepts w.*

2 Determinacy for the games $\mathcal{G}(A, w)$

In this section we shall prove that for any automaton A and word w, either the automaton or the pathfinder has winning strategy in the game $\mathcal{G}(A, w)$. As a matter of fact we prove:

Theorem 4 *Let A be an alternating automaton and w be a word. Either the automaton or the pathfinder has a positional winning strategy in the game $\mathcal{G}(A, w)$.*

Proof: We will consider every node in $\mathcal{G}(A, w)$ as a possible starting state for the game (not just the vertex with label s in the first level) and show that in each case either the automaton or the pathfinder has a positional winning strategy. We shall prove this starting at the nodes at level $n + 1$ and work our way back.

If the game begins at a node at level $n+1$ then there are no moves to be made and if the label is in F then the automaton alwys wins the game and otherwise the pathfinder wins. The strategy is trivially positional as there are no choices available.

Suppose, we have determined for every vertex in levels $> j$ which player wins the game starting at the vertex and a positional winning strategy for that player. Consider a vertex v at level j. There are two cases to consider.

Case 1: Suppose v is labelled by a state $q \in Q_\exists$. If there is even one neighbour w of v in level j from where the automaton has a (positional) winning strategy to win the game then the automaton can also win the game beginning at v. He simply moves to w from v and then follows the (positional) strategy from w. Otherwise, the pathfinder wins the games beginning at each neighbour of v in level j. Suppose the neighbours are $v_1, v_2, \ldots v_k$. Then, the automaton has to make the first move and it will move it to some v_l. The pathfinder simply plays the (positional) strategy that is winning for him from v_l. Thus, either the automaton or the pathfinder has a (positional) winning strategy at v.

Case 2: Suppose v is labelled by a state $q \in Q_\forall$. Once again it is easy to see that if at least one neighbour of v in level j is winning for the pathfinder then the pathfinder has a (positional) winning strategy from v and otherwise the automaton has a (positional) winning strategy from v.

Thus, by induction, for each vertex v in $\mathcal{G}(A, w)$ either the automaton or the pathfinder has a positional winning strategy to win the game beginning at v. In particular, if the game begins at s either the automaton or the pathfinder has a positional winning strategy. ■

Notice that this result also implies that whenever the automaton (or the pathfinder) has a winning strategy it also has a positional winning strategy.

3 Back to Alternating Automata

Now that we have the positional determinacy for the games $\mathcal{G}(A, w)$ we can complement alternating automata and prove that their expressive power is the same as that of nondeterministic automata.

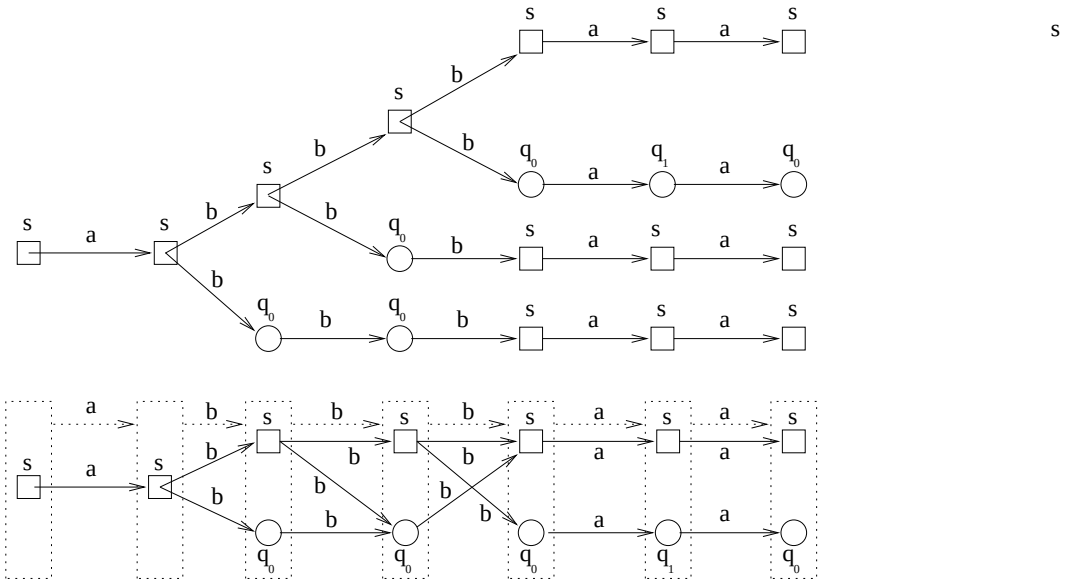
Right at the beginning of this lecture, we described a technique for complementing nondeterministic automata. In the language of alternating automata, it can be restated as follows: Given an automaton $A = (Q_\exists = Q, \emptyset, \delta, s, F)$, the automaton $(\emptyset, Q_\forall = Q, \delta, s, Q - F)$ accepts the complement of the language accepted by A. This is just a special case of the following theorem, which says that to complement an alternating automaton it suffices to exchange the $\forall$ and $\exists$ states and complement the accepting set.

Theorem 5 *Let $A = (Q_\exists, Q_\forall, \delta, s, F)$ be an alternating automata. Then $A^d = (Q_\forall, Q_\exists, \delta, s, Q - F)$ accepts the complement of $L(A)$.*

Proof: By Theorem 3 $w \notin L(A)$ if and only if the automaton does NOT have a winning strategy in the game $\mathcal{G}(A, w)$. By Theorem **??**, the automaton does not have a winning strategy in the game $\mathcal{G}(A, w)$ if and only if the pathfinder has a winning strategy in the game $\mathcal{G}(A, w)$. Finally, since the game $\mathcal{G}(A^d, w)$ is nothing but the game $\mathcal{G}(A, w)$ where the roles of the automaton and the pathfinder have been interchanged and the winning condition has been complemented, it is trivial to see that the pathfinder has a winning strategy in $\mathcal{G}(A, w)$ if and only if the automaton has a winning strategy in $\mathcal{G}(A^d, w)$. And finally, by Theorem 3, this happens if and only if A^d accepts w. ■

And finally we turn out attention to showing that every alternating automaton accepts a regular language. The naive idea that suggests itself, is to somehow generate runs of the alternating automaton using a nondeterministic automaton. A run of an alternating automaton is a tree which expands one level at a time. So the natural idea would be take the set of all possible levels as the set of states of the NFA. However, there are infinitely many possible levels, since the tree may be of unbounded width. The alternating automaton described at the beginning of the section has $k+1$ leaves on reading the word b^k. However, note that k of these leaves are labelled by s.

In general, for any alternating automaton with m states, there are at the most m distinct states at any level. So can we keep the set of states appearing at a level rather than the level itself? The answer is yes, and here is where the existence of positional accepting runs comes to our rescue. A positional run can be represented by collapsing together duplicate vertices at each level so that we are left with a DAG with at the most m nodes at each level. For example, the here is a run of the automaton A_1 on *abbbaa* and its corresponding DAG representation (ignore the dotted boxes and arrows for the moment):



The NFA we construct, attempts to generate such a DAG and accepts a word only if it can generate a DAG corresponding to a positional accepting run. The dotted boxes in the above figure represent the states of the NFA in the run on *abbbaa* and the dotted arrows are the transitions in the run.

Theorem 6 *Let $A = (Q_\exists, Q_\forall, \Sigma, \delta, s, F)$ be an alternating automaton. Let A' be the nondeterministic finite automaton $(2^Q, \Sigma, \delta', \{s\}, 2^F \setminus \{\emptyset\})$ where*

$$\delta'(X,a) = \{X' \subseteq \delta(X,a) \mid \forall q \in X \cap Q_\forall.\ \delta(q,a) \subseteq X' \ \wedge\ \forall q \in X \cap Q_\exists.\ X' \cap \delta(q,a) \neq \emptyset\}.$$

Then A' accepts $L(A)$.

Proof: In one direction, it is easy to observe that if X_i is the list of states that appear at level i in a positional accepting run on $a_1 a_2 \dots a_n$ then $X_1 \xrightarrow{a_1} X_2 \xrightarrow{a_2} X_3 \dots \xrightarrow{a_n} X_n$ is an accepting run of A'.

For the other direction, note that if $X \xrightarrow{a} Y$ is a transition in A' then any positional accepting run on any w with X as the set of labels of the leaves can be extended to a positional run on wa with Y as the set of labels of the leaves. This completes the proof.

■

In particular the NFA corresponding to A^d for some A, essentially guesses a positional run for A^d on w or equivalently a positional winning strategy for the automaton in the game $\mathcal{G}(A^d, w)$ which in turn corresponds to a positional winning strategy for the pathfinder in the game $\mathcal{G}(A, w)$.

Thus we have generalized nondeterministic automata to alternating automata. We associate a game $\mathcal{G}(A, w)$ with every word w and accepting runs of A on w are in correspondence with winning strategies for the automaton in the game $\mathcal{G}(A, w)$. The connection between the game $\mathcal{G}(A, w)$ and accepting runs extends further. The determinacy of games allows us to translate non-existence of winning strategies for the automaton to existence of winning strategies for the pathfinder. Thus, we have achieved a *quantifier switch* (i.e. we have demonstrated that we can say "exists strategy (for the pathfinder) which is winning" instead of "every strategy (for the automaton) is not winning"). This leads us naturally to the complementation of AFAs. The complement automaton is simply the dual automaton the exchanges the two players and the winning condition. The existence of positional strategies allows us to turn AFAs in NFAs.

As we shall see in the coming lectures, this trick of connecting automata to games, establishing the determinacy of the game and using that to complement the automata is a recurring theme and a very powerful technique.

Notes: Alternation as a technique was introduced by Chandra and Stockmeyer. Alternating finite state automata have been used traditionally in the setting of infinite words and infinite trees. We choose to introduce them over finite words so that technical difficulties encountered at the infinite case do not obscure the ideas that make them such a powerful technique. It is true that all our proofs about AFAs would have been simpler proofs if we used a direct method rather than use the connection to games. However, our aim is to describe the ideas behind this connection and its use at this simple setting before studying the same in more complex settings. We have also used a rather restricted definition of alternating automata, once again for the same reason.

References

Lecture 7: Büchi Automata

In this lecture we shall study finite automata as acceptors of infinite words. The study of such automata goes back to Büchi ([1]) and continues to be a topic of research ([5]).

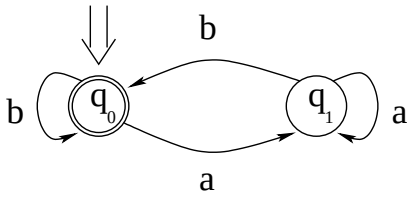
Let Σ be a finite alphabet. An infinite word (or ω-word) over Σ is simply an infinite sequence $a_1 a_2 \ldots$ where each $a_i \in \Sigma$. We shall use Σ^ω to denote the set of all infinite words over the alphabet Σ.

Let $A = (Q, \Sigma, \delta, s, F)$ be a finite automaton. There is a natural generalization of the notion of a *run* from finite to infinite words. A *run* over an infinite word $\sigma = a_1 a_2 \ldots$ is a sequence $\rho = s q_1 q_2 \ldots$ with $s \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \ldots$. But, when is such a run *accepting*? One obvious choice is to define an accepting run as one that visits some state in F. But with such a definition one can't even describe the set of words which have infinitely many as (Why?).

In any run ρ, some states of Q are visited only finite number of times and some others are visited infinitely often. Let us call these sets $\mathit{fin}(\rho)$ and $\mathit{inf}(\rho)$. There is an (infinite) suffix of the run where none of the states from $\mathit{fin}(\rho)$ appear and the states from $\mathit{inf}(\rho)$ appear infinitely often. Thus, it is reasonable to assume that the classification of a run as accepting or rejecting must rely on its behaviour in the limit and hence must depend only on $\mathit{inf}(\rho)$. Büchi's suggestion was to classify a run as accepting if it visits the set F infinitely often. Since there are only finitely many states in Q and F, this is equivalent to demanding that the run visit some fixed state in F infinitely often.

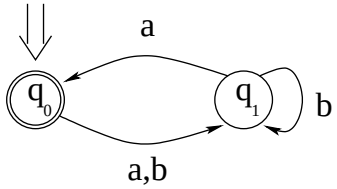
Formally, a *Büchi Automaton* is a finite automaton $A = (Q, \Sigma, \delta, s, F)$, and the language accepted by such an automaton is $L(A) = \{\sigma \mid \text{there is a run } \rho \text{ over } \sigma \text{ such that } \mathit{inf}(\rho) \cap F \neq \emptyset\}$. A langugage $L \subseteq \Sigma^\omega$ is said to be *ω-regular* if it is accepted by some Büchi automaton.

The automaton



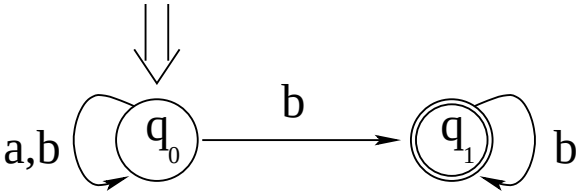
accepts all infinite words over $\{a, b\}$ in which, every a has a b occuring some where to its right. In this lecture we shall omit the ω and call a ω-word as simply a word.

The following automaton



accepts all words that have infinitely many as.

The automaton



accepts all words that have finitely many as. Call this language Finite(a).

1 Deterministic Büchi Automata

Of the three automata described above the first two are deterministic whilst the last one is not. Can one design a deterministic Büchi automaton that accepts the set of words with finite number of as? The answer is negative and this can be seen as follows: Suppose there is a deterministic Büchi automaton A accepting this language. This automaton must have an accepting run on the word ab^ω (where $b^\omega = bbbb\ldots$). Suppose this (unique) run enters a state in F after ab^{n_1} for some $n_1 \geq 1$. Now, $ab^{n_1}ab^\omega$ is also in the language and there is a unique run on this word, which extends the aforementioned run on ab^{n_1}, that is accepting and such a run must visit a state in F after reading $ab^{n_1}ab^{n_2}$ for some $n_2 \geq 1$. Repeating this argument we can construct a sequence $ab^{n_1}ab^{n_2}\ldots ab^{n_i}\ldots$ on which the unique run visits a state in F after reading ab^{n_1}, $ab^{n_1}ab^{n_2}$, $\ldots ab^{n_1}ab^{n_2}\ldots ab^{n_i}, \ldots$. Thus, this run visits the set F infinitely often and hence this string with infinitely many as is accepted by A. This contradicts our assumption that A accepted the language of words with finite number of as. Thus, nondeterministic Büchi automata are more powerful than deterministic Büchi automata.

Let L be a regular language of finite words. We define $\widehat{L}$ to be the ω-language consisting of all words that have infinitely many prefixes in L[1]. For example if $L = \Sigma^*.a$ then $\widehat{L}$ is the set of words with infinitely many as. Then, we can characterize the class of languages accepted by deterministic Büchi automata as follows:

Theorem 1 *Let A be a deterministic Büchi automaton and let $L_f(A)$ be the language of finite words accepted by A when treated as a finite automaton and let $L(A)$ be the language accepted by A as a Büchi automaton. Then,*

$$L(A) \quad = \quad \widehat{L_f(A)}$$

The proof of this theorem quite easy and we leave it as an exercise. Our proof above showing that Finite(a) is not accepted by any deterministic Büchi automaton can be seen as showing that Finite(a) is not $\widehat{L}$ for any language L.

We now examine the closure properties of ω-regular languages. Given Büchi automata recognizing languages L_1 and L_2 it is quite trivial to construct a Büchi automaton accepting the language $L_1 \cup L_2$. On the other hand constructing an automaton that accepts $L_1 \cap L_2$ requires some ingenuity. We leave that as an interesting exercise.

[1]If you wonder I write $\widehat{L}$ and not $\vec{L}$, the answer is rather embarassing. I can't get $\vec{L}$ to stretch over longer expressions like $L_1 \vec{\cup} L_2$

Exercise: Show how to construct a Büchi automaton accepting $L_1 \cap L_2$ from automata accepting L_1 and L_2.

Exercise: Show how to construct deterministic Büchi automata accepting $L_1 \cup L_2$ and $L_1 \cap L_2$ from deterministic automata accepting L_1 and L_2.

2 Complementation of Büchi Automata

We saw three different techniques to establish the closure under complementation of regular languages: via deterministic automata, via Myhill-Nerode congruences and finally via alternating automata. From the previous section it seems that the first route is not available in the case of ω-regular languages. However, using more general acceptance conditions than the Büchi condition, one can obtain deterministic automata accepting all ω-regular languages. In the next couple of lectures we shall use each of the three routes in demonstrating the closure under complementation of ω-regular languages.

The easiest technique, and the one used by Büchi himself, is via congruences. Our presentation below follows that of Thomas [7]. Recall that we associated a congruence over Σ^* with every deterministic finite automaton A, given by $x \equiv_A y$ if and only if $\forall q \in Q.\delta(q,x) = \delta(q,y)$. Since acceptance of finite words is decided by where the run starting at s ends up, this is the right notion. This equivalence is of finite index and further it saturates $L(A)$. That is, $[x]_{\equiv_A} \subseteq L(A)$ or $[x]_{\equiv_A} \cap L(A) = \emptyset$ for each $x \in \Sigma^*$. Since each $x \in \Sigma^*$ lies in some class (namely $[x]$) we can write both $L(A)$ as well as $\overline{L(A)}$ as unions of these equivalence classes (each of which is a regular language).

How do we extend these ideas to ω-regular languages? We need to extend the above construction in two ways, firstly we must be able associate a congruence with nondeterministic automata and secondly it must capture the notion of Büchi acceptance. The first step is easy: With any nondeterministic finite automaton A we can associate a congruence $\equiv_A$ defined by $x \equiv_A y$ if and only if $\forall q.\ \delta(q,x) = \delta(q,y)$ where this equality is an equality of sets. In other words, there is a run from q to q' on the word x if and only if there is a run from q to q' on the the word y (for any q and q'). This relation is a congruence, is of finite index and saturates $L(A)$. Extending this relation to capture accepting runs over infinite words requires a little bit of thought. If x and y are equivalent then we would like to be able to replace any number of occurances of x with y in any run without affecting acceptance. This leads us to the following definition: With each Büchi automaton A we associate a relation $\equiv_A$ over Σ^* given by

$$\begin{array}{rcll} x \equiv_A y & \stackrel{\triangle}{=} & \forall q, q'.\ q \xrightarrow{x} q' & \iff\ q \xrightarrow{y} q' \quad \wedge \\ & & \forall q, q'.\ q \xrightarrow{x}_g q' & \iff\ q \xrightarrow{y}_g q' \end{array}$$

where $q \xrightarrow{x}_g q'$ means that there is a run from q to q' on the word x that passes through some state in F (there may be other runs that do not pass through any state in F). We shall often write $[x]$ for $[x]_{\equiv_A}$.

If x and y are equivalent then in any infinite run we may replace any number of subruns on x by an appropriate runs on y in such a way that if the original run was accepting then the modified run continues to be accepting.

It is easy to check that this relation is a congruence. It is of finite index as the number equivalence classes is bounded by the number of functions from Q to $2^Q \times 2^Q$ (Why?). Further, we claim that for any x and y, $[x].[y]^\omega \subseteq L(A)$ or $[x].[y]^\omega \cap L(A) = \emptyset$. This can be seen as follows: Suppose $x_1y_1y_2 \ldots \in L(A)$ with $x_1 \in [x]$ and $y_i \in [y]$ for i. Consider any accepting run $\rho = s \xrightarrow{x_1} q_1 \xrightarrow{y_1} q_2 \xrightarrow{y_2} q_3 \ldots$. Let $x'_1 \equiv x_1$, $y'_1 \equiv y_1$ and so on. Then, we know that there are runs $s \xrightarrow{x'_1} q_1$, $q_1 \xrightarrow{y'_1} q_2$ and so on, such that whenever the run $q_i \xrightarrow{y_i} q_{i+1}$ visits a final state so does the run $q'_i \xrightarrow{y_i} y'_{i+1}$. Thus, the run $s \xrightarrow{x'_1} q_1 \xrightarrow{y'_1} q_2 \xrightarrow{y'_2} q_3 \ldots$ visits F infinitely often and hence $x'_1y'_1y'_2 \ldots$ is also in $L(A)$.

So what have we got so far? Suppose $\equiv$ has N congruence classes. Then, each of the N^2 ω-languages obtained as $[x].[y]^\omega$ is either completely contained in $L(A)$ or in $\overline{L(A)}$. Further, the following exercise guarantees that all these N^2 languages are ω-regular.

Exercise: Let U and V be regular languages. Show that $U.V^\omega$ is a ω-regular language.

So, it seems that we have shown that $\equiv$ "saturates" $L(A)$ and should be able to conclude that both $L(A)$ and $\overline{L(A)}$ are just finite unions of languages of the form $[x].[y]^\omega$. Then, using the above exercise we have a proof that the complement of a ω-regular language is also ω-regular. However, there is a gap. Unlike the case of finite words where it is a trival fact that each word in Σ^* lies in some equivalence class of $\equiv$, it is not clear that every ω-word is an element of $[x].[y]^\omega$ for some x, y. This needs proof and is in fact the most intricate part of Büchi's argument.

Following Gastin and Petit [4] (who attribute the original ideas to Perrin and Pin, see for instance, [8]), we shall find it convenient to use the monoid $\Sigma^*/\equiv_A$ and prove following general result about monoids.

Theorem 2 *Let M_1 be the free monoid over a (possibly infinite) alphabet Σ. M be a finite monoid and let h be a homomorphism from M_1 to M. Let $a_1a_2a_3 \ldots$ be any infinite word over Σ. Then, there are elements s and e in M such that, $e.e = e$ and $a_1a_2 \ldots \in h^{-1}(s)(h^{-1}(e))^\omega$.*

Proof: For $i < j$, let $m(i, j)$ denote $h(a_i.a_{i+1}. \ldots a_{j-1})$. This mapping is a colouring of all the 2-subsets of N using the finite set M. Then, the infinite version of Ramsey's theorem guarantees that there is a subset $i_1, i_2, \ldots$ such that the colour of any 2-subset in this set is identical. That is $m(i_j, i_k) = m(i_l, i_m)$ for any $j < k, l < m$. Let $s = m(1, i_1)$ and $e = m(i_1, i_2)$. Then, $e.e = m(i_1, i_2).m(i_1, i_2) = m(i_1, i_2).m(i_2, i_3) = m(i_1, i_3) = e$. Finally, by the definition of $m(i, j)$, $a_1 \ldots a_{i_1-1} \in h^{-1}(s)$ and $a_{i_j}a_{i_j+1} \ldots a_{i_{j+1}} \in h^{-1}(e)$.

If you do not like the infinite version of Ramsey's theorem, here is a direct argument. Let $N_0 = \{1, 2, \ldots\}$. For $i = 0, 1, 2, \ldots$ we define the set N_i, the number n_i and an element $s_i \in M$ as follows:

$$\begin{array}{lcl} n_i & = & \text{smallest number in } N_i \\ s_i & = & \text{some element of } M \text{ such that for infinitely many } j \in N_i,\ m(n_i, j) = s_i \\ N_{i+1} & = & \{j \mid m(n_i, j) = s_i\} \end{array}$$

Clearly, N_i is infinite for each i. Let e be an element that occurs as s_i for infinitely many i and let $i_1 < i_2 < \ldots$ be such that $s_{i_j} = e$. Then, $m(n_{i_j}, n_{i_k}) = e$ for any $j < k$. Thus, the indices $i_1, i_2, \ldots$ identify a Ramsey subset and the rest of the proof follows as above. Notice that since $n_0 = 1$, we also have that $s = m(1, n_{i_1}) = m(1, n_{i_2}) = \ldots$ and hence $s.e = s$. ■

A pair of elements (s, e) in a monoid with $s.e = s$ and $e.e = e$ is called a *linked pair*. Thus we have established that any infinite sequence over M_1 is in a set of the form $h^{-1}(s).(h^{-1}(e))^\omega$ for a linked pair (s, e). It turns out that linked pairs play a rather important role in the study of the algebraic theory of ω-regular languages, a topic to which we shall return later in the course. Notice that the second element of a linked pair is an idempotent and we shall exploit this fact shortly.

Using Σ^* as M_1 and $\Sigma/\equiv_A$ as M we can conclude that each $a_1a_2\ldots$ in Σ^ω is in $\eta^{-1}([x]).(\eta^{-1}([y]))^\omega$ where $\eta(x) = [x]$. But $\eta^{-1}([x]) = [x]$ and thus $a_1a_2\ldots \in [x].[y]^\omega$ for some x, y. Putting this together with our earlier calculations yields the following result:

Theorem 3 (Büchi) *Let A be a Büchi automaton. Then,*

$$\begin{aligned} L(A) &= \bigcup_{\{x,y \mid [x][y]^\omega \subseteq L(A)\}} [x][y]^\omega \\ \overline{L(A)} &= \bigcup_{\{x,y \mid [x][y]\omega \not\subseteq L(A)\}} [x][y]^\omega \end{aligned}$$

We also obtain the following characterization of ω-regular languages:

Theorem 4 *([8]) A language L over Σ^ω is a ω-regular language if and only if there is a homomorphism h from Σ^* to a finite monoid M and a colletion X of linked pairs over M such that $L = \bigcup_{(s,e)\in X} h^{-1}(s).(h^{-1}(e))^\omega$.*

One direction of this theorem is proved above and the other direction is left as a (rather trivial) exercise.

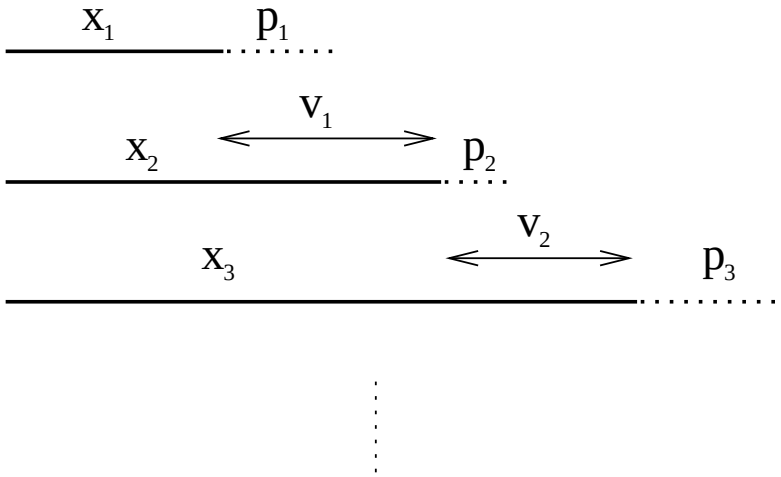
3 Determinizing Büchi Automata

We shall follow the route taken by Eilenberg and Schutzenberger [**?**], Choueka [2] as well as Rabin [9] as described by Perrin and Pin in [8]. Recall that deterministic Büchi automata recognize limit languages (i.e. languages of the form $\widehat{L}$ for some regular language L). We next show that if h is any homomorphism from Σ^* to a finite monoid M and e is an idempotent in M then $(h^{-1}(e))^\omega$ is a limit language.

Let X_e denote $h^{-1}(e)$ and let P_e denote the prefix minimal words in X_e (i.e. $P_e = \{x \in X_e \mid y < x \text{ implies } y \notin X_e\}$. Let $\sigma \in X_e^\omega$. Then, $\sigma = x_1x_2\ldots$ with each $x_i \in X_e$. For each i there is a $p_i \in P_e$ with $p_i \leq x_i$. Thus, there are infinite many prefixes of the word σ, namely $x_1p_2,\ x_1x_2p_3,\ \ldots,\ x_1x_2\ldots x_ip_{i+1}, \ldots$, in $X_e.P_e$. Thus $X_e^\omega \subseteq \widehat{X_eP_e}$. The following lemma establishes the converse.

Lemma 5 *Let h be a homomorphism from Σ^* to a finite monoid M and let e be an idempotent and let X_e and P_e be as defined above. Then, $\widehat{X_e.P_e} = X_e^\omega$.*

Proof: The discussion above established that $X_e^\omega \subseteq \widehat{X_e.P_e}$. Let $\sigma \in \widehat{X_e.P_e}$. Then, there are infinitely many x_is in X_e and p_is in P_e such that $x_1p_1 < x_2p_2 < \ldots x_ip_i \ldots < \sigma$. Suppose the lengths of these x_is were bounded. Then for some $i < j$, $x_i = x_j$ and this in turn means that $x_ip_i < x_ip_j$ contradicting the requirement that p_j has no prefixes in X_e. Thus, without loss of generality we may assume that $x_ip_i < x_{i+1}$. Let us define $v_1, v_2, \ldots$ as words such that $x_iv_i = x_{i+1}$. Note that $p_i < v_i$.



Thus, $\sigma = x_1v_1v_2v_3 \ldots$. Now, we use the fact that $x_1, v_1, v_2, \ldots$ are elements of the monoid Σ^* and Theorem 2 to conclude the existence of s and f in M such that $x_1v_1v_2 \ldots$ can be factored as $h^{-1}(s).h^{-1}(f)^\omega$ with $f.f = f$ and $s.f = s$. Since we treat $x_1, v_1, v_2, \ldots$ as elements of the monoid Σ^* the factoring respects these word boundries. That is, there is a factorization where $h^{-1}(s)$ looks like $x_1v_1 \ldots v_i$ and $v_{i+1} \ldots v_j$ is in $h^{-1}(f)$ for some $j > i$ and so on. Then, $h(x_1v_1 \ldots v_i) = h(x_{i+1}) = e$. Thus $s = e$.

From the factoring, we have that $v_{i+1} < h^{-1}(f)$ and since $p_{i+1} < v_{i+1}$ there must be a $g = h(v_{i+1}/p_{i+1})$ (where x/y is z if $x = yz$) with $eg = f$.

Here is a cute fact about idempotents. If e and f are idempotents and $eg = f$ then $ef = f$. In proof note that $ef = eeg = eg = f$. Thus, $ef = f$. But recall that $s.f = s$ and $s = e$ and thus $e.f = e$. Thus $e = f$ we have actually factored $\sigma = x_1v_1v_2 \ldots$ as $(h^{-1}(e))^\omega$ and thus $\sigma \in X_e^\omega$. ■

Now, putting together Theorem 4 and Lemma 5 we have the following characterization of ω-regular languages.

Theorem 6 *A language L is ω-regular if and only if there is a finite set I and regular languages U_i, V_i, for each $i \in I$, such that*

$$L = \bigcup_{i \in I} U_i.\widehat{V_i}$$

Actually, there is a monoid M and a homomorphism h to M and a a linked pair (s_i, e_i) for each i such that U_i is recognized via s_i and V_i via e_i. Note, that there is no hope of replacing $U_i\widehat{V_i}$ by some $\widehat{W_i}$ (Why? Use the fact that limit languages are closed under union).

Notes: The most widely used introductary article on ω-automata is [7]. Another good introduction to this topic is found in [6]. Our presentation has drawn from [7] and [8].

References

[1] J.R. Büchi: "On a Decision Method in Restricted Second-order Arithmetic", Proceedings of the 1960 Congress on Logic, Methdology and Philosophy of Science, Stanford Univeristy Press, Stanford, 1962.

[2] Y. Choueka: "Theories of automata on ω-tapes: A simplified approach", Journal of Computer and System Sciences **8** (1974) No. 2, 117–141.

[3] S. Eilenberg and M.P.Schutzenberger:

[4] Paul Gastin and Antoine Petit: "Infinite Traces", In the *The Book of Traces*, World Scienctific, 1995.

[5] O. Kupferman and Moshe Vardi: "Complementation Constructions for Nondeterministic Automata on Infinite Words", In the Proceedings of TACAS'05, Springer-Verlag Lecture Notes in Computer Science, Vol 3440, 2005.

[6] Madhavan Mukund: "Finite Automata on Infinite Words", see http://www.cmi.ac.in/ madhavan/papers/ps/tcs-96-2.ps.gz

[7] Wolfgang Thomas: "Automata over Infinite Words", In the *Handbook of Theoretical Computer Science"* Vol B, Elevsevier, 1990.

[8] Dominique Perrin and Jean-Eric Pin: *Infinite Words: Automata, Semi-groups, Logic and Games*, Academic Press, 2004.

[9] Michael Rabin:

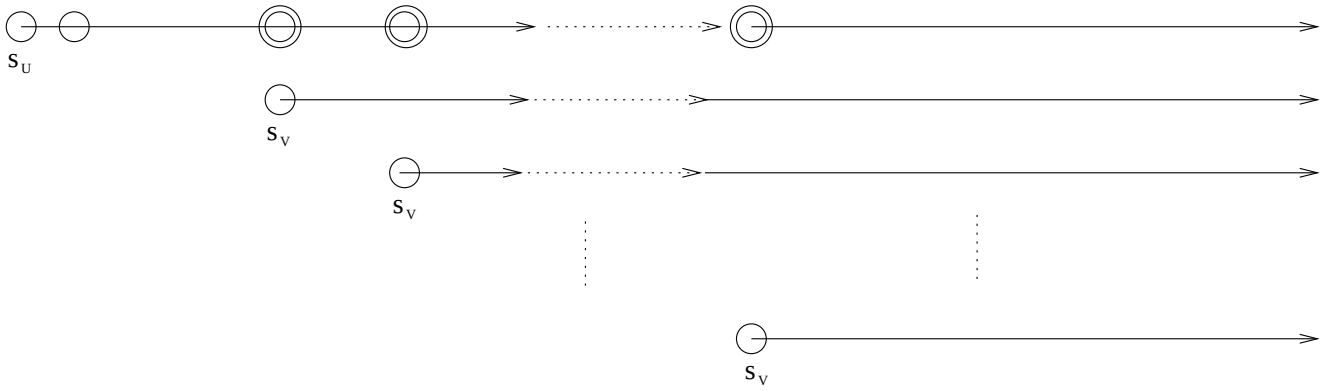
Lecture 8: Determinizing Büchi Automata

At the end of the last lecture we proved that every ω-regular language can be expressed as a finite union of languages of the form $U.\widehat{V}$. Let us try and design deterministic automata for a language of the form $U.\widehat{V}$ (This automaton cannot be a Büchi automaton. But it will differ from a Büchi automaton only in the definition of accepting runs.) Let $A_U = (Q_U, \Sigma, \delta_U, s_U, F_U)$ and $A_V = (Q_V, \Sigma, \delta_V, s_V, F_V)$ be two deterministic finite automata accepting the languages U and V respectively. Note that A_V, considered as a Büchi automaton, recognises $\widehat{V}$.

Exercise: Show that $\widehat{U.V}$ need not be equal to $U.\widehat{V}$ in general.

What we need is a sort of sequential composition of A_U and A_V. However, a given ω-word σ may have several (possibly infinitely many) prefixes that belong to U and not all of them may act as a witness to the membership of σ in $U.\widehat{V}$. That is, even though $\sigma \in U.\widehat{V}$, it may be possible to write $\sigma = u.\sigma'$ where $u \in U$ but $\sigma' \notin \widehat{V}$. So, our sequential composition must be able to pick the "right" prefix of σ. Of course, one can construct an automaton that nondeterministically picks some prefix of σ that belongs to U and verifies that the rest of the word is in $\widehat{V}$. But, we are looking for a deterministic automaton.

The right idea is to carry out a "powerset" construction on this nondeterministic automaton. Our automaton keeps a copy of A_U and whenever it identifies that a prefix of σ is in U, it forks a copy of A_V to read the rest of the input. A word is accepted if any one of these copies of A_V visits its final state infinitely often.

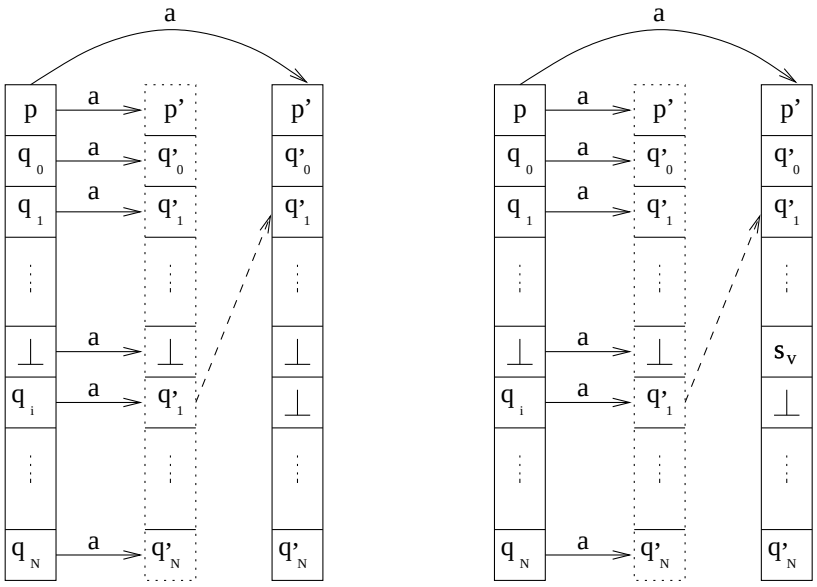


The top line is the "run" of A_U on σ. Note that at every prefix where A_U enters an accepting state, a new copy of A_V is started off. As it stands, this construction is not finite state as there may be even infinitely many prefixes of σ in U and so there would be unboundedly many copies of A_V.

The future behaviour of an automaton is completely determined by its current state. Thus, if two of these forked copies reach the same state at some point we might as well merge these two copies. Thus, at any point we will have at the most N copies where N

is the number of states in A_V. With these intuitions in mind, we propose the following construction. Let $A = (Q, \Sigma, \delta, s, \emptyset)$ be the deterministic automaton defined as follows: Q consists of elements of $Q_U \times (Q_V \cup \{\perp\})^{N+1}$ in which no element of Q_V repeats. We set $s = (s_U, \perp, \perp, \ldots, \perp)$ if $\epsilon \notin U$ and set $s = (s_U, s_V, \perp, \perp, \ldots, \perp)$ otherwise. The state is designed to carry one state of A_U and upto $N+1$ states of A_V (We only need N. The reason for using $N+1$ will be apparent soon.) We use the symbol $\perp$ to denote that a particular coordinate in the state is not currently carrying a copy of A_V.

Let $(p, q_0, q_1, \ldots, q_N)$ be an element of Q and a be in Σ. Suppose $p \xrightarrow{a} p'$ and $q_i \xrightarrow{a} q'_i$ (where we assume $\perp \xrightarrow{a} \perp$ for each $a \in \Sigma$). Note that, at least one of the q_is was $\perp$ and so at least one of the q'_is will also be $\perp$. Suppose j is the smallest index with $q'_i = \perp$. If $p' \in F_U$ then we set $q'_j = s_V$. Finally, if a state q appears more than once in $q'_0, q'_1, \ldots q'_N$ then, replace everything except the lowest index copy by $\perp$. Let the resulting tuple be $(p', q''_0, q''_1, \ldots, q''_N)$. We set $\delta((p, q_0, q_1, \ldots, q_N), a) = (p', q''_0, q''_1, \ldots q''_N)$.

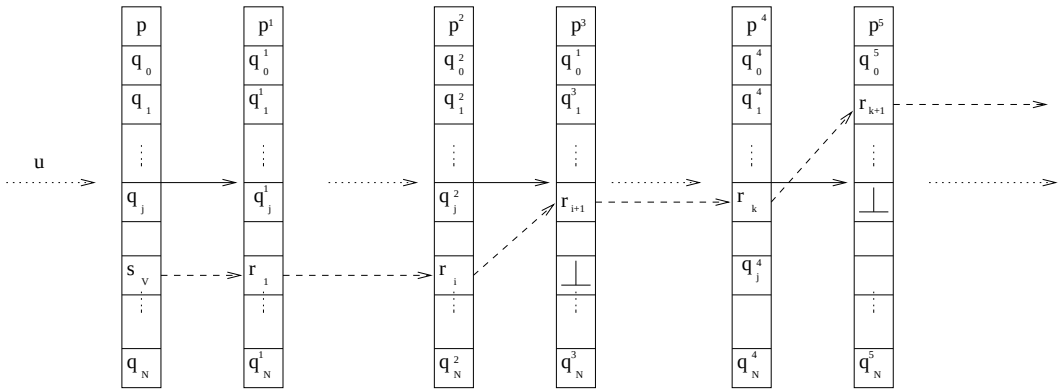


In the picture above, the left handside illustrates a move when $p' \notin F_U$ and the right handside illustrates the case when $p' \in F_U$. Note, how q'_i is replaced by $\perp$ since $q'_i = q'_1$. The run of the copy of A_V at coordinate i has been "handed over" to copy at coordinate 1. We shall refer to this as i *merges* with 1. Observe that whenever i merges with j, $i > j$.

Note that if the ith coordinate merges with the jth at a move, the resulting state necessarily has $\perp$ at coordinate i (If we did not use $N+1$ copies this would not be guaranteed. If we had only N copies then we might have to use this position to start a new copy of A_V.) Another point to note is that a newly created copy might immediately merge and thus "may not start at all". This happens if one of the other components is already at state s_V.

We have not specified an acceptance condition as yet and left it as the empty set. Let us examine the runs of this autmaton on words in $U.\widehat{V}$. Suppose $\sigma = u.\sigma'$ with $u \in U$ and $\sigma' \in \widehat{V}$. After reading u, suppose our automaton is at state $(p, q_1, \ldots, q_{i-1}, s_V, \perp, \ldots, q_N)$ (the state reached on reading u is guaranteed to be of this form since $u \in U$ and from the

definition of δ). Suppose $s_V \longrightarrow r_1 \longrightarrow r_2 \longrightarrow \ldots$ is the run of A_V on σ'. This run visits the set F_V infinite often. The copy at coordinate i that reads σ' would also follow the run $r_1 \longrightarrow r_2 \longrightarrow$, however it is not guaranteed that this copy would stay at coordinate i as it may merge with a lower numbered coordinate and merge again and so on. Here is how the corresponding run might look:



Note how the dashed line of the "accepting run" of the copy of A_V forked on reading u merges with coordinate j and then from there with coordinate 1.

But the key point to note is the following, since a coordinate merges only with a lower numbered coordinate, a copy changes coordinate only finitely often. Thus there is a finite number of moves, say n_0, after which the copy of A_V forked on reading u stays at some coordinate j forever. Thus the states $r_{n_0}, r_{n_0+1}, \ldots$ appear consecutively as the jth component of the states in some suffix of the run of A on σ. For this j, we have that

1. States from F_V appear infinitely often at coordinate j. (In particular this happens infinitely after the copy forked on reading u stabilises at coordinate j).

2. The symbol $\perp$ appears only finitely often at coordinate j.

We shall now argue that the converse is also true. Suppose the run of A on σ satisfies these two properties w.r.t. some coordinate j. Then, the run can be written in the form $s \xrightarrow{u} s' \xrightarrow{\sigma'}$ where, $u \in U$, the jth component of s' is s_V and further the jth component of every state that appears to the right of this state is not $\perp$. (A proof of this is quite easy. There are two cases to consider. If the jth component was $\perp$ sometime along the run, then take the s' to be the successor of the state where $\perp$ appears for the last time at coordinate j. Otherwise, take $s' = s$.) Suppose the jth coordinate of the states in the run $s' \xrightarrow{\sigma'}$ are $r_1, r_2 \ldots$ then, it is clear from the definition of δ that $r_1 \xrightarrow{\sigma'_1} r_2 \xrightarrow{\sigma'_2} \longrightarrow$ is the run of A_V on σ'. By requirement 2, this run visits the set F_V infinitely often. Thus $\sigma' \in \widehat{V}$ and thus $\sigma \in U.\widehat{V}$.

For each coordinate j, we define two ω-regular languages W_i and W'_i as follows:

$$\begin{aligned} W_i &= \{\sigma \mid \text{In the run } \rho \text{ of } A \text{ on } \sigma, \text{ the } i\text{th component visits } F_V \text{ infinitely often}\} \\ W'_i &= \{\sigma \mid \text{In the run } \rho \text{ of } A \text{ on } \sigma, \text{ the } i\text{th component is } \perp \text{ infinitely often}\} \end{aligned}$$

By the above discussion, $U.\widehat{V} = \bigcup_{0 \leq i \leq N} W_i \cap \overline{W_i'}$. Further, both W_i and W_i' are accepted by deterministic Büchi Automata. For W_i we use the deterministic automaton A with $F_i = \{(p, v_0, \ldots, v_N \mid p \in Q_U,\ \forall j.\ v_j \in A_V \text{ and } v_i \in F_V\}$ as the set of good/accepting states and for W_i' we use $E_i = \{(p, v_0, \ldots, v_{j-1}, \perp, v_{j+1}, \ldots, v_N) \mid p \in Q_U \text{ and } \forall j.\ v_j \in Q_V\}$ as the set of good/accepting states.

Theorem 1 *Every ω-regular language is a boolean combination of limit languages.*

Proof: From the above discussion, any language of the form $U.\widehat{V}$ can be written as a boolean combination of limit languages. We showed in the previous lecture that every ω-regular language is a finite union of languages of the form $U.\widehat{V}$. ■

1 Rabin and Müller Automata

We just showed that every ω-regular language can be expressed as a union of lanuages of the form $W_i \cap \overline{W_i'}$, where W_i and W_i' are limit languages accepted by Büchi automata which differ merely in the set of good/accepting states. Rabin suggested an accepting criterion that directly reflects this requirement.

Definition 2 *A Rabin automaton A is of the form $A = (Q, \Sigma, \delta, s, (E_1, F_1), (E_2, F_2) \ldots (E_k, F_k))$ where Q, Σ, δ and s are as in the case of Büchi automata. Each E_i and F_i is a subset of Q. Runs of such an automaton are defined as in the case of Büchi automata. A run ρ is accepting if there is an i, $1 \leq i \leq k$ such that $inf(\rho) \cap E_i = \emptyset$ and $inf(\rho) \cap F_i \neq \emptyset$. Thus a run is accepting if and only if there is an i such that, the set E_i is visited finitely often and F_i is visited infinitely often.*

A Rabin automaton is deterministic if δ is a transtion function. Thus the automaton constructed above, A, with $(E_0, F_0), \ldots (E_N, F_N)$ as the set of accepting pairs (where the definition of E_i and F_i is given at the end of the last section) is a deterministic Rabin automaton that accepts $U.\widehat{V}$.

Earlier Müller had proposed that acceptance be defined by directly specifying the $\inf(\rho)$s permitted.

Definition 3 *A Muller automaton A is of the form $A = (Q, \Sigma, \delta, s, \mathcal{F})$ where $\mathcal{F} \subseteq 2^Q$. A run ρ is accepting if $inf(\rho) \in \mathcal{F}$. As usual, such an automaton is said to be deterministic if δ is a function.*

We can equip A with the following $\mathcal{F}$

$$\mathcal{F} \quad = \quad \{X \subseteq Q \mid \exists i.\ X \cap E_i = \emptyset \ \wedge \ X \cap F_i \neq \emptyset\}$$

Quite clearly this automaton accepts a word precisely when the Rabin automaton described earlier accepts a word and that is precisely when the word is in $U.\widehat{V}$. (Notice that

this construction allows us to turn any Rabin automaton into an equivalent Müller automaton.) Thus, if we can show that deterministic Müller automata are closed under union, that would complete the proof showing that all ω-regular languages are accepted by deterministic Müller automata, giving us the desired determinzation construction.

Lemma 4 *If L_1 and L_2 are accepted by deterministic Müller automata then $L_1 \cup L_2$ is accepted by a deterministic Müller automaton.*

Proof: Let $A_1 = (Q_1, \Sigma, \delta_1, s_1, \mathcal{F}_1)$ and $A_2 = (Q_2, \Sigma, \delta_2, s_2, \mathcal{F}_2)$ be deterministic Müller automata accepting L_1 and L_2 respectively. Let $A = (Q_1 \times Q_2, \Sigma, \delta_1 \times \delta_2, (s_1, s_2), \mathcal{F})$, where $\mathcal{F} = \{X \subset Q \mid X\downarrow_1 \in \mathcal{F}_1\} \cup \{X \subseteq Q \mid X\downarrow_2 \in \mathcal{F}_2\}$ (where $X\downarrow_i$ is the projection to coordinate i of X). It is quite easy to check that A accepts $L_1 \cup L_2$. ■

Exercise: Show that deterministic Rabin automata are closed under union.

With this we have established that all ω-regular languages are accepted by Deterministic Müller automata. But can they accept more? Notice that the language accepted by a deterministic Müller automaton $A = (Q, \Sigma, \delta, s, \mathcal{F})$ is a boolean combination of the languages accepted by the Deterministic Büchi automata A_q, $q \in Q$, given by $A_q = (Q, \Sigma, \delta, s, \{q\})$. In particular,

$$L(A) \quad = \quad \bigcup_{F \in \mathcal{F}} \left(\bigcap_{q \in F} L(A_q) \cap \bigcap_{q \notin F} L(A_q)\right)$$

Exercise: Why does this construction fail in the case of Nondeterministic Muller automata?

Thus we have established that deterministic Müller automata accept precisely the class of ω-regular languages. We now show that even nondeterministic Müller automata accept ω-regular languages.

Lemma 5 *Let $A = (Q, \Sigma, \delta, s, \mathcal{F})$ be a Müller automaton. Then, $L(A)$ is ω-regular.*

Proof:(Sketch) We construct a NBA accepting $L(A)$ as follows: the NBA simulates the A. Further it nondeterministically guesses a set $X \in \mathcal{F}$ and a position in the run and verifies that after this point, every state in X is hit infinitely often and no state outside of X is visited. How do we check that everything in X is hit infinitely often? Order the set X as say $x_1, x_2, \ldots x_k$. The automaton keeps a current index from the set $\{0, 1, 2, \ldots k\}$. Whenever the automaton visits a state x_i and the current index is $i > 0$, the current index is incremented to $i + 1$ (modulo k+1) otherwise the index is left unchanged. If the current index is 0, the index is always incremented to 1 in the next move. We use a Büchi condition to simply check that the current index is 0 infinitely often. ■

Exercise: Write down the construction described in the proof of Lemma 5 precisely.

Theorem 6 *The following statements are equivalent:*
1. L is a ω-regular language.
2. L is a finite union of languages of the form $U.\widehat{V}$.
3. L is a boolean combinated of languages of the form $\widehat{V}$.
4. L is accepted by some nondeterministic Büchi automaton.
5. L is accepted by some nondeterministic Müller automaton.
6. L is accepted by some nondeterministic Rabin automaton.
7. L is accepted by some deterministic Müller automaton.
8. L is accepted by some deterministic Rabin automaton.

Proof: The equivalence of the first 4 statements was proved earlier. We have just shown that NBA can be translated to DMA and that NMA can be translated to NBA. Thus NBA, NMA and DMA are equivalent. We also showed that NRA can be translated to NMA. One of the exercises (showing that DRAs are closed under union) shows that NBAs can be translated to DRAs. That established the equivalence of NRAs and DRAs with ω-regular languages. ■

2 MSO over infinite words

We can interpret the logic MSO over infinite words. The first order variables $x, y, \ldots$ are interpreted as positions in the word, second order variables $X, Y, \ldots$ are interpreted as sets of positions, $x \in X$, $a(x)$ etc. have the usual interpretations. For instance, the following formula

$$\forall x.\exists y.\ (x < y) \wedge a(x)$$

asserts that there are infinitely many a's.

Exercise: Write down a formula in MSO that asserts that there is some subword of the form $ba^{2*n}b$ for some $n > 1$.

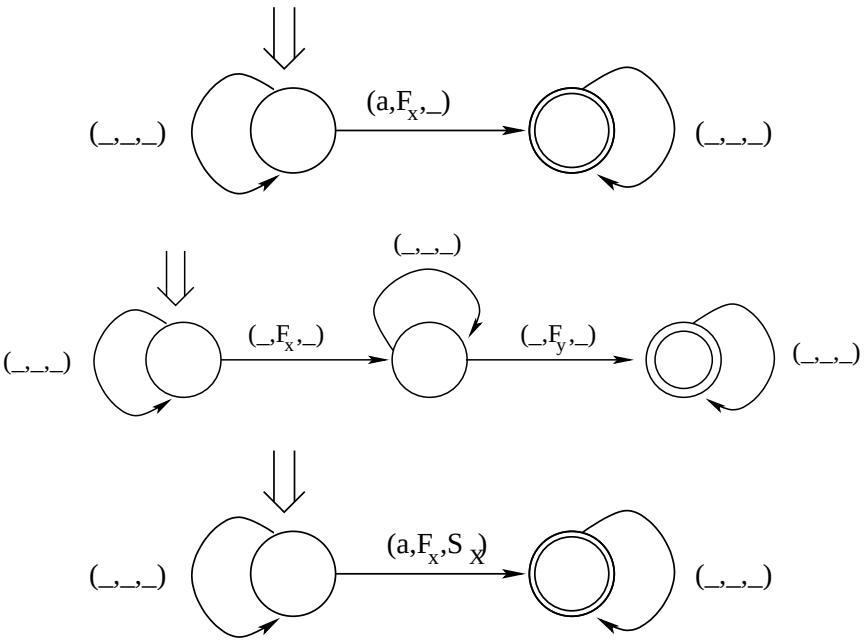
Let $A = (Q, \Sigma, \delta, s, F)$ be a NBA. Using one variable X_q for each state $q \in Q$, it is quite easy to write down a formula in MSO, that describes the set of words accepted by A. There are some differences w.r.t. to the proof for the finite case: Firstly, A may be nondeterministic. It is quite trivial to generalize the construction there to work over nondeterministic automata. Secondly, we need to describe the Büchi acceptance rather than asserting that the automaton enters a final state at the end. Büchi acceptance is described by the following formula:

$$\forall x.\exists y.(x < y) \wedge \bigvee_{q \in F} (y \in X_q)$$

Thus, every ω-regular language can be described using sentences in MSO.

In order to describe the languages defined by formulas with free variables, we shall consider (V, W)-words where V is a set of first order variables and W is a set of second order variables (i.e. ω-words over the alphabet $\Sigma \times 2^V \times 2^W$ where each $x \in V$ appears exactly in one letter in the word). It is quite easy to construct a NBA accepting precisely the set of (V, W)-words. We now sketch an argument showing that for any MSO formula ϕ, and sets V and W with $\text{free}_1(\phi) \subseteq V$ and $\text{free}_2(\phi) \subseteq W$, the set of (V, W)-words satisfying ϕ is a ω-regular language.

The construction of a NBA automaton accepting $L(\phi)$ proceeds by induction on the structure of the formula. In all cases we assume that the autmaton is intersected with an automaton recognising the set of (V, W)-words. Here are the automata for $a(x)$, $x \in X$ and $x < y$ in that order:



These are the same automata as used in Lecture 3. We simply interpret then as NBAs here.

Since we have already established that Büchi automata are closed under boolean operations, translating the boolean operators is quite trivial and thus we are left with the quantifiers. As usual, we can express the universal quantifiers using the existential quantification and negation.

Suppose $\phi = \exists x.\alpha$. Let $\text{free}_1(\phi) \subseteq V$ and $\text{free}_2(\phi) \subseteq W$. Then $\text{free}_1(\alpha) \subseteq V \cup \{x\}$ and $\text{free}_2(\alpha) \subseteq W$. By the induction hypothesis, the set of $(V \cup \{x\}, W)$-words satisfying α is a ω-regular language. Further, $(a_1, F_1, S_1)(a_2, F_2, S_2) \ldots$ satisfies α if and only if $(a_1, F_1 \setminus \{x\}, S_1), (a_2, F_2 \setminus \{x\}, S_2) \ldots$ satisfies $\exists x.\alpha$. In the finite word case, we simply applied closure under homomorphism at this point. Since homomorphic images of infinite words might be finite words, we can't directly use homomorphism closure. However, it is rather trivial to construct a Büchi automaton that guesses a position for x and verifies that resulting word satisfies α. We leave the details as an exercise. The construction for $\exists X.\alpha$ also proceeds similarly.

Thus we have the following theorem due to J.R.Büchi.

Theorem 7 *A language L is ω-regular if and only if it can be described by a sentence in MSO.*

As an immediate corollary of this result we have:

Corollary 8 *The problem of checking whether a formula ϕ in MSO is satisfied by any word is decidable. The problem of checking whether a formula ϕ in MSO satisfied by all words is also decidable.*

Proof: We can translate ϕ into a Büchi automaton A such that $L(\phi) = L(A)$. The first question corresponds to checking emptiness of the automaton A, and the second corresponds to checking that $L(A) = \Sigma^*$. Both of which are decidable. ■

3 Applications of MSO

As an application of this theorem, we shall show that Presburger arithmetic, that is, the theory of natural numbers with addition, is a decidable theory.

Formulas in Presburger arithmetic use first order variables $x, y, \ldots$ that range over natural numbers and the predicate $x + y = z$. Here is a valid sentence in this logic:

$$\forall x.\forall y.\exists z.(x + z = y) \vee (y + z = x)$$

Here is a sentence that is not true:

$$\forall x.\forall y.\exists z.(x + z = y)$$

Notice that the relation $x < y$, $x = y + 1$, and so on can be defined in Presberger arithmetic. Presberger showed that the problem of determining whether a given sentence in Presburger arithmetic is valid (or true) is decidable, by the method of quantifier elimination. Büchi showed that one can translate sentences in Presburger arithmetic to sentences in MSO (over the one letter alphabet) preserving validity. We can then use Corollary 8 to conclude the decidability of Presburger arithmetic.

There is exactly one word over the one letter alphabet. Let the positions of this word be $0, 1, 2, \ldots$. Büchi's idea as to denote a natural number n by a subset of positions. Suppose $n = 2^{i_1} + 2^{i_2} + \ldots + 2^{i_k}$. Then, n is represented by the set $\{i_1, i_2, \ldots, i_k\}$. Conversely, every finite subset of $0, 1, 2, \ldots$ represents a number. The translation map $\mathcal{T}$ is defined as follows: $\mathcal{T}(\exists x.\phi) = \exists X.Fin(X) \wedge \mathcal{T}(\phi)$, $\mathcal{T}(\forall x.\phi) = \forall X.Fin(X) \Rightarrow \mathcal{T}(\phi)$, $\mathcal{T}(\phi \vee \phi') = \mathcal{T}(\phi) \vee \mathcal{T}(\phi')$, $\mathcal{T}(\neg\phi) = \neg\mathcal{T}(\phi)$, and $\mathcal{T}(x + y = z) = \mathsf{Add}(X, Y, Z)$. Here, $\mathsf{Fin}(X)$ simply asserts that the set X is finite and it is quite easy to write a formula saying this. That leaves the definition of $\mathsf{Add}(X, Y, Z)$. We need to show that there is MSO formula that verifies that the number reprepresented by the set Z is the sum of the numbers represented by the sets X and Y. One can write this down directly, but we prefer the route via automata. We need

to show that the set of $(\emptyset, \{X, Y, Z\})$-words satisfying the sum property described above is a ω-regular language. Each input letter can be thought of as consisting of 3 bits, one bit indicating whether the position is in X or not, and similarly one each for Y and Z. With this interpretation, the automaton is reading three bit sequences starting at the least significant bit and it needs to verify that one sequence is the sum of the other two. This is an easy programming exercise. Thus, we can translate formulas in Presburger arithmetic to MSO.

Theorem 9 *(Presburger/Büchi) The theory of natural numbers with addition is decidable.*

Things change quite drastically if we allow multiplication and this is one of the implications of Gödel's famous theorem.

As a second application of the decidability of MSO we turn to model checking. This is a problem that arises in computer science. You are given a program and a property of runs of programs and you wish to check that every run of the program satisfies this property. For example, the property could say "It is never the case that more than one process is in the critical section" or " whenever a process requests for a resource, it is eventually granted access to the resource". Such properties can be quite easily seen to expressible in MSO. On the program side, we know that almost everything is undecidable if we work with general programs. However, if we restrict the set of programs to finite state programs then we can think of the program as Buchi automaton (where all states are accepting) A_p and the property as a MSO formula that can also be translated in to a Büchi Automaton A_ϕ and the model checking problem reduces to checking if $L(A_p) \subseteq L(A_\phi)$. But this is quite easily seen to be decidable.

Exercise: What is the complexity of the complementation construction for NBAs described in this lecture?

It turns out that translating MSO into automata is prohibitively expensive. On the other hand, there are restricted logics, in which properties such as the two listed above are expressible, and that can be translated to automata with reasonable cost. These will be some of the topics we shall study later in this course.

It turns out that it is possible to construct a complement BA whose size is bounded by $O(2^{O(nlogn)})$. We shall examine this in the coming lectures. Note that the translations from one kind of ω-automata to another is also an expensive operation. We shall examine the exact complexity of these translations later.

Lecture 9: Büchi Games over Infinite Graphs

We shall first generalize the definition of alternating automata over finite words. Recall that in our definition of alternating automata, the set of states Q is divided into two sets $Q_\forall$ and $Q_\exists$ and every transition out of any $Q_\forall$ state is interpreted as a logical and while transitions out of any $Q_\exists$ state are interpreted as logical ors.

Instead of associating the type of the transition to a state, we could associate it with a (state, letter) pair. In other words, we define an alternating automaton to be a tuple $(Q, \Sigma, \delta, s, F)$, where for each $q \in Q$ and $a \in A$, $\delta(q, a) = \forall S$ or $\delta(q, a) = \exists S$, for some some set S of states. The interpretation of these transitions is the obvious one: if $\delta(q, a) = \forall S$ then, we need to start one copy of the automaton for each state in S to read the rest of the input, and $\delta(q, a) = \exists S$ represents a nondeterministic choice and we need to pick one state from S to read the rest of the input.

For each such automaton A and word w we can associate a game. The only difference w.r.t the game defined in the Lecture 6 is that, whether a state q in the ith copy belongs to the automaton or the pathfinder depends on the input letter a_i. if $\delta(q, a_i)$ is a $\forall$ transition then the state belongs to the pathfinder and it belongs to the automaton otherwise. And with this definition we can reprove all the results in Lecture 6 once again without any difficulty.

We can generalize this further as follows: We define an alternating automaton to be a tuple $(Q, \Sigma, \delta, s, F)$ where $\delta(q, a)$ is positive boolean formula over Q. That is, a formula constructed using elements of Q and the operators $\vee$ and $\wedge$ as well as the constants false and true. In particular, negation is not permitted. Before venturing into the precise definition of runs etc, let me explain this definition informally. Suppose $\delta(q, a) = (q_1 \vee q_2) \wedge q_3$. This means that in order to read the word aw starting at state q, we must start one copy of the automaton at state q_3 to read w and a second copy starting at state q_1 or q_2 to read w. We could also start three copies, one each at states q_1, q_2 and q_3 (recall, that all the copies must accept for a run to be accepting).

Let $S \subseteq Q$ and let ϕ be a formula over Q. We define when S satisfies the formula ϕ as follows:

$S \models \mathsf{true}$	always
$S \models q$	if $q \in S$
$S \models \phi_1 \wedge \phi_2$	if $S \models \phi_1$ and $S \models \phi_2$
$S \models \phi_1 \vee \phi_2$	if $S \models \phi_1$ or $S \models \phi_2$

We can now define a run of an alternating automaton from state q on a word $w = a_1 a_2 \ldots a_n$ to be a tree labelled by Q satisfying the following properties

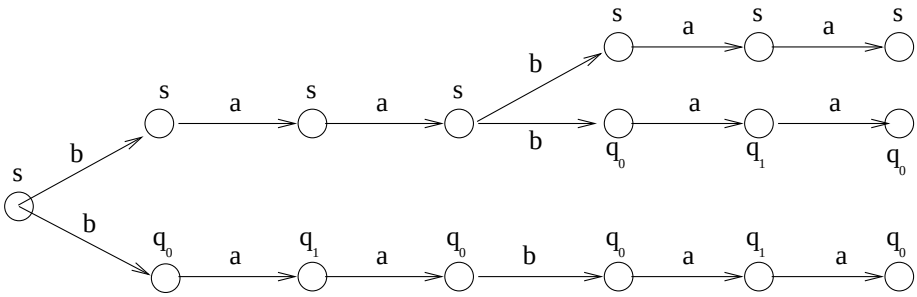
1. The tree has $n + 1$ levels.
2. The root is labelled by q.
3. If a node at level i is labelled by q and then the labels of its children constitute a set S such that $S \models \delta(q, a_i)$.

The run is accepting if all the leaves of this tree are labelled by states in F and a word w is accepted if there is an accepting run starting at state s on w.

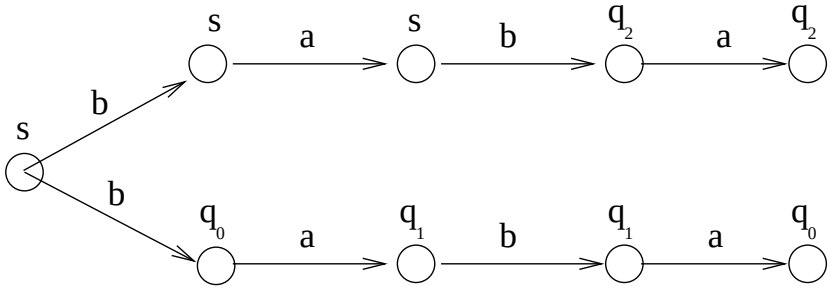
We illustrate these definitions with the automaton $A_1 = (\{s, q_0, q_1, q_2\}, \{a, b\}, \delta, s, \{s, q_0, q_2\})$ where

$$\begin{array}{lcl} \delta(s, a) & = & s \vee q_2 \\ \delta(s, b) & = & (s \vee q_2) \wedge (q_0 \vee q_2) \\ \delta(q_0, a) & = & q_1 \\ \delta(q_0, b) & = & q_0 \\ \delta(q_1, a) & = & q_0 \\ \delta(q_1, b) & = & q_1 \\ \delta(q_2, a) & = & q_2 \\ \delta(q_2, b) & = & \mathsf{false} \end{array}$$

Here are is an accepting run of this automaton on the input *baabaa*:



The set $\{s, q_0\}$ satisfies $(s \vee v_2) \wedge (q_0 \vee q_2)$. Here is an accepting run on *baba*.



Here we also use the fact that $q_2 \models (s \vee q_2) \wedge (q_0 \vee q_2)$.

Exercise: What is the language accepted by this alternating automaton?

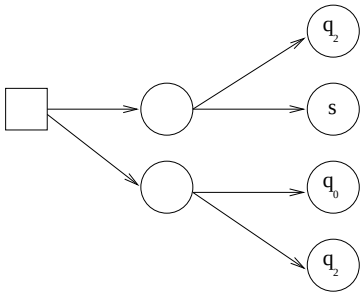
We shall say that a run is *positional*, if for each level i and any two states in level i labelled by the same state, the entire subrun (subtree) rooted at these states is identical.

1 Games from Automata

How do we associate a game with a given automaton A and word w? In the "simpler" version of alternating automaton considered earlier, nodes where the choice is nondeterministic belong to the automaton while nodes with conjunctive choice belong to the pathfinder.

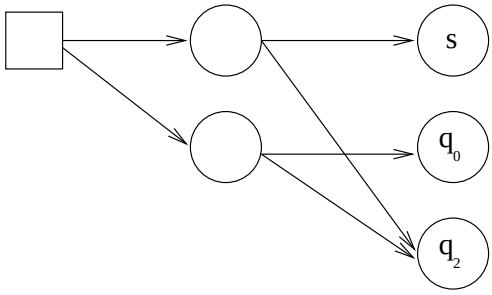
However, in the more elaborate model a transition may involve both nondeterministic and conjunctive choices. (For example consider $\delta(s, b)$ above.)

The idea is to first translate each transition $\delta(q, a)$ into a small game graph. First we construct the tree representation of this formula with the operators in the formula constituting the interior nodes and the states in the formula constituting the leaves. For example, the formula $(s \wedge q_2) \vee (q_0 \wedge q_2)$ gives:

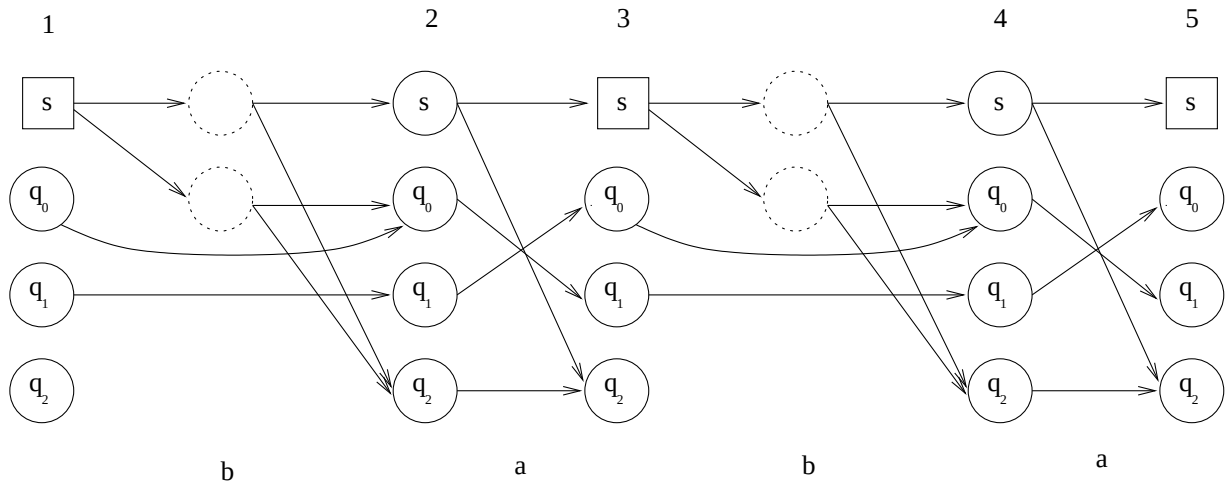


The nodes corresponding to the $\wedge$ operator are treated as pathfinder nodes and and those corresponding to $\vee$ operator belong to the automaton. The leaves of this tree are labelled by states.

We then combine the leaves labelled by the same state. In the above example, we get:



Finally, the game graph corresponding to an automaton A and word $a_1 a_2 \ldots a_n$ is constructed using $n + 1$ copies of the set state set Q. The levels are connected up as follows: Pick the game graph associated with $\delta(q, a_i)$, identify its root with the copy of q at level i and its leaves with their corresponding copies in level $i + 1$. Thus, the type of a state q at level i is determined by the outermost operator in $\delta(q, a_i)$. Here is the game graph associated with the automaton A (described earlier) on the input word *baba*.



The state s belongs to the pathfinder at levels 1 and 3 since the first and third letters are b. It belongs to the automaton at levels 2,4 and 5. Also notice that there are states (corresponding to the operators in the transition formulas) that appear in between the levels.

We can show that an automaton A accepts a word w if and only if the player automaton has a positional winning strategy in the game $\mathcal{G}(A, w)$. Recall that in games played on DAGs either the automaton or the pathfinder always has a positional winning strategy. So we might as well restrict our attention to positional winning strategies.

Fix any positional strategy f for the automaton in a game G. By $\mathcal{R}(f, p)$ we shall refer to the subgraph consisting of all positions in the game graph that appear in some play consistent with f and beginning at position p. (We shall write $\mathcal{R}(f)$ if p is the copy of s in level 1.)

We can show that if f is a positional winning strategy of the automaton then $\mathcal{R}(f)$ corresponds to the folding into a DAG (where nodes with identical subtrees rooted below them have been identified) of some positional accepting run of the automaton.

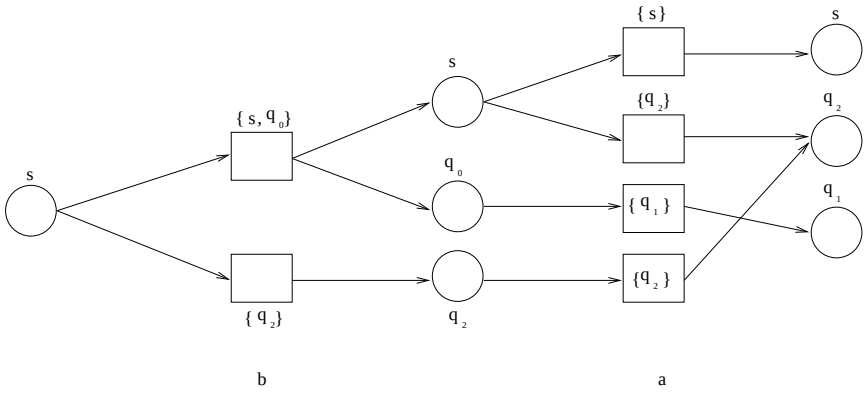
We wish to reformulate this game into a more useful form. First note that if G_ϕ is the game graph obtained from a formula ϕ, and f is ANY strategy for the automaton in this game then the set of labels of the leaves in $\mathcal{R}(f, r)$, where r is the root (the node corresponding to the top level operator), satifies ϕ. Conversely, note that if a set S satisfies a formula ϕ then there is a strategy f_S for the automaton that ensures that the leaves in $\mathcal{R}(f_S, r)$ are labelled by a subset of S.

Thus a strategy for the automaton over a graph of the form G_ϕ corresponds to restricting the set of possible outcomes (i.e. the state reached at the end of a play) to some subset S that satisfies ϕ (or if ϕ is false, then the automaton is stuck and cannot make a move.) and which of these states in S is reached at the end of a particular play is fixed by the choices made by the pathfinder.

Thus, we can reformulate the game $\mathcal{G}(A, w)$ equivalently as the follows: The game begins at state s at level 1. The automaton picks a subset X_1 of Q that satisfies $\delta(s, a_1)$. The pathfinder then picks an element q_1 of X_1. The automaton then picks a subset satisfying $\delta(q_1, a_1)$ and then the pathfinder picks a state and so on till we reach some state q_n. The

automaton wins if q_n is in F and the pathfinder wins otherwise (with the understanding that the pathfinder wins if at any point the automaton is unable to make a move.) Notice that if X satisfies $\delta(q, a)$ then it makes no sense for the pathfinder to play any set X' with $X \subseteq X'$. So we might as well restrict the choices available to the automaton to minimal subsets satisfying ϕ. Henceforth by $\mathcal{G}(A, w)$ we shall refer to this game.

Here is the game corresponding to the earlier defined automaton A on input ba.



In this setting it is quite easy to see that if winning strategies for the automaton on $\mathcal{G}(A, w)$ correspond to accepting runs for A on w and positional winning strategies correspond to positional accepting runs for A on w. Thus we have the following theorem:

Theorem 1 *An alternating automaton A accepts a word w if and only if the player automaton has a positional winning strategy in the game $\mathcal{G}(A, w)$.*

As discussed in Lecture 6, this immediately results in the equivalence of nondeterministic and alternating automata. The nondeterministic automaton simulates the runs of the the alternating automaton level by level and by keeping just one copy of each state that appears at a level.

Corollary 2 *For every alternating automaton A, there is a nondeterministic automaton A' such that $L(A) = L(A')$.*

Is it easy to complement these generalized alternating automata? To complement alternating automata, the standard technique is as follows: First transform the automaton A into another automaton A' so that the game graph $\mathcal{G}(A', w)$ is simply the game graph $\mathcal{G}(A, w)$ with the roles of the two players interchanged. Secondly, complement the accepting condition. A' with the complement accepting condition accepts a word if and only the automaton wins in $\mathcal{G}(A', w)$ if and only if the pathfinder wins in $\mathcal{G}(A, w)$ if and only if A does not accept w.

How to construct such an A'? Here we find the original definition of the game to be more useful. Recall that for $\delta(q, a)$ we simply constructed a game graph where the nodes labelled by $\wedge$ belongs to the pathfinder while $\vee$ nodes belong to the automaton. Thus, by simply

interchanging the $\vee$ and $\wedge$ operators we get a game graph where the roles of the pathfinder and automaton are interchanged. Given a formula ϕ we write ϕ^d (i.e. the dual of the formula ϕ) for the one obtained by replacing $\wedge$ and $\vee$ by $\vee$ and $\wedge$ respectively (and false is replaced by true and vice versa). Thus, we have the following theorem:

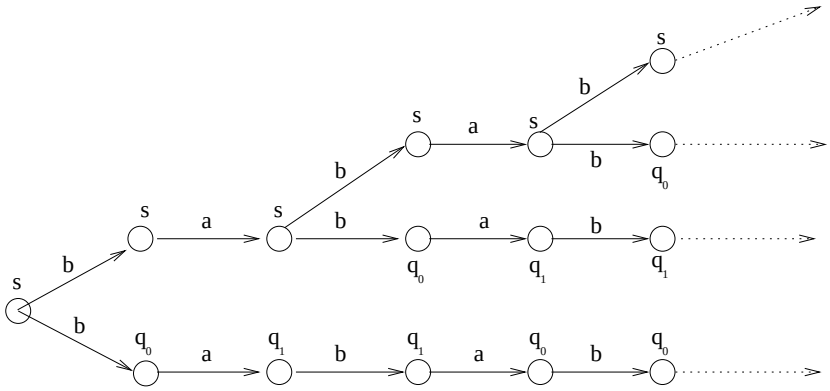
Theorem 3 *Let $A = (Q, \Sigma, \delta, s, F)$ be an alternating automaton. Then, the automaton $A' = (Q, \Sigma, \delta^d, s, Q \setminus F)$, where $\delta^d(q, a) = \delta(q, a)^d$, accepts the complement of the language accepted by A.*

2 Alternating Büchi Automata

An *Alternating Büchi Automaton* is a tuple $A = (Q, \Sigma, \delta, s, F)$ where $\delta(q, a)$ is a positive boolean formula over Q. Thus it is essentially a alternating automaton treated as an automaton over infinite words. The run of such an automaton over an infinite word $a_1 a_2 \ldots a_n \ldots$ is a infinite tree labelled by states from Q satisfying:

1. The root is labelled by q.
2. If a node at level i is labelled by q and then the labels of its children constitute the set S such that $S \models \delta(q, a_i)$.

A run is accepting if every complete path through the run tree is infinite and visits the set F infinitely often. Here is an accepting run of the automaton A_1 on the input *bababa* ...



As usual we say that a runtree is positional if it has the following property: for any two nodes labelled by the same state at the same level, the subtrees rooted at these two nodes are identical.

Clearly the class of languages accepted by Alternating Büchi automata subsumes the class of ω-regular languages. What about the converse? Can we simulate every alternating Büchi automaton using a traditional Büchi automaton? In the case of finite words, we simulated the alternating automaton level by level using the fact that we need to keep only one copy of each state at a level. This in turn relied on the fact that alternating finite automata have positional accepting runs on every word they accept and this was proved by establishing

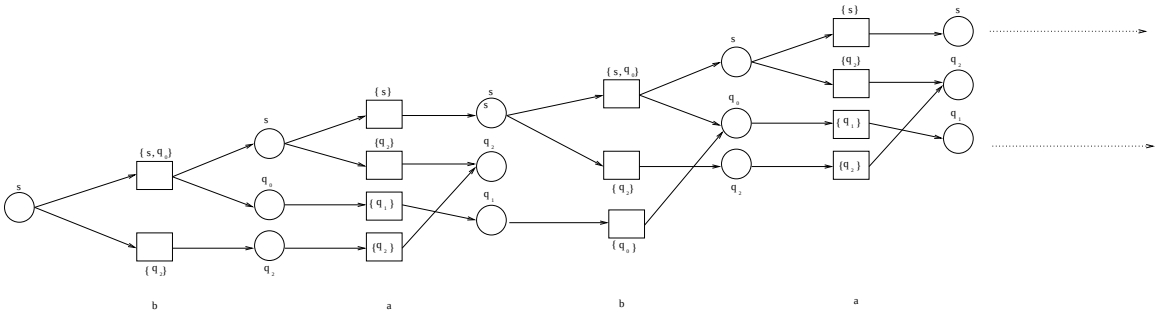
that the player automaton has a positional winning strategy in the game (associated with the automaton and any word w).

We shall follow the same route here. First we associate a game with every automaton–word pair. Then we show that the word is accepted if and only if the player automaton has a positional winning strategy in the game. This in turn implies the existence of positional accepting runs, which in turn allows us to simulate alternating automata via nondeterministic automata.

2.1 The Game $\mathcal{G}(A, \sigma)$

Let A be an automaton and a ω-word $\sigma = a_1 a_2 \ldots$ we associate a game as follows. The nodes are classified as those in level $1, 2, \ldots$. At level $2i - 1$ there are nodes labelled by the elements of Q. At level $2i$ there are nodes labelled by elements of 2^Q. Nodes in the odd levels belong to the automaton and the nodes at the even levels belong to the pathfinder. From a node labelled q at level $2i-1$, there is an edge to a state labelled X in level $2i$ if and only if $X \models \delta(q, a_i)$. From a node labelled X at level $2i$ there is an edge to a node labelled q at level $2i + 1$ if and only if $q \in X$. Thus, the automaton picks sets of states that satisfy the transition formula and the pathfinder picks a state from this set.

Since we are interested only in the result of games starting at the state s at level 1 we might as well omit all the states that are not reachable in any play starting at this state. We may also assume that the automaton will never play an X' if there is an $X \subseteq X'$ that also satisfies the transtion formula. (Thus, this is exactly the same game as defined for the finite case, however, we have not stated the winning condition yet.) Here is the game graph $\mathcal{G}(A_1, bababa \ldots)$:



In this game, the winning criterion is as follows: Any play that is finite is winning for the pathfinder (this happens if the play reaches a state q at some level i with if $\delta(q, a_i) = \mathsf{false}$). An infinite play is winning for the automaton if it visits states in F infinitely often. This game is what is called a *Büchi Game* (we define and analyse Büchi games in the next section). Thus, a winning strategy for the automaton is one in which, every play consistent with the strategy is infinite and visits the set F infinitely often. A winning strategy for the pathfinder is one in which every play consist with the strategy is either finite or visits elements of F only finitely often.

For any strategy f of the automaton we set $\mathcal{R}(f)$ to be the subgraph of the game graph that is reached by some play consistent with the strategy f. It is quite easy to see that we

can unfold the DAG $\mathcal{R}(f)$ (i.e. expand the DAG into a tree, duplicating vertices whenver necessary in the obvious manner) for any strategy f for the automaton to get a run for A on σ. This unfolding yields a positional run whenever the strategy is positional. And clearly, the strategy is winning implies that the unfolded runtree is accepting.

Conversely, given an accepting run for A on w, we can easily construct a winning strategy in the game: For any position $s \to X_1 \to x_1 \to X_2 \to x_2 \ldots \to X_i \to x_i$, if $s \to x_1 \to x_2 \ldots x_i$ is a path in the accepting run and if X is the set of labels of the children of this x_i, then play $x_i \to X$. If this path does not appear in the accepting run, play anything. It is quite easy to verify that if the automaton plays this strategy, then any play will stay within a path that appears in the accepting tree and hence visit the set F infinitely often and will therefore be winning for the automaton. Further, notice that if the accepting run was positional then this construction yields a positional winning strategy for the automaton. Thus we have the following theorem:

Theorem 4 *Let A be an alternating Büchi automaton and let σ be an ω-word. A has an (positional) accepting run on w if and only if the player automaton has a (positional) winning strategy in the game $\mathcal{G}(A, \sigma)$.*

3 Reachability Games

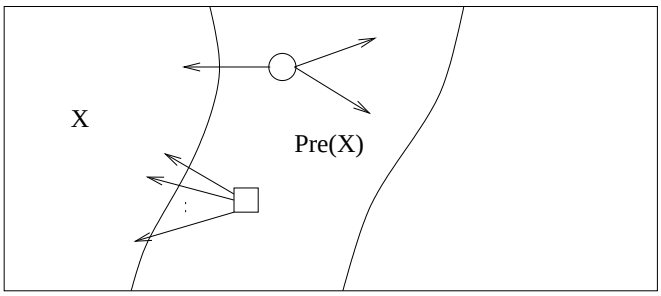
We begin by considering a simple game called the *reachability game*. In a reachability game we are given a graph $G = (V_0, V_1, \to)$ (where V_0 is the set of nodes from where player 0 makes moves and V_1 is the set of vertices from where player 1 makes moves) and a set $X \subseteq V = V_0 \cup V_1$. The aim of player 0 is to ensure that the game enters some vertex in X while the aim of player 1 is to ensure that this does not happen. For the moment, we shall assume that every vertex has at least one out going edge.

We would like to calculate the set of nodes W_0 from which the player 0 can force the game to enter X. Clearly $X \subseteq W_0$. What else? Consider any vertex $v \in V_0$ with at least one outgoing edge into X. Clearly player 0 can win from v too. Moreover, if $v \in V_1$ and all the outgoing edges from v go into X then once again player 0 will win from v (he needs do nothing, the first move by player 1 will force the game into X.)

For any set U let us define $\mathsf{pre}(U)$ to be the following set:

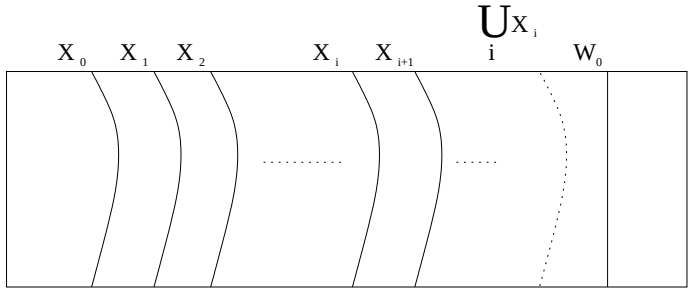
$$\mathsf{pre}(U) \quad = \quad \{v \in V_0 \mid \exists w.\ v \to w \ \wedge \ w \in U\} \bigcup \{v \in V_1 \mid \forall w.\ v \to w \ \Rightarrow \ w \in U\}$$

The above argument says that $X \subseteq W_0$ and $\mathsf{pre}(X) \subseteq W_0$ and moreover player 0 has a positional strategy to win from all the nodes in $X \cup \mathsf{pre}(X)$.

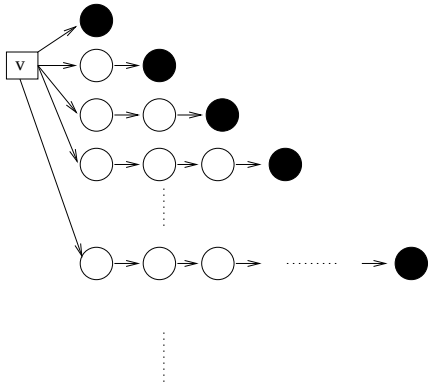


We can generalise this to say that whenvever $U \subseteq W_0$ then $\mathsf{pre}(U) \subseteq W_0$ and if player 0 has a positional strategy that ensures that he wins the game starting at any position in U then he also has a positional winning strategy that wins the game starting at any position in $U \cup \mathsf{pre}(U)$. The strategy is the following: Within U play the (positional) winning strategy that is promised by the hypothesis. For any node $q \in \mathsf{pre}(U) \cap V_0$, we are assured that there is at least one neighbour w in U. Fix such a w for each q and the strategy plays $q \rightarrow w$ at q. Thus, after the first move the game enters U and in U the strategy is already assured force the game into X after some sequence of moves. For any $v \in \mathsf{pre}(U) \cap V_1$, the first move by player 1 will move the game into U where the winning strategy for player 0 will force the game to enter X. Thus, player 0 has a winning strategy from all of $U \cap \mathsf{pre}(U)$ and in particular has a positional winning strategy if he has a postional strategy to win from U.

Therefore, if we set $X_0 = X$ and $X_{i+1} = \mathsf{pre}(X_i) \cup X_i$ then, for all i, $X_i \subseteq W_0$.



However, there may be nodes in W_0 that are not in $\bigcup_{i<\omega} X_i$. This is because we have placed no restrictions on our game graphs. They could be of any cardinality, and the degree of a vertex could be infinite or even uncountable. In the following game (where the nodes in X are the dark coloured ones), the node $v \notin \bigcup_{i<\omega} X_i$ but player 0 wins from v.



But note that $\mathsf{pre}(\bigcup_{i<\omega} X_i) \subseteq W_0$ and we could continue our iterations to higher ordinals. We define the sequence X_i as follows:

$$\begin{array}{lcll} X_0 & = & X & \\ X_{i+1} & = & \mathsf{pre}(X_i) \cup X_i & i+1 \text{ is a successor ordinal} \\ X_\beta & = & \bigcup_{i<\beta} X_i & \text{when } \beta \text{ is a limit ordinal} \end{array}$$

We know that there is a smallest ordinal κ such that $X_\kappa = X_{\kappa+1}$. We shall now show that $W_0 = X_\kappa$. We define the positional winning strategy over X_κ now. Suppose $v \in X_\kappa \cap V_0$ then there is a smallest ordinal β such that $v \in X_\beta$. We call this ordinal $\mathsf{ord}(v)$. From the inductive construction it is clear that $\beta = i+1$ for some i (or $\beta = 0$ which means that player 0 has already won. So we may pick any outgoing edge as the strategy.) Therefore there is some $w \in X_i$ and $v \to w$. Fix such a w and define the positional strategy at v to be $v \to w$. We call this strategy $\mathsf{reduce}(X)$ and we shall refer to X_κ as $\mathsf{wforce}(X)$.

Also note that if $v \in X_\kappa \cap V_1$ and $\mathsf{ord}(v) = i+1$ then every edge out of v leads to a node in X_i. Thus, if the game starts in X_κ and player 0 plays the strategy $\mathsf{reduce}(X)$ then every move in the play reduces the value of ord. But ordinals are wellfounded and so within finite number of moves the play must enter some node node w with $\mathsf{ord}(w) = 0$, in other words, the game must enter the set X. Thus, this positional strategy is winning for player 0 starting at any node in X_κ.

Suppose $v \in V \setminus X_\kappa$. Since $v \notin \mathsf{pre}(X_\kappa)$, it follows that either $v \in V_0$ and $\forall w.\ (v \to w) \Rightarrow w \in V \setminus X_\kappa$ or $v \in V_1$ and $\exists w.\ (v \to w) \ \wedge\ w \in V \setminus X_\kappa$. Thus, from a node in $V \setminus X_\kappa$, player 0 cannot move the game into X_κ.

Here is a positional strategy $\mathsf{avoid}(X)$ for player 1: Given any $v \in V_1 \setminus X_\kappa$, pick any $v \to w$ such that $w \in V \setminus X_\kappa$ and play $v \to w$ as the move at v. (For nodes in X_κ the moves for player 1 are defined arbitrarily.) This positional strategy ensures that player 1 never moves the game into X_κ from any vertex in $V \setminus X_\kappa$.

Combining the conclusions of the previous two paragraphs we see that player 1 has a positional strategy that ensures that a game starting in $V \setminus X_\kappa$ stays within $V \setminus X_\kappa$ for ever and thus results in a win for player 1 (since $X \subseteq X_\kappa$). This gives us the following *determinacy theorem* for reachability games:

Theorem 5 *Let $(V_0, V_1, \to)$ and X specify a reachability game. We can partition the set V as W_0 and W_1 such that player 0 has a positional winning strategy that wins the game starting at any position in W_0 and player 1 has a postional winning strategy that wins the game starting at any position in W_1.*

Also observe that if G is a finite graph then the sets W_0 and W_1 can be computed and the positional winning strategies for player 0 and 1 can also be computed.

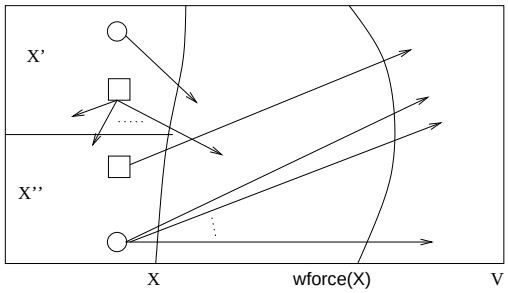
An useful fact: If the game starts within $V \setminus \mathsf{wforce}(X)$ and player 1 plays $\mathsf{avoid}(X)$ then the play never enters $\mathsf{wforce}(X)$ (or equivalently stays within $V \setminus \mathsf{wforce}(X)$.

Exercise: Given a finite reachability game G what is the complexity of computing the winning sets for player 0 and 1 ?

3.1 A useful variant

Suppose we modified that the reachability game to demand that the game visit X after at least one move has been made (i.e. player 0 does not win immediately if the game starts in a vertex in X.). Then what are the winning sets for player 0 and player 1?

Notice that any node in $\mathsf{wforce}(X) \setminus X$ is still winning for player 0. Because, starting at such a node , the strategy $\mathsf{reduce}(X)$ forces the game to visit the set X. Also, every node in $V \setminus \mathsf{wforce}(X)$ is still winning for player 1 as he can keep the game within $V \setminus \mathsf{wforce}(X)$. So the only nodes which may change hands are those within X. States in X can be divided into two sets X' and X'' where $X' = X \cap \mathsf{pre}(\mathsf{wforce}(X))$ and $X'' = X \setminus \mathsf{pre}(\mathsf{wforce}(X))$.



We claim that player 0 has a positional winning strategy that wins from all the vertices in $\mathsf{wforce}(X) \setminus X \cup X'$. At vertices in $\mathsf{wforce}(X) \setminus X$ it plays the usual $\mathsf{reduce}(X)$ strategy. If $v \in X' \cap V_0$ then, there is a $w \in \mathsf{wforce}(X)$ such that $v \rightarrow w$. Fix such a w and player 0 plays $v \rightarrow w$ at v. It is quite easy to check that this is a winning strategy for player 0 starting at any position in $\mathsf{wforce}(X) \setminus X \cup X'$ and we shall henceforth refer to this set as $\mathsf{force}(X)$.

Player 1 has a positional winning strategy that wins from all the vertices in $V \setminus \mathsf{wforce}(X) \cup X''$. The strategy for player 1 is the usual $\mathsf{avoid}(X)$ strategy at vertices in $V \setminus \mathsf{wforce}(X)$. For a vertex v in $X'' \cap V_1$ notice that there is at least one edge $v \rightarrow w$ with $w \in V \setminus \mathsf{wforce}(X)$. Fix such a w and player 1 player $v \rightarrow w$ at this v. When player 1 plays this strategy, after the first move the game never enters the set $\mathsf{wforce}(X)$ and thus is winning for player 1.

We shall refer to these strategies for player 0 and 1 as $\mathsf{reduce}(X)$ and $\mathsf{avoid}(X)$ (since they continue to be winning positional winning strategies for 0 and 1 in the simple reachability game though over different winning sets).

4 Büchi Games

A Büchi game consists of game graph $(V_0, V_1, \rightarrow)$, a finite set of colours C, a labelling function λ assigning a colour to each vertex, and a set $F \subseteq C$. A play in this game is winning for player 0 if it visits vertices coloured with colours from F infinitely often. For any Büchi automaton A and word w, the game $\mathcal{G}(A, w)$ is a Büchi game. Take the colouring set to be

$Q \cup \{\bot\}$. Colour every vertex at the even levels by $\{\bot\}$ and colour a vertex q at any odd level by q.

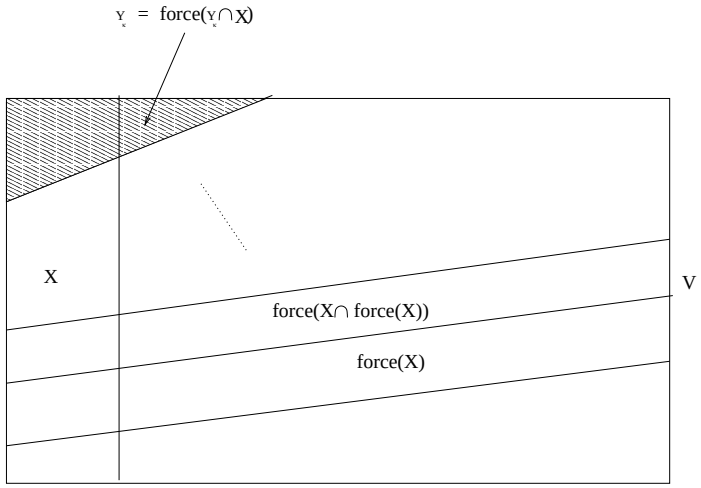
We wish to establish that in every Büchi game, the set of nodes can be divided into to sets such that player 0 has a postional winning strategy for games starting in one set and player 1 has a positional winning strategy starting in the other.

So, we are interested in vertices from where player 0 can force the game to enter the set $X = \{v \mid \lambda(v) \in F\}$ infinitely often. We know that $\mathsf{force}(X)$ is the set of vertices from where he can force the game to enter X at least once (not including the start vertex). From where can we force the game to visit vertices in X at least twice? If we start from a vertex in $\mathsf{force}(X \cap \mathsf{force}(X))$ then clearly we can force the game to enter some vertex $v \in X \cap \mathsf{force}(X)$ and from such a vertex we can clearly force the game to enter X once more. Similarly, from a vertex in $\mathsf{force}(X \cap \mathsf{force}(X \cap \mathsf{force}(X)))$ player 0 can force the game to enter X at least three times and so on.

If we can find a set Y such that $Y = \mathsf{force}(X \cap Y)$ then clearly, player 0 would win starting at any vertex in Y. He simply plays the $\mathsf{reduce}(X \cap Y)$ strategy and this would force the play to enter the set $X \cap Y$. But $Y = \mathsf{force}(X \cap Y)$ and so the play would be forced to enter $Y \cap X$ again (Note that this argument would not work if we had used wforce instead) and so on. Thus the play would enter X infinitely often and player 0 wins.

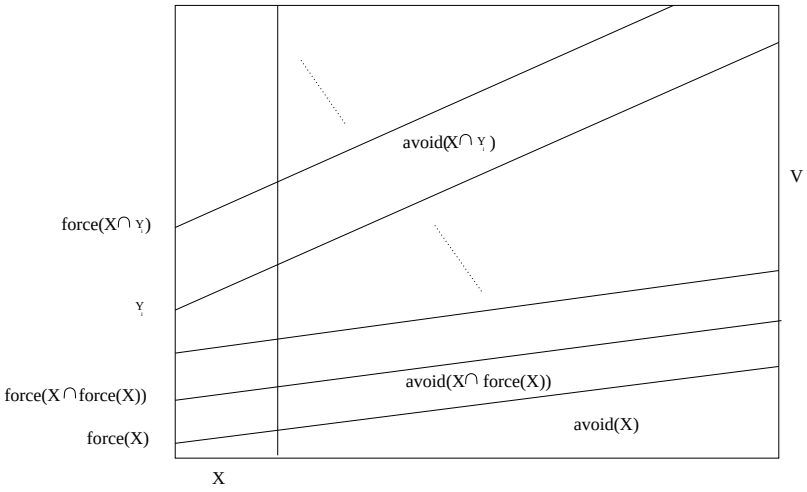
The Tarski-Knaster theorem suggests how to find the most generous such Y and here is how: Define a sequence $Y_0, Y_1, \ldots$ as follows:

$$\begin{array}{lcll} Y_0 & = & V & \\ Y_{i+1} & = & \mathsf{force}(X \cap Y_i) & i+1 \text{ is a successor ordinal} \\ Y_\beta & = & \bigcap_{i<\beta} Y_i & \text{when } \beta \text{ is a limit ordinal} \end{array}$$



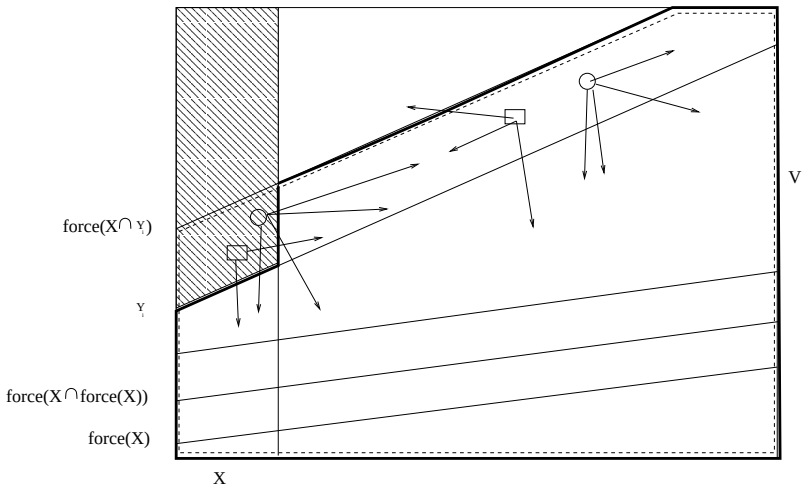
Notice that the function force is monotone (i.e. if $S \subseteq T$ then $\mathsf{force}(S) \subseteq \mathsf{force}(T)$). And since $Y_0 = V$, $Y_1 \subseteq Y_0$. Thus, for all i $Y_{i+1} \subseteq Y_i$ and consequently $Y_i \subseteq Y_j$ for any $i > j$. Thus, there must a least ordinal κ such that $Y_{\kappa+1} = Y_\kappa$. Thus $Y_\kappa = \mathsf{force}(X \cap Y_\kappa)$ and thus player 0 has a positional winning strategy (i.e. $\mathsf{reduce}(X \cap Y_\kappa)$). that wins from all positions in Y_κ.

What about vertices not in Y_κ? Suppose $x \in V \setminus Y_\kappa$. $Y_0 = V$, therefore there is a least i such that $x \in Y_i$ and $x \notin Y_{i+1}$ for some successor ordinal $i+1$. We define $\mathsf{ord}(x)$ to be the least ordinal i such that $x \notin Y_{i+1}$. If $\mathsf{ord}(x) = i$ then $x \in V \setminus \mathsf{force}(X \cap Y_i)$. Within the set $V \setminus \mathsf{force}(X \cap Y_i)$ player 1 can play $\mathsf{avoid}(X \cap Y_i)$ to ensure that the play never enters $X \cap Y_i$. We define a positional strategy for player 1 as follows: at any $x \in V \setminus Y_\kappa$ play $\mathsf{avoid}(X \cap Y_{\mathsf{ord}(x)})$.



Notice that as long as player 1 plays $\mathsf{avoid}(X \cap Y_i)$ the game can never enter Y_{i+1} and thus all vertices visited in such a play have ord values less than or equal to i. If at some point the ord value j becomes strictly less than i then our strategy would play $\mathsf{avoid}(X \cap Y_j)$ and consequently the game would never visit a vertex in Y_{j+1} and so on . Thus, in any play consistent with this strategy for player 1, the ord values of the vertices visisted forms a nonincreasing sequence.

Now let us examine what happens whenever a play enters a vertex $x \in X$. Suppose $\mathsf{ord}(x) = i$.



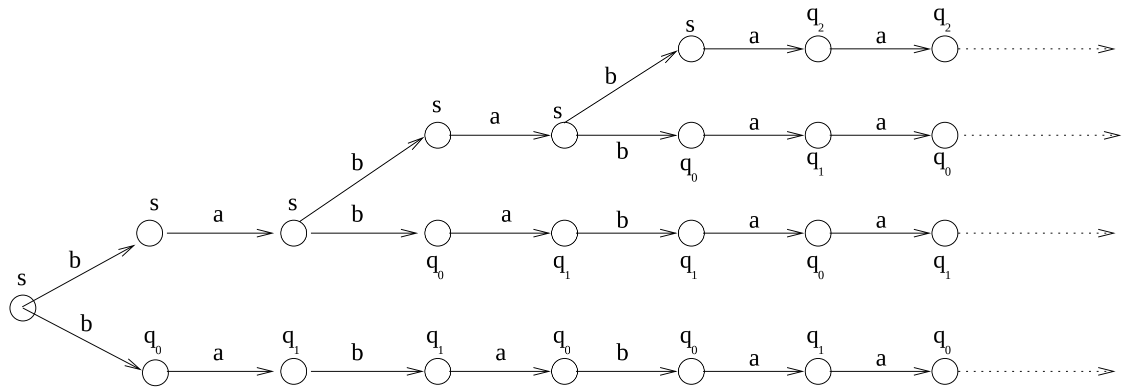
In this figure, the area enclosed by the thick boundary is $V \setminus \mathsf{wforce}(X \cap Y_i)$ while the area enclosed by the dotted boundary is $V \setminus \mathsf{force}(X \cap Y_i)$ (i.e. $V \setminus Y_{i+1}$). If $x \in X$ and $\mathsf{ord}(x) = i$, then $x \in X \cap Y_i$ (and therefore $x \in \mathsf{wforce}(X \cap Y_i)$). On the other hand, $x \notin \mathsf{force}(X \cap Y_i)$. Therefore, by the discussion in section 3.1, the $\mathsf{avoid}(X \cap Y_i)$ strategy for player 1 ensures that the play never returns to $X \cap Y_i$ after the first move. In particular, the first move results in the game moving to $V \setminus \mathsf{wforce}(X \cap Y_i)$. Thus as long as the game stays within positions with ord equal to i, the game will never return to $X \cap Y_i$. And since every vertex $v \in X$ with $\mathsf{ord}(v) = i$ is in Y_i this means that game never returns to X as long as the ord value stays at i. This together with the fact that the ord values are nonincreasing along any play consistent with our strategy ensures that X is visited only finitely often.

Thus, we have the following theorem:

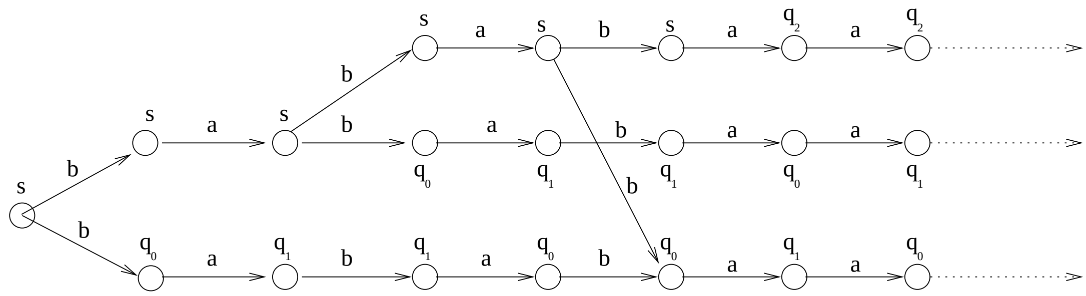
Theorem 6 *In any Büchi game the set of vertices can be partitioned in two sets W_0 and W_1 such that player 0 has a positional winning strategy that wins all plays starting from any position in W_0 and player 1 has a positional winning strategy that wins all plays starting at any position in W_1.*

Lecture 10: Complementation via Alternating Automata

The last theorem of Lecture 9 assures us that in any Büchi game one of the two players has a positional winning strategy. Thus, for any alternating Büchi automaton (ABA) A and word w, either the automaton or the pathfinder has a positional winning strategy in the game $\mathcal{G}(A, w)$. Thus, if A accepts w there is a positional run for A on w. Consider the ABA A_1 from the previous lecture. Here is a positional accepting run for this automaton on input *babab*$aaaa\ldots$.



Observe that positionality only places requirements on occurances of a state at the same level and NOT across levels. The move at the s-labelled state at level 2 is not the same as that at the s-labelled state at level 6. However, both the q_0-labelled states at level 6 have identical subtrees below them. In any level, we can collapse together all the vertices that are labelled by the same state (or equivalently (since the run is positional) whenever the entire subtree rooted at these vertices is isomorphic) to obtain a DAG. Such a DAG has at the most $|Q|$ vertices at each level. Here is the DAG obtained from the above run:



Henceforth, by a "positional run" we refer to this DAG representation of the run. We can also directly define this run-DAG for an automaton $A = (Q, \Sigma, \delta, s, F)$ on an input $a_1 a_2 \ldots$ as the DAG satisfying:

1. The vertices at level i in are labelled by elements of $Q \times \{i\}$ (and at the most one vertex is labelled by any label.)

2. There is unique vertex at level 1 labelled by $(s, 1)$.

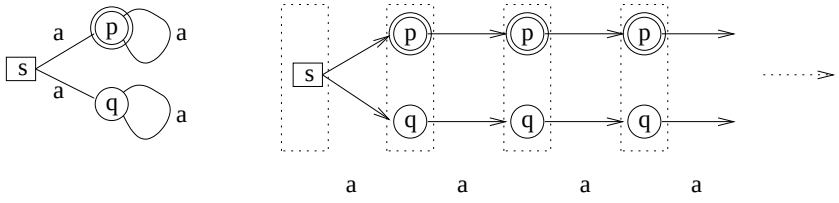
3. If some vertex at level i is labelled by (q, i) and the set of vertices at level $i+1$ to which this vertex is connected by edges is $\{(q_1, i+1), (q_2, i+1) \ldots (q_m, i+1)\}$ then $\{q_1, \ldots q_m\} \models \delta(q, a_i)$.

(We could also insist that any vertex at levels $2, 3, \ldots$ must have an indegree of at least 1, but we don't have to.)

A run is accepting if every vertex in the run has an outdegree of at least 1 and every path through the DAG visits vertices in F infinitely often.

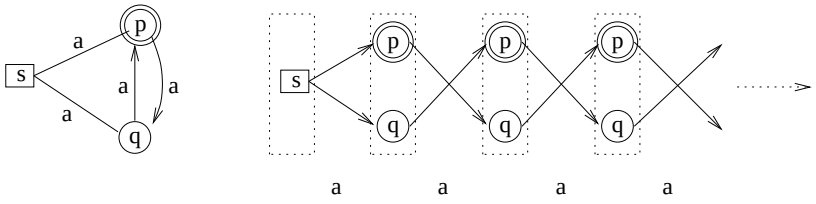
As in the case finite automata, positional runs of an alternating automaton can be simulated using a nondeterministic automaton which simply generates the run-DAG level by level. That is, given an alternating autmaton $A = (Q, \Sigma, \delta, s, F)$ we can construct the nondeterministic automaton $A' = (2^Q, \Sigma, \delta', \{s\}, \emptyset)$ (ignore the accepting set for the moment), with $\delta'(X, a) = \{X' \mid \forall q \in X.\ X' \models \delta(q, a)\}$. Thus X' must have enough states to ensure that $\delta(q, a)$ is satisfied for each $q \in X$. It is quite easy to check that this automaton simulates the positional runs of the alternating automaton level by level.

Can we set the accepting set to some F' so that the resulting NBA accepts the same language as A? Well, some obvious candidates are $F_1' = \{X \mid X \cap F \neq \emptyset\}$ and $F_2' = \{X \mid X \subseteq F\}$. Will either of these work? Here is a counter example to the choice F_1'. The transitions out of s are conjunctive and the right-handside of the figure indicates the unique run of this automaton on $aaa \ldots$.



At each level there is at least one state from F, however the run is not accepting as there is a path that does not visit F infintely often. Thus this word is not accepted by the alternating Büchi automaton. On the other hand, every level of this run intersects the set F. In the above figure, the dotted boxes indicate the states of the NBA simulating the ABA level by level. From the second state onwards every state of this NBA is in F_1 and thus it would erroneously accept this word.

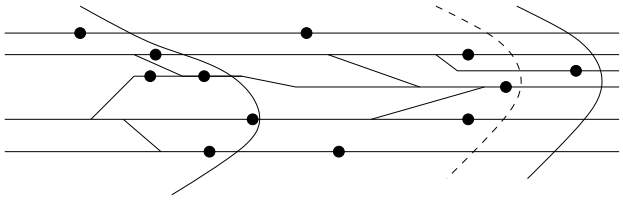
Now, consider the following example:



Here the word is accepted by the alternating automaton as you can see that every path in the unique run on $aaa \ldots$ visits p infinitely often. The states of the NBA are marked by the dotted lines. Notice that none of these states would belong to F_2' since $q \notin \{p\} = F$.

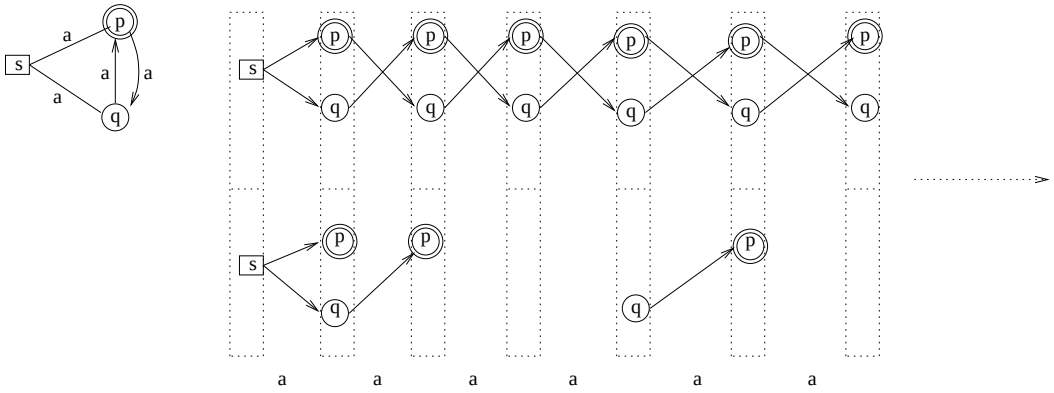
In effect the first choice is an overapproximation and the latter an underapproximation. It turns out that in order for the NBA to simulate the ABA and verify that every path visits F infinitely often we need to maintain more information in the NBA. The idea is to do the following:

1. Simulate the ABA level by level.
2. Verify that we can slice the run into infinitely many segments such that in each segment every path in the segment visits F.



The dark circles denote elements of F. The solid curve on the left identifies a segment (everything to its left) in which every path visits an element of F. The part between the first solid curve and the dotted curve is not good enough. There is one path ("forked" within this segment) that does not visit states in F. The part between the first and second solid curves is a good segment — every path in this segment visits F.

We simulate the run of the ABA level by level as usual. At the "beginning" of each slice, we make a copy of the current set of paths (that are not already in a state in F) in the simulation. In this second copy we extend a path only if it has not visited a final state yet. If at some point the second copy becomes empty (indicating that we have completed one more good slice), we copy the current set of paths from the first copy and repeat this process all over again. If the second copy becomes empty infinitely often then we know that the run can be sliced into infinitely many good slices and thus every path in this run visits F infinitely often. Here is an alternating automaton and a run of its equivalent NBA on the word $aaa\ldots$.



The dotted rectagular boxs contains the states of the NBA. Each box is divided into two by a line in the middle. The part of the state above the middle line simulates the ABA

level by level. Below the middle line, a simulation on the current slice is carried out. At the beginning s is copied at the lower level and after 2 moves we find that all paths starting at s have visited a state in F. Thus the lower component of the state reached after reading aaa is empty indicating that the first slice has ended. In the next move we copy the non-final states from the upper copy to start the simulation on next slice and so on.

Here is a formal description of the NBA A' simulating the alternating automaton A. $A' = (Q', \Sigma, \delta', (\{s\}, \{s\}), F')$ where

$$\begin{array}{lcl} Q' & = & \{(X,Y) \in 2^Q \mid Y \subseteq X\} \\ F' & = & \{(X,\emptyset) \mid X \in 2^Q\} \\ \delta'((X,Y),a) & = & \{(\bigcup_{q\in X} X_q, \bigcup_{q\in Y} X_q \setminus F) \mid \text{ where } X_q \in 2^Q \text{ such that } X_q \models \delta(q,a)\} \end{array}$$

Note that for states in $X \cap Y$, we make the same move in both copies and clearly this is necessary.

Theorem 1 *(Miyano and Hayashi) For any alternating Büchi automaton A with n states there is a nondeterministic Büchi automaton A' with at the most $2^{O(n)}$ states with $L(A') = L(A)$. Thus the class of languages accepted by alternating Büchi automata is the class of ω-regular languages.*

Proof: It is not difficult to check that the automaton A' defined above satisfies the requirements of this theorem. ■

1 Complementation and co-Büchi Automata

How do we complement alternating Büchi automata? Well, we take the automaton with the "dual" transition relation and "complement" the acceptance condition. Unfortunately, the complement of the Büchi acceptance condition is NOT a Büchi condition. The complement says that "the path does NOT visit F infinitely often" or equivalently it says that " the path visits F finitely often". This is not equivalent to insisting that $Q \setminus F$ is hit infinitely often as a path could visit both Q as well as $Q \setminus F$ infinitely often.

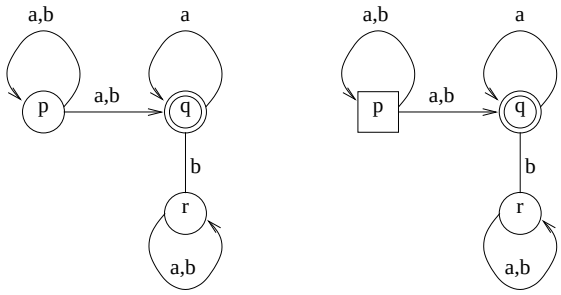
Let us define a *co-Büchi automaton* A to be a tuple $(Q, \Sigma, \delta, s, F)$ where Q, Σ, δ, s and F are as before. The definition of a run is also the same as for Büchi automata. The only difference is in the definition of *accepting runs*. Here, a run ρ is accepting if $inf(\rho) \cap F = \emptyset$. We can also extend this idea to define alternating co-Büchi automata (Aco-BA). We leave the details to the reader.

Theorem 2 *Let $A = (Q, \Sigma, \delta, s, F)$ be an alternating Büchi automaton. Then, the alternating co-Büchi automaton $A^d = (Q, \Sigma, \delta^d, s, F)$ accepts the language $\overline{L(A)}$.*

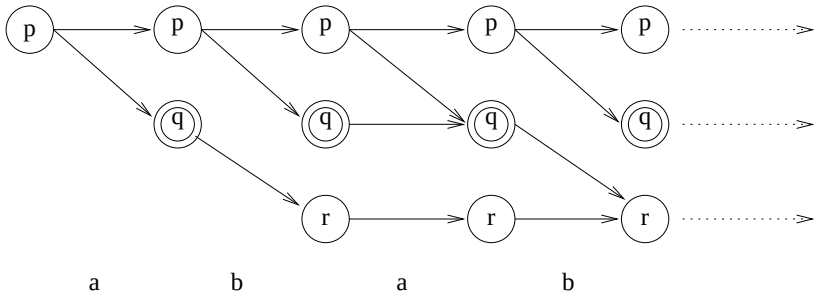
Proof: Let $w \in \Sigma^\omega$. A does not accept w if and only if the automaton does not have a winning strategy in the Büchi game $\mathcal{G}(A, w)$. This happens if and only if the pathfinder has a winning strategy in the game $\mathcal{G}(A, w)$. That is, if and only if, the pathfinder has a strategy

that ensures that every play in $\mathcal{G}(A, w)$ visits F only finitely often. But the game $\mathcal{G}(A^d, w)$ is just the game $\mathcal{G}(A, w)$ with the roles of the automaton and the pathfinder interchange. Thus this is equivalent to saying that the automaton has a strategy in the game $\mathcal{G}(A^d, w)$ to ensure that any play consistent with this strategy visits F finitely often. And by the connection between wins and strategies, this happens if and only if the co-Büchi automaton A^d has an accepting run on the word w. ■

Here is a NBA automaton A_f accepting the set of words with finite number of $b's$ and the corresponding Aco-BA A_∞ accepting the set of words with infinitely many bs.



Since the only state exhibiting nondeterminism is p, the correponding dual automaton is one in which this state is a $\forall$ state. Here is an accepting run for this automaton on the word $ababab\ldots$



Notice that every path through this DAG visits q only finitely often (as a matter of fact, no path visits q more than twice).

To obtain a NBA accepting $\overline{L(A)}$ from a NBA A, we need to tranform the Aco-BA A' to an equivalent NBA.

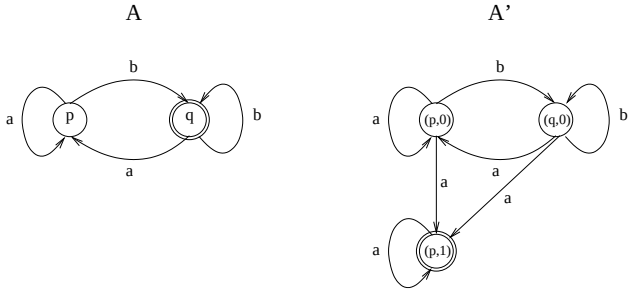
2 Transforming co-Büchi automata into Büchi Automata

How do we transform a Nondeterministic co-Büchi automaton into an equivalent Büchi automaton? For this, let us examine the structure of an accepting run of a Nco-BA . The run looks like $q_1 \xrightarrow{a_1} q_2 \ldots q_{i-1} \xrightarrow{a_{i-1}} q_i \xrightarrow{a_i} \ldots$, where $q_j \notin F$ for all $j \geq i$ for some i. The run decomposes into two parts, an initial finite fragment and an infinite suffix which stays entirely within $Q \setminus F$. Thus, we simply have to nondeterministically guess a position and verify that F is not visited beyond that point.

Can we translate this into a Büchi condition? We need to transform a property of the form $\exists x.\forall y > x.\phi(y)$ to a property of the form $\exists^{\infty} y.\phi(y)$ (i.e. there exists infinitely many ys such that ...).

This is quite easy to arrange. Make two copies of the automaton, in the second copy delete all the states in F. The automaton begins in the first copy and nondeterministically moves to the second copy. This automaton visits states in the second copy infinitely often if and only if beyond a point it stays entirely within the second copy. Thus by treating every state in the second copy as an accepting state (recall that the second copy only has the states $Q \setminus F$) we get an equivalent Büchi automaton.

Here is a co-Büchi automaton A and its equivalent Büchi automaton A':



Moreover, observe that the automaton A' when considered as a co-Büchi automaton with the set of states in the first copy as the accepting set accepts $L(A)$. This is because of the special structure of this automaton. A run either visits the first copy infinitely often and the second copy finitely often or visits the first copy finitely often and second copy infinitely often. So saying "finitely often" about one copy is the same as saying "infinitely often" about the other copy.

Theorem 3 *Let $A = (Q, \Sigma, \delta, s, F)$ be a nondeterministic co-Büchi automaton. Then, the Büchi automaton $A' = (Q', \Sigma, \delta', (s,0), Q \times \{1\})$ where*

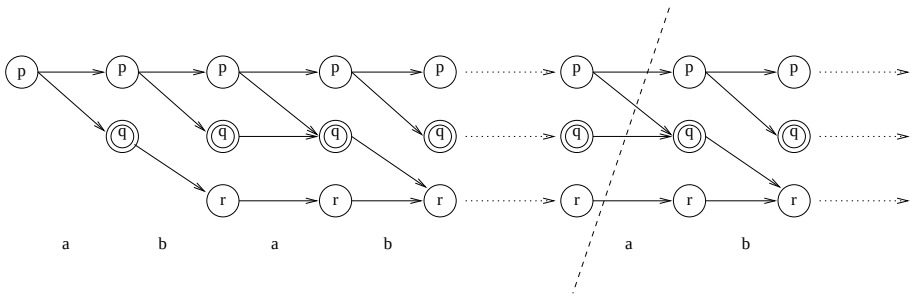
$$\begin{array}{lcl} Q' & = & Q \times \{0\} \ \cup \ (Q \setminus F) \times \{1\} \\ \delta((q,0),a) & = & \delta(q,a) \times \{0,1\} \cap Q' \\ \delta((q,1),a) & = & \delta(q,a) \times \{1\} \cap Q' \end{array}$$

accepts the same language as A. Further the co-Büchi automaton $(Q', \Sigma, \delta, (s,0), Q \times \{0\})$ also accepts the same language.

Exercise: Write down a formal proof of Theorem **??**.

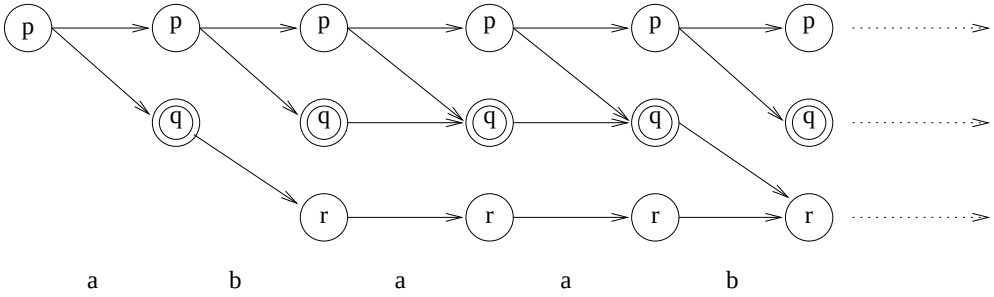
2.1 Alternating co-Büchi automata to Büchi automata

Can we lift the above argument to work for Alternating co-Büchi automata? Let us consider the Aco-BA A_∞ described in the previous section and its run on *ababab*...

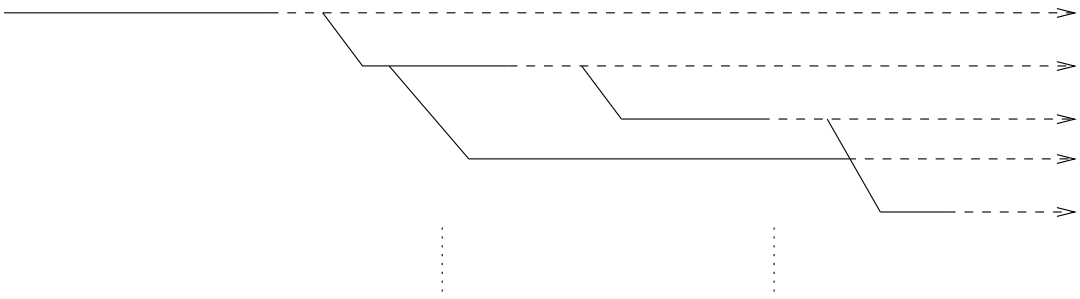


No matter where we slice this DAG the suffix always contains states in F. Thus, a simple two copy idea like the one used for nondeterministic co-Büchi automata will not work here.

Well, in this run no path visits an accepting state more than twice. If we try go generalize this to claim that we may bound the number of visits to accepting states visited along any path in an accepting run of a Aco-BA we shall fail miserably. Consider a run of A_∞ on $abaabaaabaaaab\ldots$ and you will see that for each i there is a path in this DAG that visits the accepting state q at least i times. Yet, there is no path with infinitely many visits to q.



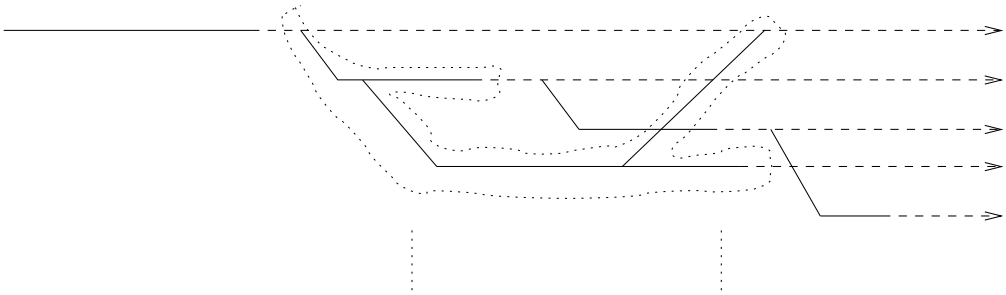
This reason why we cannot slice the run into two with the suffix staying entirely within $Q \setminus F$, is because we are dealing with a DAG and not a path. Even though a path within the DAG may have settled down to staying within $Q \setminus F$, there may be other branches that lead out from this path that visit F. Now, let us consider any such branch. Any extension of such a branch would also settle down to staying within $Q \setminus F$, but once again there may be other branches that lead out that visit final states and so on.



In this figure, the dashed lines correspond to suffices of paths that do not visit states in F. The top line branches after it settles into its $Q \setminus F$ phase to the second line, which in turn branches into the third line after settling into its $Q \setminus F$ phase and so on. We shall call such a branch from a $Q \setminus F$ suffix to a segment that visits F to be a bad branch. Once we enter

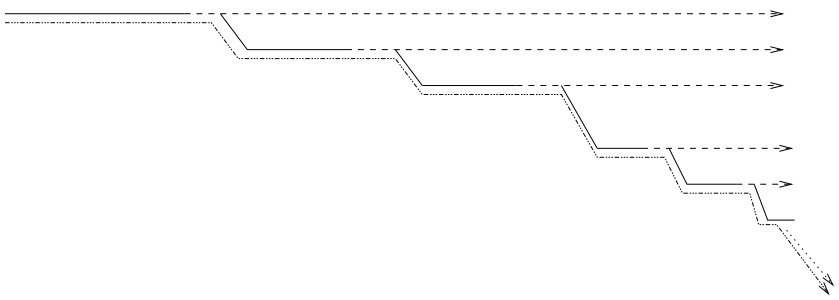
a bad branch (i.e. a segment that visits F) we stay within the SAME bad branch as long as we do not reach a point from where there is a $Q \setminus F$ path leading out.

To be precise: First let us say that a vertex is *good* if there is an infinite path leading out from the vertex that stays entirely within $Q \setminus F$. Thus any good vertex must be in $Q \setminus F$. We call a vertex *bad* if it is not good. The set of good vertices will divide the set of bad vertices into groups and it is these groups that we refer to as bad branches. Two bad vertices belong to the same bad branch if we can reach one from the other without visiting any good vertices, using the edges of the DAG in either direction.



In the above figure there are 3 bad branches and the largest of the three is indicated by an enclosing dotted line.

Can any path through the run DAG visit infinitely many bad branches? Of course not, for if that were the case there would also be an infinite path that that visits F infinitely often. (Well, this needs proof, but take it on faith for the moment!)



So any path visits only a finite number of bad branches. From the assertion that every path visits F finitely often we have moved to the assertion that any path visits finitely many bad branches.

Suppose that there is a bound K for the longest sequence of such branchings among all accepting runs of A. With this assumption I claim that we can now translate the automaton A into an equivalent Büchi automaton that uses $2K + 1$ copies of the A. I shall sketch the construction in words here and a more formal presentation is made a little later.

The idea is to duplicate the construction used in the nondeterministic case, but with $2K + 1$ copies. The copies $0, 2, \ldots 2K$ have an entire copy of the automaton A. The copies $1, 3, 5 \ldots 2K + 1$ have only copies of states in $Q \setminus F$. Transitions are set up so that one could either move at the same level or to the next level (or for that matter any lower level.) Note that now we are dealing with alternating automata, and so the effect of a transition is a set

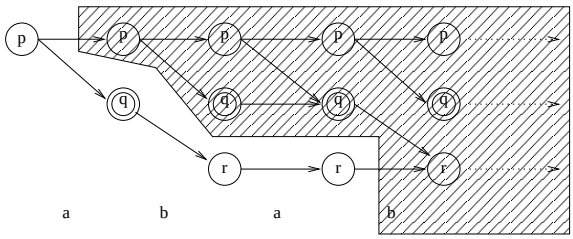
of states (that satisfy $\delta(q, a)$). We are free to choose which subset of this set is to stay at the same level and which subsets move to each of the higher numbered levels.

We start at the state s in level 0. We simulate the accepting run of the alternating automaton in the Büchi automaton by moving down to the next odd level from a even level whenever the current path has an extension that never visits F and moving down from an odd level to the next even level on a branch that visits final states. The fact that the number of bad branchings along any path is bounded by K ensures that we have enough levels to go down. Moreover, since every path eventually visits no more bad branches, it settles down in some odd level. Thus, in the simulation of an accepting run of the Alternating co-Büchi automaton by this alternating automaton, every path in the run settles down in some odd numbered level. On the other hand, in the simulation of a nonaccepting run, there must be at least one path that settles down in an even numbered level (since odd numbered levels do not have any F labelled vertices). Thus, we have constructed a Alternating Büchi automaton (with the set of vertices at odd numbered levels as the accepting set) that accepts the same language as the Alternating co-Büchi automaton we started with.

2.2 The Kupferman-Vardi Construction

Let G be an accepting run (DAG) of the alternating co-Büchi automaton A on some work $a_1 a_2 \ldots a_i \ldots$. We shall associate a number $\mathsf{rank}(v)$ with each vertex v in this DAG. It measures the maximum number of bad branches along any path starting at this vertex. Clearly, if there is a path from v to v' then $\mathsf{rank}(v) \geq \mathsf{rank}(v')$. Since G is a DAG computing such a rank is sort of similar to a topological sorting of this graph starting at the "leaves" and that is what we do. Leaves in our setting will correspond to vertices from where one cannot visit bad branches. These are vertices from where every path stays within $Q - F$.

In any DAG G', for any vertex v we write $\mathcal{I}_{G'}(v)$ to mean the sub-DAG rooted at v.

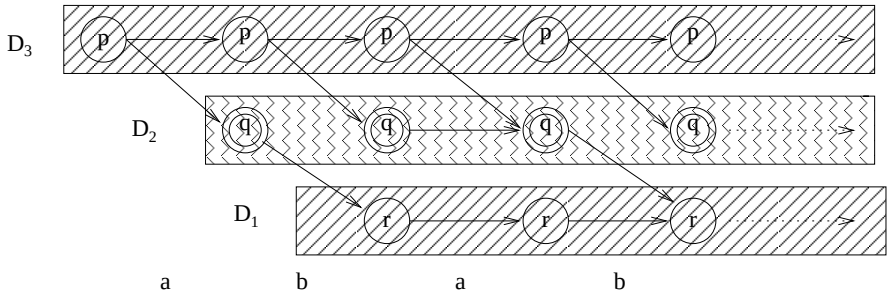


The ideal defined by $(p, 2)$ in the accepting run of A on $abab\ldots$.

We define two sequences of vertices $G_0, G_1, \ldots$ and $D_0, D_1, \ldots$. We shall refer to these sets also as DAGs, to mean the induced sub-DAG on these vertices.

$$
\begin{aligned}
G_0 &= G \\
\\
D_{2i} &= \{v \mid \mathcal{I}_{G_{2i}}(v) \text{ is finite}\} \\
D_{2i+1} &= \{v \mid \mathcal{I}_{G_{2i+1}} \text{ has no vertices from } F\} \\
\\
G_{i+1} &= G_i \setminus D_i
\end{aligned}
$$

Thus, at even numbered levels we remove vertices that have only a finite number of reachable vertices. In odd numbered levels we remove a vertex v if the subDAG rooted at v has no vertices in F. Here is the breakup of the accepting run of A on $abab\ldots$.



Note that any vertex in $D_1, D_3, \ldots$ must be a good vertex. Vertices in $D_2, D_3 \ldots$ are bad vertices. D_0 is an anamoly as it might pick up vertices that have only finitely many successors but none of those successors may be in F. We will assume that D_0 is empty (equivalently that $\delta(q, a) \neq \mathsf{false}$ for any q, a, which can always be arranged).

In D_1 we pick out all the vertices in the DAG for which the subtree below contains no vertices in F. That is, the vertices in the DAG for which any path starting at any of these vertices stays within $Q \setminus F$ and thus visits NO bad branches.

In D_2 we pick up all the vertices that form parts of bad branches which lead into D_1. Thus starting from any such vertex we can visit at the most one bad branch before settling into D_1. In D_3 we pick up good vertices such that any bad branch reachable from these vertices lead into D_1. Thus any path starting at these vertices visits at most one bad branch along the way.

In D_4 we pick out bad vertice that form parts of bad branches that lead into D_3, D_2 and D_1. In D_5 we pick up good vertices that lead into $D_4, D_3, \ldots$. Thus any path leading out from such a vertex may visit at the most two bad branches (one from D_4 and one from D_2) and so on.

Thus, for any vertex in $D_{2i} \cup D_{2i+1}$ we may visit at the most i bad branches along any path. We now show that G_{2n} is empty. Thus every vertex in G belongs to one of $D_1, D_2 \ldots D_{2n-1}$. Thus no path in the run-DAG visits more than n bad branches.

The following facts are quite easy to check:

1. Any path through vertices in D_{2i} is finite. This follows from the definition of D_{2i}.

2. If G_{2i+1} is nonempty then it is infinite. As a matter of fact if $v \in G_{2i+1}$ then there is an infinite path starting at v.

3. If G_{2i} is infinite then G_{2i+1} is nonempty (and hence infinite). This follows from the fact (using König's Lemma) whenever G_{2i} is infinite, it must have an infinite path and all the vertices in this infinite path will appear in G_{2i+1}.

4. For each i if G_{2i+1} is non-empty then D_{2i+1} is nonempty.
   Proof: We know that if G_{2i+1} is nonempty then it is infinite. If D_{2i+1} is empty then

every vertex v in G_{2i+1} has some vertex labelled by F in $\mathcal{I}_{G_{2i+1}}(v)$. It is quite trivial to conclude via König's Lemma that then there must be a path visiting F labelled vertices infinitely often in G_{2i+1}. This contradicts the fact that we started with an accepting run. ■

5. If $v \in D_{2i+1}$ then every node in $\mathcal{I}_{G_{2(i+1)}}(v)$ is in D_{2i+1}.

Let the *width* of a level in a DAG be the number of vertices at that level.

Lemma 4 *For each i, there is an N_i such that for each $j \geq N_i$ the width of level j in G_{2i} is bounded by $n - i$.*

Proof: By induction on i. The base case with $i = 0$ is trivially true. Suppose the result holds for G_{2i}. If this graph is finite then G_{2i+1} and G_{2i+2} are both empty and the result follows. Otherwise, this graph is infinite, which means G_{2i+1} is infinite and thus D_{2i+1} is non-empty. Thus, by 2 and 5 above, there is an N such that at least one vertex from all levels above N belongs to D_{2i+1}. Thus, there is an $N_{i+1} = Max(N, N_i)$ such that the width of all levels above N_{i+1} is at the most $n - i - 1$. ■

Thus, $G_{2n} = \emptyset$. We set $\mathsf{rank}(v) = i$ if $v \in D_i$. Thus if $\mathsf{rank}(v) = m$ then any path starting at v visits at the most $m/2$ bad branches. Moreover, $\mathsf{rank}(v) \leq 2n$ for any $v \in G$.

2.3 Constructing the equivalent alternating Büchi automaton

The automaton A' has $2n + 1$ copies, numbered $0, 1, \ldots 2n$, of the automaton A.

Let $A' = (Q', \Sigma, \delta', (s, 2n), F')$ where

$$\begin{array}{lcl} Q' & = & Q \times \{0, 2, 4, \ldots, 2n\} \cup (Q \setminus F) \times \{1, 3, \ldots, 2n-1\} \\ F' & = & (Q \setminus F) \times \{1, 3, \ldots, 2n-1\} \\ \delta'((q,i),a) & = & \delta(q,a) \otimes \{i, i-1, \ldots, 0\} \end{array}$$

where $\phi \otimes \{i, i-1, \ldots 0\}$ is the formula obtained by replacing each occurance of any state p in ϕ by $\bigvee_{j \geq i}(p, i)$.

Note the special structure of the transitions: They simulate the transitions of A, however one is free to choose to move to a lower numbered copy. Moreover, transitions either connect states in the same copy or go from a higher numbered copy to a lower numbered copy. In particular, there is no way a run can move from a lower numbered copy to a higher numbered copy. Thus, any path must eventually settle down in one of the copies $\{0, 1, \ldots, 2n\}$ (i.e. In any path, there is an infinite suffix that stays within one copy.)

Given an accepting run ρ of the Alternating co-Büchi automaton A on a word w, by computing the rank of each vertex v and labelling it by $(\rho(v), \mathsf{rank}(v))$ we get a run of the Alternating Büchi automaton A' on w. We claim that this run is also accepting. This is because, if some path settles down in some even numbered copy $2i$ that means that there is an infinite path in G of vertices in D_{2i} which is not possible. So it must settle down in some odd numbered level. Thus this run is accepting.

Further, given any run of A' on w, by simply erasing the second component, we get a run of A on w. Moreover, if this run was accepting then every path settles down in some fixed odd numbered level and thus every path stays within $Q \setminus F$ eventually. Thus we get an accepting run of A. Hence $L(A) = L(A')$. Thus we have the following theorem:

Theorem 5 *(Kupferman–Vardi) Let A be a alternating co-Büchi automaton with n states. Then the alternating Büchi automaton constructed above, with $O(n^2)$ states, accepts the same language.*

Since every run of A' eventually settles down in some level or the other it follows that the alternating co-Büchi automaton $A'' = (Q', \Sigma, \delta', (s, 2n), Q' \setminus F')$ accepts the same language as A and A'.

Definition 6 *(Muller-Schupp) An alternating automaton $A = (Q, \Sigma, \delta, s, F)$ is said to be a* weak alternating automaton *if there is a partial order $(P, <)$ and a map $\mathsf{rank} : Q \to P$ such that every state p that appears in $\delta(q, a)$, $\mathsf{rank}(p) \leq \mathsf{rank}(q)$ and $F = \lambda^{-1}(X)$ for some subset X of P.*

The construction described above has thus translated an Alternating co-Büchi automaton into an equivalent weak alternating automaton.

It is quite trivial to observe that the weak alternating Büchi automaton $A = (Q, \Sigma, \delta, s, F)$ accepts the same language as the weak alternating co-Büchi automaton $A = (Q, \Sigma, \delta, s, Q \setminus F)$. Thus, we have identified a subclass of alternating automata for which the translation from the co-Büchi acceptance to Büchi acceptance is trivial, and shown how to translate any alternating co-Büchi automaton into this class.

Putting together the theorems **??** and **??** we see that every alternating co-Büchi automaton can be translated to an equivalent nondeterministic Büchi automaton A'' of size $2^{O(n^2)}$ (its state space is $\{(X, Y) \mid Y \subseteq X \subseteq Q'\}$). Hence, we can complement any Büchi automaton with a state space blowup of at most $2^{O(n^2)}$.

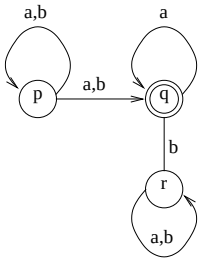
A simple observation will allow us to reduce this complexity to $2^{O(nlogn)}$. The accepting run of A' constructed from an accepting run of A (described above) has a special property: At any level there is at the most one copy of any state. Thus, if X is any state reached in the level by level simulation of this automaton and $(p, i) \in X$ for some $p \in Q$, then $(p, j) \notin X$ for any $j \neq i$. Thus, we can restrict the first component of the states of A'' to be $\{X \subseteq Q' \mid \forall p.\ ((p, j) \in X \wedge (p, k) \in X) \Rightarrow j = k\}$. What is the size of this set? This size of this set is bounded by the product of the number of subsets of Q' ($2^{O(n)}$) and the number of maps that assign ranks (in the range $0, 1, \ldots 2n$) to elements of Q ($O(n!)$). That is, $O(2^{O(n)} n^{O(logn)}) = O(2^{O(nlogn)})$. As we shall see in the next lecture this is optimal.

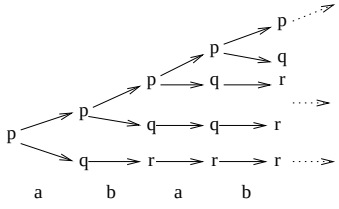
Lecture 11: Safra's Determinization Construction

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 $ababa\ldots$.



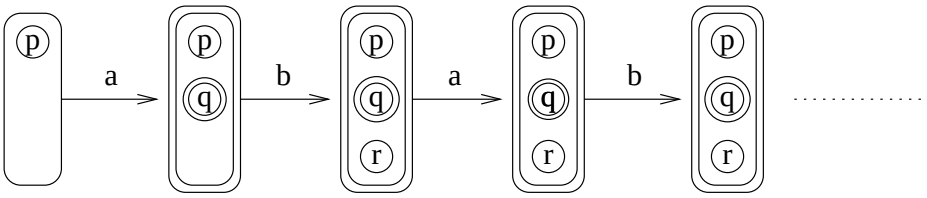


We could deterministically simulate this automaton level by level. That is pretty much all that we can do in a deterministic simulation. But the number of nodes in a level is not bounded. So, we need approximate and maintain some bounded amount of information. In the so called "power-set" construction, 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.

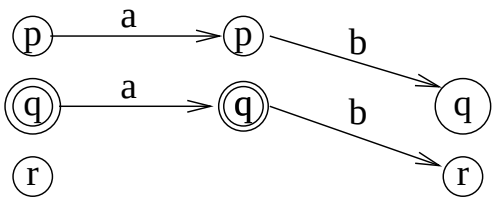
Demanding that that infinitely many levels contain final states is too generous. In the above run, the accepting state F appears in every level in the run tree, however there is no path in this tree that visits F infinitely often (and hence $ababab\ldots$ is not accepted by the above automaton). Thus 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 $S_0 \xrightarrow{a_1} S_2 \xrightarrow{a_2} \ldots$ be the run of the powerset automaton on $a_1 a_2 \ldots$. We accept provided we can decompose this run as $S_0 \xrightarrow{x_1} S_{i_1} \xrightarrow{a_2} S_{i_2} \ldots$ such that for each $q \in S_{i_{k+1}}$, there is a $q' \in S_{i_k}$ such that $q' \xrightarrow{x_{k+1}}_g q$.

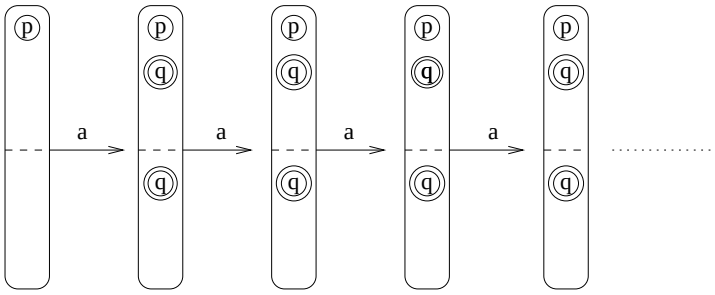
We claim that acceptance under this criterion guarantees that A accepts w. We can see this as follows: Construct a DAG whose nodes at level k are the states in S_{i_k}. An edge is drawn from a node at q' at level i to a node at level $i+1$ if and only if $q' \xrightarrow{x_{i+1}}_g q$. Label this edge 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

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

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.

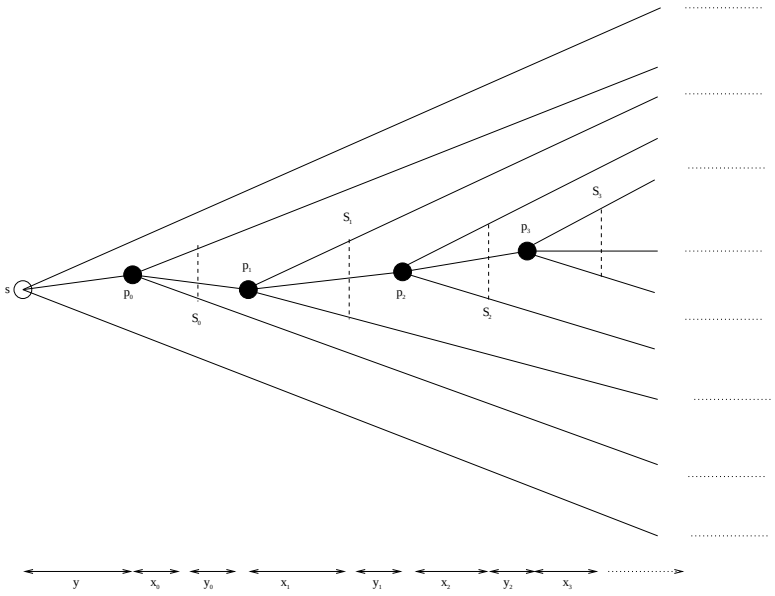
Quite clearly, $L(A_m)$ will not in general be the same as $L(A)$. For instance, the marked powerset automaton constructed from A_f does not accept $aaa\ldots$ which is accepted by A_f.



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.

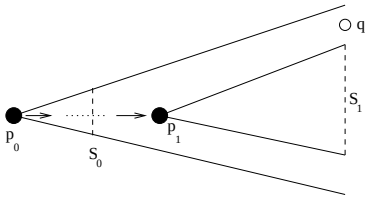
Consider the following figure. It describes a runtree of an NBA A on a word $w = a_1 a_2 \ldots$ which is accepted by A. In the figure, we have marked one accepting path in this runtree, and indicated the states of F that appear along this path by darkened circles.



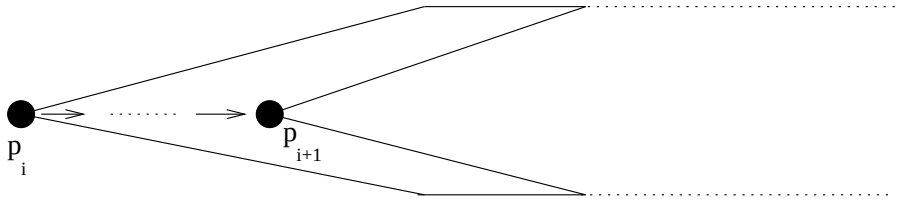
Here $p_0, p_1, \ldots$ are the states from F reached along the marked accepting run. We have placed slices $S_0, S_1, \ldots$ on the sub-runtree rooted at $p_0, p_1, \ldots$. S_0 is the set of nodes reached

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}.)

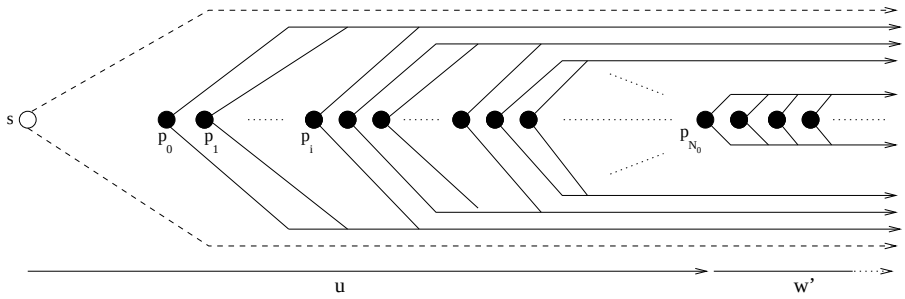
Notice that $S_i \xrightarrow{y_i x_{i+1}}_g S_{i+1}$. This might lead us to believe that we simply have to start the automaton A_m at p_0. Unfortunately, S_1 need not be set of states visited by the automaton A started at state p_0 (on $x_0y_0x_1$), i.e. $\delta_p(p_0, x_0y_0x_1)$ could be a strict 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 $K = 2^{2n} + |F|$.

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

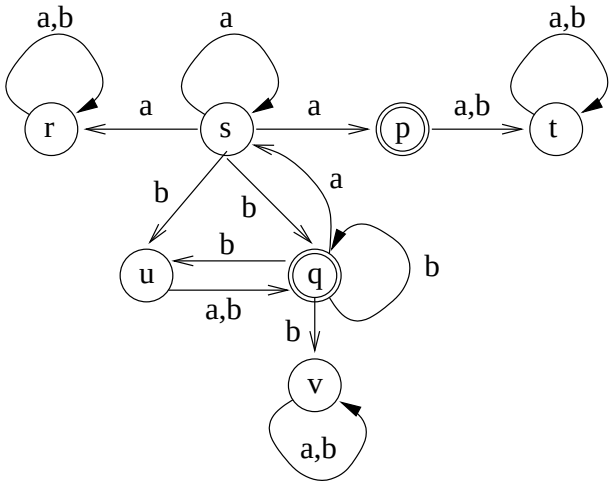
Let $(S, W_1, W_2, \ldots W_K)$ be a state. To compute $\delta_d((S, W_1, W_2 \ldots, W_K), a)$, first compute the tuple $V = (\delta_p(S, a), \delta_m(W_1, a), \delta_m(W_2, a) \ldots \delta_m(W_K, a))$ with the understanding that $\delta_m(\perp, a) = \perp$. Suppose $\{p_1, p_2, \ldots, p_l\} = \delta(S, a) \cap F$. 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 $L(A)$.

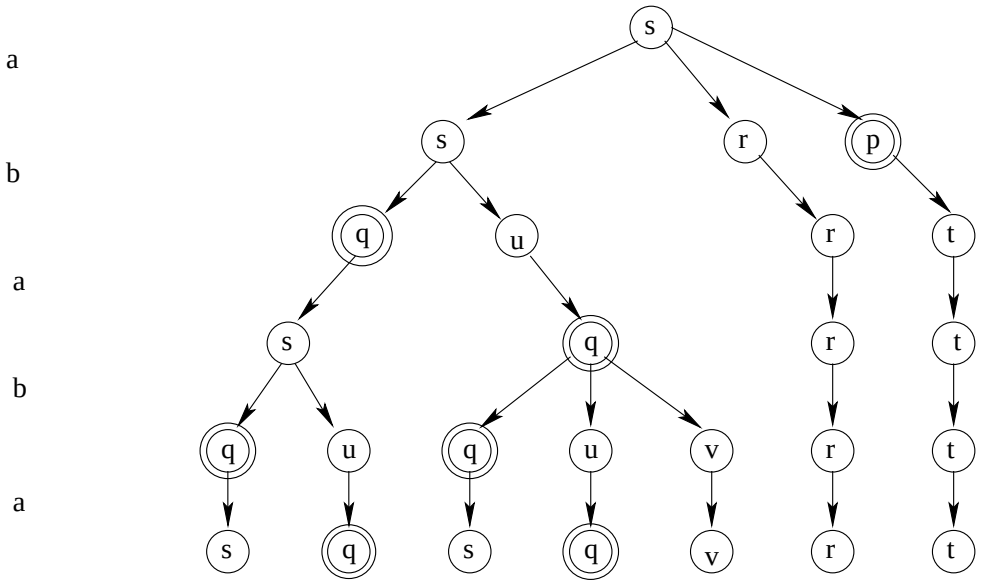
How to transform this double exponential sized automaton to a single exponential sized automaton is the topic of the next section.

1 Safra's Construction

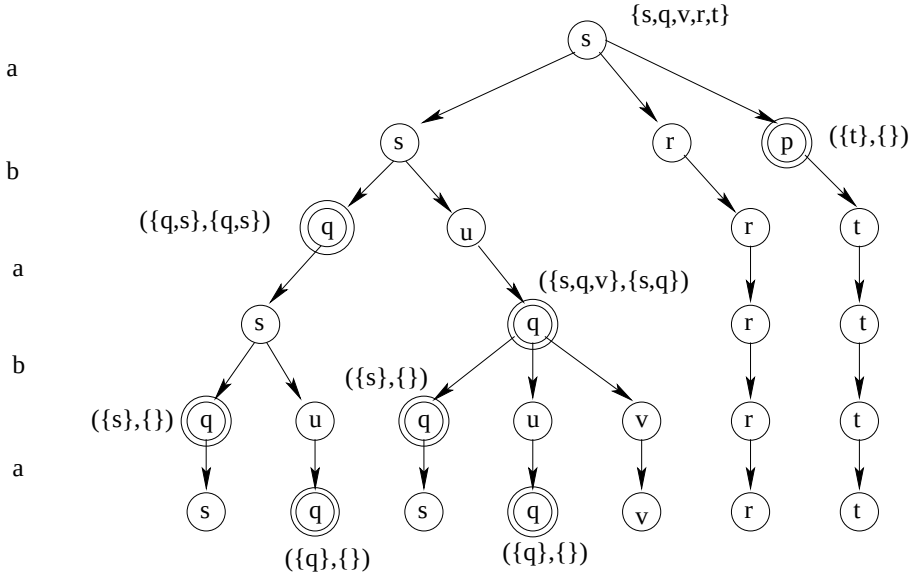
Safra's construction incorporates the structure of the runtree into the state space of the deterministic automaton. We shall illustrate this using the following complex automaton which accepts the set of all words with infinitely many bs:



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.



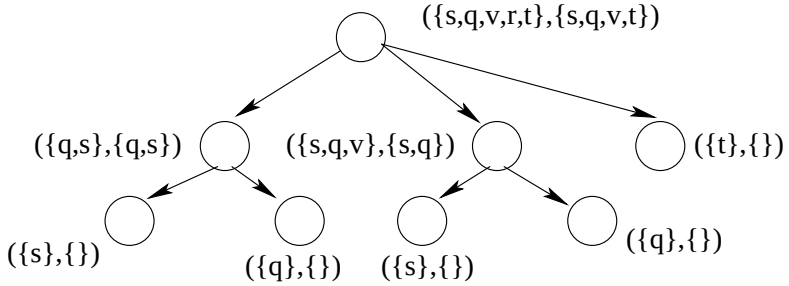
In the autmaton A^d this would have been represented by a tuple

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

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

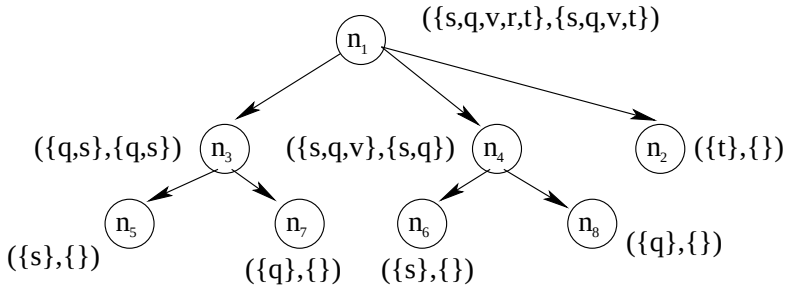
Safra's construction stores more information. It records relationships between these various copies. For example, the three copies corresponding to the p at level 2, and the qs at

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.)

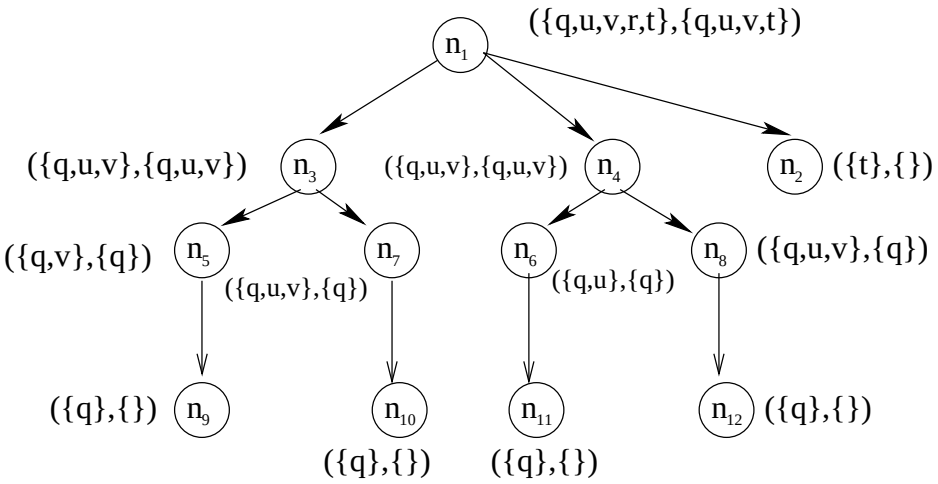
In A_d we maintained the state as a tuple and so the index i allows us to refer to a particular copy of A_m. To do this here, we assume that there is a an infinite ordered set of nodes $n_1 < n_2 \ldots$ and each time we add a node in the tree, we pick the next node from this sequence and use it. For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



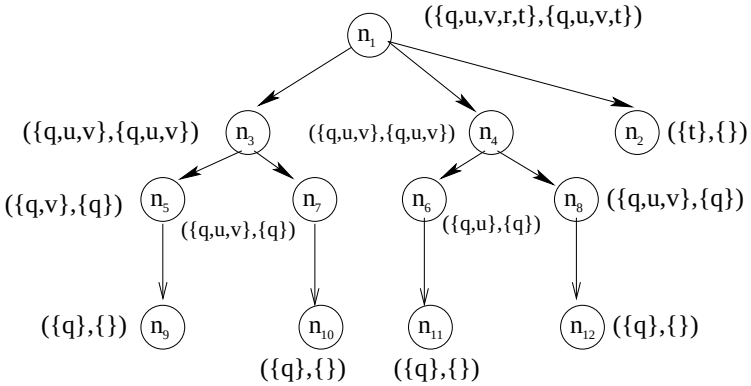
Notice that the nodes are numbered in the order in which the nodes were introduced into the tree (We assume some fixed ordering on Q so when multiple children are added to a node in a single step, the order is determined by the ordering on Q). To understand the behaviour of this automaton let us see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

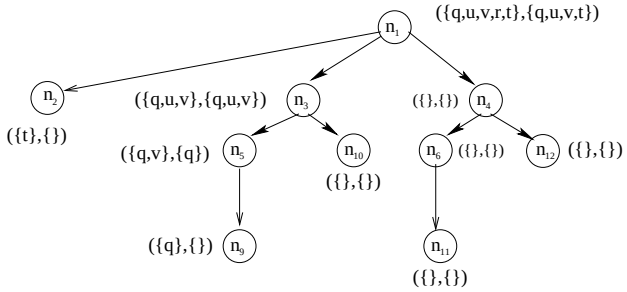
(**Observation 1:** Note that whenever a child of n merges with n, n must be labelled by a state of the form (X, X), i.e. an accepting state of A_m. This is because if a node labelled (X, Y) has children with labels $(X_1, Y_1) \ldots (X_k, Y_k)$ then $Y = X_1 \cup X_2 \ldots X_k$.)

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

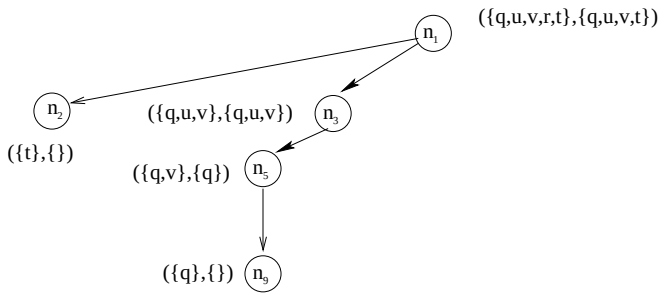
X_j is a strict subset of X_i. We take our set of states to be trees over $n_1, n_2 \ldots$ labelled by states of A_m that further satisfy this property. Thus the trees constituting states have depth bounded by $|Q|$. However, since we have an infinite set of labels the automaton we have constructed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

Lecture 12: Optimality of Safra's Determinization and the Kupferman-Vardi Complementation

I shall begin by presenting the "original" Safra's construction (see [2]). Let $A = (Q, \Sigma, \delta, s, F)$ be a nondeterministic Büchi automaton. We shall tranform this into an equivalent deterministic Rabin automaton A_s. The states of A_s are "Safra trees". A Safra tree is a tree with nodes drawn from the set $\{n_1, n_2, \ldots, \}$ (we shall restrict this to a finite set soon), equipped with a labelling function that labels nodes with subsets of Q and a colouring function that colours the nodes with either white or green. We shall write $\lambda(n)$ to denote the set labelling node n and $c(n)$ to denote the colour of the node n. A Safra tree satisfies the following properties:

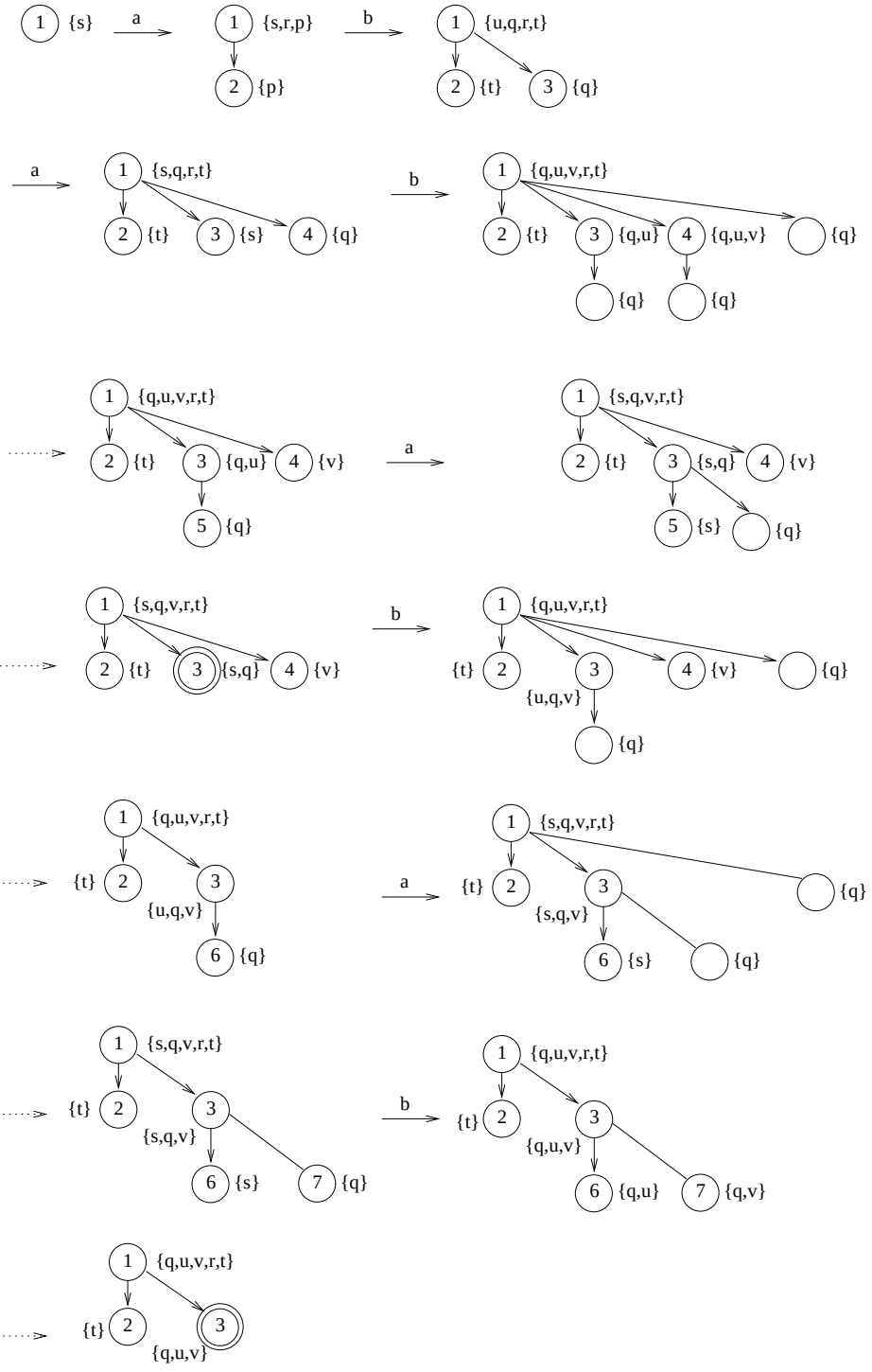
1. If $\{n_1, n_2, \ldots n_k\}$ are the children of n then $\lambda(n)$ is a strict superset of $\bigcup_i \lambda(n_i)$.
2. If n_1 and n_2 are children of some n then $\lambda(n_1) \cap \lambda(n_2) = \emptyset$.

Observation 1: The depth of a Safra tree is bounded by $|Q|$. This follows from item 1 above.
Observation 2: The number of nodes in a Safra tree is bounded by $|Q|$. This can be seen as follows: pick any n in a Safra tree T. We claim that there is a q such that q appears in the sets labelling the nodes on the path from the root to n and no where else. In proof, pick any q that appears in $\lambda(n)$ but not in the labels of any of its children. Such a q must exist by item 1. (Observation 2 will allow us to restrict the set of nodes to the finite set $\{n_1, n_2, \ldots, n_{2|Q|+1}\}$.)

The start state of A_s is the tree with the single node n_1 labelled by the set $\{s\}$. It is coloured green if $s \in F$ and white otherwise. We now describe the transition function. Given T and a letter a we shall construct a Safra tree T' as follows:

1. Every node in n in T is also a node of T'. The label of such a node is $\delta(\lambda(n), a)$ (Thus each node runs its own copy of the powerset automaton A_p instead of a copy of the marked powerset automaton A_m. As observed in the previous lecture, the second component of A_m is available in the state of the children of a node and so it suffices to simulate A_p at each node.)
2. For each $n \in T$, if $\delta(\lambda(n), a) \cap F$ is not empty then we create a new child for n in T' and label it with $\delta(\lambda(n), a) \cap F$. (Instead of forking one copy per accepting state, we simply start a single copy simultaneously from all of those states.)
3. If a state q appears along more than one path starting at the root delete it from all paths except the left most (with the understanding a child inserted earlier appears to the left of a child inserted later.) If in this process any node has the label $\emptyset$ then delete that node.
4. If n is any node with children $m_1, m_2, \ldots m_k$ with $\lambda(n) = \bigcup_i \lambda(m_k)$ then delete $m_1, \ldots m_k$ along with all their descendents and colour n with green. Label other nodes white.

The figure below describes a run of the automaton on page 6 of Lecture 11, on the input *ababab*. The effect of steps 1 and 2 are indicated via $\longrightarrow$ while the effect of steps 3 and 4 are indicated via dotted arrows.



Lemma 1 *A word $w = a_1 a_2 \ldots$ is accepted by A if and only if in the unique run of A_s on w, some node n is coloured green infinitely often.*

Proof: Let an accepting run of A on $a_1 a_2 \ldots$ be $q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} q_3 \ldots$. Let $T_1 \xrightarrow{a_1} T_2 \xrightarrow{a_2} \ldots$ be the run of A_s.

First of all note that if n appears in T without children, $T \xrightarrow{x} T'$ such that the node n is not deleted along this run and further n is coloured green in T' then, $\lambda(n) \xrightarrow{x}_g \lambda'(n)$ (where $\lambda(n)$ is the label of n in T and $\lambda'(n)$ is the label of n in T'.) This follows from the definition of the transition relation of A_s. Also observe that, whenever a node is coloured green it has no children. From this it is quite easy to see that if some node n is coloured green infinitely often along the run of A_s on w and if the sets labelling n at these green coloured stages are $X_1, X_2, \ldots$ then $w = w_1 w_2 \ldots$ with $\{s\} \xrightarrow{w_1}_g X_1 \xrightarrow{w_2}_g X_2 \ldots$. Thus w is accepted by A.

For the converse, we associate a node $N_i \in T_i$ with the accepting run $q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} \ldots$. It is the highest level node (i.e. farthest from the root) whose label carries the state q_i. We claim that $N_i = n$ for some n infinitely often. In proof note that if the associated node turns out to be the root (which appears in all the trees T_i) then there is nothing to prove. Otherwise, after some M_1, the associated node N_i is never the root for all $i \geq M_1$. Let m_1 be the node at level 2 visited after this. Beyond M_1, whenever the associated node is at level 2 it can only be m_1 or any other node at that level that appears to the left of m_1. Thus there are only finitely many choices. If it visits one such node infinitely often, we are done. Otherwise, there is is a point M_2 beyond which the associated node appears at level 2 or more. Continuing this argument and using the fact that the depth of the trees is bounded by $|Q|$ we conclude that there is a n that is hit infinitely often as the associated node.

Now we claim that this node n must be coloured green infinitely often. This follows from the fact that whenever q_i is an accepting state the associated node moves from n to a child of n. Since it returns back to n, there is a point where the children of n are deleted. At this point n must be coloured green.

Thus, the node n is coloured green infinitely often along the run of A_s on w. ■

Now, we can use the fact that the size of the tree is bounded by $|Q|$ to use a finite number of nodes (akin to the argument in Lecture 11, Lecture 8 and so on). We simply use a collection of $2|Q| + 1$ nodes and ensure that whenever a node n is deleted at a step, it does not appear in the tree constructed (as all the new nodes added can be given other labels). As usual, we shall use a Rabin acceptance condition (F_i, I_i) one for each node i. F_i consists of all the trees where i does not appear and I_i consists of all the trees where i appears and is coloured green. It is easy to see that this finite state deterministic Rabin automaton accepts the same language as A_s. Henceforth we shall use A_s to refer to this automaton. What is the size of this automaton? Simply observe that there are at most n^{n-2} trees on n nodes and if we insist on ordered trees we get $n^{n-2} * n!$ and labelling involves picking functions from Q to nodes. Thus the total number of trees is of the order of $n^{n-2} * n! * n^{|Q|}$. But $n = 2 * |Q| + 1$. Thus, the number of trees is of the order of $2^{O(|Q| log(|Q|))}$.

Theorem 2 *(Safra) Any nondeterministic Büchi automaton with n states can be transformed into an equivalent deterministic Rabin automaton of size $O(2^{O(nlogn)})$ with $O(n)$ ac-*

cepting pairs. Any NBA can be complemented into a NBA with a size blowup of at most $O(2^{O(nlogn)})$.

For a proof of the second part use the following exercise.

Exercise: Show that every nondeterministic Streett automaton with n states and r accepting pairs can be tranformed into an equivalent nondeterministic Büchi automaton with $O(n * 2^r)$ states.
[**Hint:** The set of F_is hit infinitely often must be a subset of the set of E_is hit infinitely often. Keep track of the set of indices for which E_i has been hit and the set of indices for which F_i has been hit. If the latter is a subset of the former then reset both sets to the empty set and continue...]

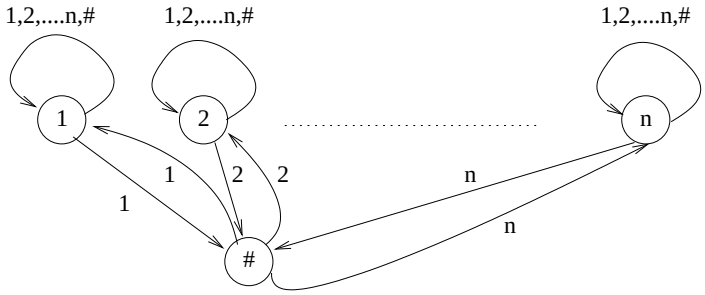
1 A lower bound

In this section we shall show that we cannot hope to improve Safra's construction. This result is due to M. Michel. Our presentation here follows that of C. Löding [1].

The idea is to construct a family of nondeterministic Büchi automata A_i of size n such that any NBA accepting $\overline{L(A_i)}$ must have at least $n!$ states. The automaton $A_n = (\{1, 2, \ldots, n, \#\}, \{1, 2, \ldots, n, \#\}, \delta_n, \{1, 2, \ldots n\}, \{\#\})$. (We have used a set of initial states instead of one. We can add one additional state and use that to replace this set by a single intitial state.)

$$\begin{array}{lcll} \delta(i,j) & = & \{i\} & \text{if } i \neq \# \text{ and } i \neq j \\ \delta(i,i) & = & \{i,\#\} & \text{if } i \neq j \\ \delta(\#,i) & = & \{i\} & \end{array}$$

Here is a pictorial presentation of A_n.



What is the language accepted by this automaton? Say we start at i. We could stay at i for some time reading say x_0, but eventually we have to move to # and this must be on reading i. The very next move takes us out of # to the state j where j is the next input letter. Thus, the input read up to the first trip through the accepting state would be of the form x_0ij for some j. Now, we could consume some input, say x_1 at j but then eventually we would have to make a trip through # and that would require us to read jk for some k

and we would end up at state k and so on. Thus the input read up to the first two trips through the accepting state looks like x_0ijx_1jk. Thus it is the pairs of adjacent letters in the given word that arrange for trips through the accepting set.

Claim 1: Suppose there is a sequence $i_1i_2 \ldots i_k$ of elements from $\{1, 2, \ldots n\}$ such that the words i_1i_2, i_2i_3, ..., i_ki_1 all appear infinitely often in w then w is accepted by A_n.
Proof: Start at i_1. At the first i_1i_2 travel through # to reach i_2, then at the next i_2i_3 travel through # to i_3 and so on. This ensures that # is visited infinitely often. ■

Claim 2: If w is accepted by A_n then there is a sequence $i_1i_2 \ldots i_k$ of elements from $\{1, 2, \ldots n\}$ such that the words i_1i_2, i_2i_3, ..., i_ki_1 appear infinitely often in w.
Proof: Let ρ be an accepting run on w. Suppose j_1 is a state from $\{1, 2, \ldots, \#\}$ that appears infinitely often along this run. The run must go from j_1 to # infinitely often. Let j_2 be a state that is reached (in two steps) via # from j_1 infinitely often. Similarly, let j_3 be a state reached from j_2 via # infinitely often and so on. Since there are only finitely many possibilities, some $j_l = j_{l+k}$ for some l and k. Let $i_1, i_2 \ldots i_k$ be the sequence $j_l, j_{l+1}, \ldots j_{l+k-1}$. ■

Thus we have a characterization of the language accepted by this automaton: Let $IP(w) = \{(i, j) \mid 1 \leq i, j \leq n$ and the word ij appears infinitely often in $w\}$. Then, w is accepted if and only if the directed graph on $\{1, 2, \ldots, n\}$ with edge set $IP(w)$ has a cycle. This fact is used in the proof of the following theorem of Löding, that generalizes an earlier theorem of Michel.

Theorem 3 *(C. Löding) Any Streett automaton $A = (Q, \Sigma, \delta, s, ((E_1, F_1), (E_2, F_2) \ldots (E_k, F_k)))$ accepting $\overline{L(A_n)}$ must have at least $n!$ states.*

Proof: Pick any two distinct permutations $i_1i_2 \ldots i_n$ and $j_1j_2 \ldots j_n$ of $\{1, 2, \ldots, n\}$. The words $w_1 = (i_1i_2 \ldots i_n\#)^\omega$ and $w_2 = (j_1j_2 \ldots j_n\#)^\omega$ are not in $L(A_n)$ since $IP(w_1)$ and $IP(w_2)$ have no cycles. Let I_1 (I_2) be the set of states hit infinitely often along some accepting run of A on w_1 (w_2). We shall show that $I_1 \cap I_2$ is empty.

Suppose $q \in I_1 \cap I_2$. Then, following the run on w_1 we can construct a word $x_1 = ui_1i_2 \ldots i_nv$ such that $q \xrightarrow{x_1} q$ visiting exactly the set of states I_1. Similarly, following the run on the word w_2 we can construct a word $x_2 = u'j_1j_2 \ldots j_nv'$ such that $q \xrightarrow{x_2} q$ visiting exactly the set of states I_2. Thus there is a run $s \xrightarrow{y} q \xrightarrow{x_1} q \xrightarrow{x_2} q \xrightarrow{x_1} q \xrightarrow{x_2} q$ that visits precisely the set of states $I_1 \cup I_2$ infinitely often. This run is also an accepting run! This follows from the following exercise:

Exercise: Show that if the set X as well as the set Y satisfy the Streett condition $((E_1, F_1), \ldots (E_k, F_k))$ then $X \cup Y$ also satisfies this Streett condition.

However, $IP(y(x_1x_2)^\omega)$ has a cycle: Let k be the least number so that $i_k \neq j_k$. Then j_k is some i_l for $l > k$. Similarly, $i_k = j_m$ for some $m > k$. Then $i_ki_{k+1} \ldots i_lj_{k+1} \ldots j_{m-1}$ forms a cycle. This contradicts the fact that $L(A) = \overline{L(A_n)}$. Thus, $I_1 \cap I_2$ is empty. Thus, the total number of states in A is at least $n!$. ■

Thus, Safra's construction gives an optimal way to transform a NBA into an equivalent Rabin automaton.

Corollary 4 *Any nondeterministic Büchi automaton accepting* $\overline{L(A_n)}$ *has at least* $n!$ *states.*

Proof: A NBA $(Q, \Sigma, \delta, s, F)$ is also the Street automaton $(Q, \Sigma, \delta, s, ((F, Q)))$. ■

Thus, Safras construction as well as the Kupferman-Vardi construction for complementing NBAs are optimal.

2 Transforming acceptance conditions

In this section we shall try to see how to translate automata of one kind into another. Let us recall what we know:

1. By Lemma 5 of Lecture 8, any nondeterministic Müller automaton of size n with m accepting sets can be transformed into a NBA of size $O(m.n)$.

2. Quite trivially, a NRA automaton can be transformed into a NMA with the same number of states. The number of accepting sets however is $O(2^n)$ where n is the number of states of the NRA. This construction transforms deterministic automata into deterministic automata.

3. The same as above for Streett automata.

4. NBAs can be transformed into DRA of size $2^{O(nlogn)}$ and $O(n)$ accepting pairs. This follows from Safra's construction.

5. NBAs can be transformed into DMA with $2^{O(nlogn)}$ states (and double exponential number of accepting sets). This follows from items 2 and 4.

6. By the exercise at the end of Safra's construction in this lecture, any nondeterministic Streett automaton of size n with r accepting pairs can be transformed into an equivalent NBA of size $O(n.2^r)$.

7. A NRA with n states and r accepting pairs can be transformed into an equivalent NBA of size $O(n.r)$.
   Proof: The Büchi automaton, guesses an i and checks whether the word is accepted via the accepting pair (E_i, F_i). This is done by "waiting" for the point from where no state from E_i appears and then verifying that this is indeed the case and also that some state from F_i is visited infinitely often. ■

But here are some transformations that are missing in the list above:

1. Transforming Müller automata into Streett/Rabin automata.

2. Transformations between deterministic Streett and Rabin automata. Note that the last two items in the above list allow transformations between NSA and NRA (since every Büchi automaton can be thought of as a Streett or as a Rabin automaton with one accepting pair). We can combine the $O(n.2^r)$ transformation from Streett to NBA with Safra's construction to get a $O(2^{O(n.2^r log(n.2^r)})$ translation from NSAs to DRAs. But notice that this is double-exponential in the number of accepting pairs. As we shall see, we can do better.

3 Transforming Müller automata into Streett/Rabin automata

This is a rather ingenius construction that will be used again and again. The original ideas behind this construction appear in the work of J.R.Büchi. Our presentation will follow that of Wolfgang Thomas [3].

The idea is to keep track of the history of the states visited. Let A be a Müller automaton with n states. Suppose a run of the NMA automaton on input $a_1a_2 \ldots a_k \ldots$ is $\rho = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \ldots \xrightarrow{a_k} q_k \ldots$. The history after the first k moves is $q_0q_1 \ldots q_k$. Recall that the run is accepting if there is an $X \in \mathcal{F}$, such that $inf(\rho) = X$. This happens if

1. Every state in X is visited infinitely often.

2. There is a N such that for all $i \geq N$ the last $|X|$ distinct states in $q_0q_1 \ldots q_i$ is the set X. The last k distinct states in $q_0q_1 \ldots q_i$ is obtained by removing duplicate occurances of any state by deleting all but the right most copy of each state. For example the last 3 distinct states in $pqrsppqqrssspppp$ are $\{p, s, r\}$ as after deleting the duplicate entries we get $qrsp$. This permutation of (some subset of) Q obtained by deleting all but the right most occurance of each state in a state sequence σ is called the LAR (Last Appearance Record) of σ, written $\mathsf{LAR}(\sigma)$.

(Note that condition 2 alone is not enough. The last 3 distinct states of of $r(pq)^i$ is $\{p, q, r\}$ for all i. Thus condition 2 does not ensure that states in X are hit infinitely often.)

This suggests that we construct an automaton whose states are LARs. Our LARs will be permutations Q (that is, every state in Q will appear). Note that $\mathsf{LAR}(q_0q_1q_2 \ldots q_{i+1}) = \mathsf{LAR}(\mathsf{LAR}(q_0q_1 \ldots q_i).q_{i+1})$. This leads us naturally to the following definition for the transition relation: Given a LAR $q_1q_2 \ldots q_n$ and a letter a, we may move to a state $p_1p_2 \ldots p_n$ if $p_1p_2 \ldots p_n = \mathsf{LAR}(q_1q_2 \ldots q_np)$ where, $p \in \delta(q_n, a)$. Notice that $p_1p_2 \ldots p_n = q_1 \ldots q_{i-1}q_{i+1} \ldots q_np$ when $q_i = p$. The effect of such a transition is to delete p from the current list and append it at the right end. We shall say that *position i is hit* in a transition to mean that the ith state was moved to the right end by this transition. Thus, if $q_0 \xrightarrow{a_1} q_1 \xrightarrow{\cdots}$ is a run in Q then the LAR automaton simulates this run via the run $\mathsf{LAR}(q_0) \xrightarrow{a_1} \mathsf{LAR}(q_0q_1) \xrightarrow{a_2} \mathsf{LAR}(q_0q_1q_2) \ldots$. (Well, we are fudging things here. Some of these are not really permutations of Q. Well, we pad it to the left to get a permutation. I leave it to the reader to figure out what to do.)

For a fixed $X \in \mathcal{F}$, if we use conditions 1 and 2 to describe acceptance in the LAR automaton then we get something that is almost a Rabin accepting pair. Condition 2 says that LARs in which the last $|X|$ entries include states from $Q \setminus X$ are visited only finitely often. Thus, condition 2 is of the form "E_i is hit finitely often". Condition 1 says that every state in X is visited infinitely often. Condition 1 is not quite equivalent to saying "F_i is hit infinitely often ". We shall fix that now. The trick is to record at each move the position from which a state was deleted (and moved to the right end).

Henceforth by an LAR we shall refer to a pair consisting of a permutation of the states of Q and an index i with $1 \leq i \leq n$. Given $(q_1q_2 \ldots q_n, i)$ and a letter a, we allow a transition $(q_1q_2 \ldots q_n, i) \xrightarrow{a} (p_1p_2 \ldots p_n, j)$ if $q_n \xrightarrow{a} p$ and $q_j = p$ and $p_1p_2 \ldots p_n = q_1 \ldots q_{j-1}q_{j+1} \ldots q_np$. A run $q_0 \xrightarrow{a_1} q_1 \xrightarrow{q_2} \ldots$ is simulated by this automaton via a run of the form $(f_1, i_1)(f_2, i_2) \ldots$ where each f_i is a permutation of Q. We write $f_i(j)$ for the jth element of the permutation f_i. Then $f_i(n) = q_i$ for all $i \geq 1$ and further $f_i = \mathsf{LAR}(q_1q_2 \ldots q_i)$.

For each $X_i \in \mathcal{F}$ we shall define a Rabin accepting pair (E_i, F_i) as follows: E_i is the set of states $(q_1q_2 \ldots q_n, j)$ such that the last $|X_i|$ states of $q_1q_2 \ldots q_n$ do not form the set X_i. Thus if E_i is hit finitely often along some run then there is an N such that for all $k \geq N$, the state reached along this run is of the form (f_k, j) where the last $|X_i|$ elements of f_k form the set X_i.

F_i consists of all tuples of the form $(f, n - |X_i| + 1)$. Suppose E_i is hit finitely often and F_i is hit infinitely often. Let $x \in X_i$. Pick any state (f_k, j) in the run with $k > N$ (where N is as defined in the previous paragraph). Therefore x appears somewhere among the positions $f_k(n - |X_i| + 1), f_k(n - |X_i| + 2) \ldots f_k(n)$. Notice that when the position j is hit in a transition, states that appear at positions $j + 1$ through n shift to the left by one position. The only way in which a state shifts right is when it is visited and in that case it is moved to the right end of the LAR. Since position $n - |X| + 1$ is hit infinitely often, at each such hit x is shifted to the left by one step. And if it is not visited then it will eventually reach the position $n - |X_i| + 1$. But then we are guaranteed to visit x as this position is hit sometime after this. Thus for all $k \geq N$ there is a $k' > k$ such that x is visited at step k'. Thus each state of X_i is visited infinitely often.

It is easy to check that the converse, that is if $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \ldots$ accepts via X_i then the corresponding run $(f_0, i_0) \xrightarrow{a_1} (f_1, i_1) \ldots$ satisfies the Rabin condition (E_i, F_i). Thus, the LAR automaton accepts via the pair (E_i, F_i) if and only if the original automaton accepted via the set X_i. Thus we have transformed a Müller automaton into an equivalent Rabin automaton. Moreover this transformation sends deterministic automata to deterministic automata. This also allows us to translate deterministic Müller automata to deterministic Streett automata (simply complement, transform to Rabin and complement again). The number of states is $O(n!)$ and the number of accepting pairs is $O(m)$ where m is the number of sets in $\mathcal{F}$.

We shall now modify this construction to make the number of accepting pairs independent of m (and dependent only on n). At present, E_i ensures requirement 2 while F_i ensures requirement 1 (assuming E_i is hit finitely often). The idea is to redistribute this work.

Let E_i consist of all the pairs of the form (f, j) where $j <= n - |X_i|$. Let F_i consists

of all the pairs of the form $(f, n - |X_i| + 1)$ where the last $|X_i|$ states in F_i are the states of X_i. Now, if E_i is hit finitely often along some run, then after some point the positions $1, 2, \ldots n - |X_i|$ are not hit. If further F_i is hit infinitely often then there must a later point at which the last $|X_i|$ states form the set X_i. From here on, since the hit never reaches positions $1, \ldots, n - |X_i|$, it is guaranteed that the last $|X_i|$ states will constitute the set X_i. More over, position $n - |X_i| + 1$ is hit infinitely often. Thus, the new Rabin condition (E_i, F_i) also guarantees that the set of states of A hit infinitely often is X_i.

Now, if $|X_i| = |X_j|$ then $E_i = E_j$. Can we combine the sets F_i and F_j together? Yes, because $F_i \cup F_j$ is hit infinitely often if and only if one of F_i or F_j is hit infinitely often. Thus, we can replace (E_i, F_i) and (E_j, F_j) with $(E_i = E_j, F_i \cup F_j)$. Thus, we can obtain a Rabin automaton which has at the most n accepting pairs. Thus, every Müller automaton with n states can be transformed into a Rabin automaton with $O(n!)$ states and $O(n)$ accepting pairs.

Theorem 5 *Let $A = (Q, \Sigma, \delta, s, \mathcal{F})$ be a Müller automaton with n states. Then, there is a Rabin automaton A' accepting the same language as A with $O(n!)$ states and $O(n)$ accepting pairs.*

Proof: Let $Q' = \mathsf{Perm}(Q) \times \{1, 2, \ldots n\}$. Let $s' = (f, n)$ for some fixed permutation f with $f(n) = s$. The transition relation is given by

$$((q_1 q_2 \ldots q_n, i) \xrightarrow{a} (q_1 q_2 \ldots q_{j-1} q_{j+1} \ldots q_n q_j, j) \quad \Longleftrightarrow \quad (q_n, a, q_j) \in \delta$$

The accepting condition consists of a family of pairs $(E_1, F_1), (E_2, F_2) \ldots (E_n, F_n)$ where:

$$\begin{array}{rcl} E_i & = & \{(f, j) \mid j \leq i\} \\ F_i & = & \{(f, i) \mid \{f(i+1), f(i+2) \ldots, f(n)\} \in \mathcal{F}\} \end{array}$$

The correctness of this construction follows from the discussion above. ■

We shall not stop here, but modify this transformation to get a Rabin automaton with a very interesting structure.

3.1 Rabin Chain/Parity Automata

Observe that in any Rabin automaton we may replace the list of accepting pairs

$$(E_1, F_1), (E_2, F_2), \ldots, (E_k, F_k)$$

with

$$(E_1, E_1 \cup F_1), (E_2, E_2 \cup F_2) \ldots (E_k, E_k \cup F_k)$$

without changing the language accepted. In other words, we may always ensure that E is a subset of F in each accepting pair (E, F). Thus, we may replace the accepting pairs used in the proof of theorem 5 by the following:

$$\begin{array}{rcl} E_i & = & \{(f, j) \mid j \leq i\} \\ F_i & = & E_i \cup \{(f, i) \mid \{f(i+1), f(i+2) \ldots, f(n)\} \in \mathcal{F}\} \end{array}$$

Well, this gives us more — we also have $F_i \subseteq E_{i+1}$ for $1 \leq i < n$. Thus the family of Rabin accepting pairs actually form a chain $E_1 \subseteq F_1 \subseteq E_2 \subseteq F_2 \ldots \subseteq E_n \subseteq F_n$. A Rabin condition with this property is called a *Rabin Chain condition.* ,Thus, we have the following theorem:

Theorem 6 *Any Müller automaton can be transformed into an equivalent Rabin Chain automaton with $O(n!)$ states and n accepting pairs.*

Rabin chain automata are also Streett automata! Let A be a Rabin chain automaton with accepting pairs $(E_1, F_1), (E_2, F_2), \ldots, (E_k, F_k)$. Now consider the automaton A' obtained from A be replacing the list of accepting pairs by $(\emptyset, E_1), (F_1, E_2), \ldots, (F_{k-1}, E_k), (F_k, Q)$. A run is accepting in A if and only if it is rejecting in A'. This gives us the following results:

Theorem 7 *Deterministic Rabin chain automata can be complemented without any blow up in state space or accepting pairs.*

In proof, note that the automaton A' accepts the complement.

Theorem 8 *Any Müller automaton can be transformed into an equivalent Street (chain) automaton with $O(n!)$ states and $n+1$ accepting pairs.*

In proof note, that A' as a Street automaton accepts the same set of words as A accepts as a Rabin automaton.

Parity Automata Here is a alternative way of looking at a Rabin chain condition $E_1 \subseteq F_1 \subseteq E_2 \subseteq F_2 \ldots \subseteq E_k \subseteq F_k$. We shall assume without loss of generality that $F_k = Q$. (Otherwise we may add an additional pair (Q, Q) without altering the language accepted and maintaining the chain property.) Thus we have a chain of $2k$ sets:

E_1	$\subseteq$	F_1	$\subseteq$	E_2	$\subseteq$	F_2	$\subseteq$	$\ldots$	$\subseteq$	E_k	$\subseteq$	F_k
1		2		3		4		$\ldots$		$2k-1$		$2k$

The index of the set E_i in this list is $2i-1$ while that of F_i is $2i$. With each $q \in Q$, we assign a number $c(q)$ where $c(q)$ is the index of the first set (from the left) in which it appears.

A run ρ is accepted under this Rabin (chain) condition if and only if there is some i such that E_i is hit finitely often (and hence $E_1, F_1, E_2, \ldots, F_{i-1}$ are hit finitely often) and F_i is hit infinitely often (and $E_{i+1}, F_{i+1} \ldots F_k$ are hit infinitely often). This is equivalent to saying that the minimum element of $c(\mathit{inf}(\rho))$ is an even number.

Definition 9 *A* parity automaton *is an automaton $(Q, \Sigma, \delta, s, c)$ where Q, Σ, δ and s are as before and c is a function from Q to the set of natural numbers. A run ρ of a parity automaton is accepting if the minimum element of $c(\mathit{inf}(\rho))$ is an even number.*

We just showed that every Rabin chain automaton can be transformed into an equivalent parity automaton with the same state space. The converse is also true. Let $(Q, \Sigma, \delta, s, c)$ be a parity automaton. Let $i_1 < i_2 < i_3 \ldots < i_k$ be the numbers that appear in the range of c. We may assume without loss of generality that $i_1, i_3, \ldots$ are odd numbers and $i_2, i_4, \ldots$ are even numbers. (Hint: Suppose i_2 is also an odd number, simply set $c(q) = i_1$ whenever $c(q) = i_2$.) Let $E_j = \bigcup_{k \leq 2j-1} c^{-1}(k)$ and $F_j = \bigcup_{k \leq 2j} c^{-1}(k)$.

Exercise: Show that the above construction yields a Rabin chain automaton equivalent to the parity automaton we started with.

To summarize we have shown the following:

Theorem 10 *Any Müller automaton A with n states can be transformed into an equivalent parity automaton with $O(n!)$ states that uses only $O(n)$ numbers to label the vertices. This translation preserves determinism.*

It is also quite evident that deterministic parity automata can be complemented trivially. Just add 1 to $c(q)$ for each q to invert the parity.

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

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

Lecture 13: From Streett automata to Rabin automata and Back

Rabin and Street conditions are "complements" of each other and so it is not obvious as to how to transform an automaton of one type into the other. Let A be a Streett automaton with accepting family $(E_1, F_1), (E_2, F_2) \ldots (E_k, F_k)$. A run ρ is accepting iff

$$\forall i.\ (\mathit{inf}(\rho) \cap F_i \neq \emptyset) \Rightarrow (\mathit{inf}(\rho) \cap E_i \neq \emptyset).$$

We keep track of the E_is and F_is hit along a run as part of the state. We then use this to translate the Street acceptance condition into an equivalent Rabin condition. The idea is similar to that used in translating Müller conditions to Rabin conditions via LARs. Instead of keeping a permutation of the states we keep a permutation of the indices $1, 2 \ldots k$ giving the order in which the sets $E_1, E_2, \ldots E_k$ were last seen. We shall use $\mathsf{Perm}(k)$ to refer to the set of permutations of $1, 2, \ldots k$. However, we need to do a little more work here because, in each move the run would visit a number E_i's (as opposed to single state) and all of them have to be moved to the right end of the sequence. As in the case of LARs we use a pointer e to keep track of the leftmost position from which a index was moved right in the last move.

With these ideas we set a state to be a triple (q, I, e) where q is a state of A and I is a permutation $(i_1, i_2, \ldots i_k)$ of $(1, 2, \ldots k)$ and $1 \leq e \leq k$. The transition relation is given by: $(q', I', e') \in \delta'((q, I, e), a)$ if I' is obtained from I by moving all the indices i with $q' \in E_i$ to the right end (in some fixed order, say the lexicographic order), and e' identifies the leftmost position p in I, such that such that $q' \in E_{i_p}$ (thus, e' identifies the leftmost position from which an index was moved right in the move leading to the current state). For example, consider a transtion $q \xrightarrow{a} q'$ where $q' \in E_1$ and $q' \in E_4$. From a IAR state $(q, (3, 2, 4, 5, 1), 2)$ we would get a transition on a to $(q', (3, 2, 5, 4, 1), 3)$. The last component is 3 since the left most position from which an index (4) was moved right is 3. To ensure that e is always defined we throw in the pair (Q, Q) to the set of Streett pairs (if it is not already there). Of course, this does not change the language accepted.

For any run ρ we write $\mathsf{Infl}(\rho)$ to denote the list of indices i such that the set E_i is visited infinitely often and $\mathsf{Finl}(\rho)$ to denote the indices i such that E_i is visited finitely often along ρ. In any infinite run

$$(q_0, I_0, e_0) \xrightarrow{a_1} (q_1, I_1, e_1) \xrightarrow{a_2} \ldots$$

of this automaton on a word $a_1 a_2 \ldots$, there is a point N beyond which all indices in $\mathsf{Infl}(\rho)$ appear to the right of all the indices in $\mathsf{Finl}(\rho)$. Let m be the leftmost position whose value is taken by e infinitely often along this run. Then, for all $j \geq N$, the first $m - 1$ positions of I_j are identical and all of them belong to Finl. Moreover, if the index i appears among positions $m, m + 1, \ldots k$ at I_j, $j \geq N$, then $i \in \mathsf{Infl}(\rho)$. So far everything has been pretty much as it was in the case of LARs, with states replaced by indices.

When is such a run accepting in A? The Streett acceptance requires that if F_i is hit infinitely often then the corresponding E_i must also be hit infinitely often. Equivalently, if F_i is hit infinitely often then i must appear among positions $m + 1 \ldots k$ in $I_N, I_{N+1} \ldots$. To keep track of this we add another component, an index f, to the state. So, a state is

a 4-tuple (q, I, e, f) where q,I and e are as before. f keeps track of the leftmost position p such that $q \in F_{i_p}$ where $I = (i_1, i_2, \ldots, i_k)$. (Notice the difference between e and f. e refers to the leftmost position in the *previous state* that was moved right in the last transition. f refers to a position in the current state.)

With this definition, a run $(q_0, I_0, e_0, f_0) \xrightarrow{a_1} (q_1, I_1, e_1, f_1) \ldots$ is accepting precisely when leftmost position that appears as e_j for infinitely many js is to the left of the leftmost position that appears as f_j for infinitely many js. Equivalently, the run is accepting precisely when there is a position p such that

1. For infinitely many j, $e_j = p$.
2. $f_j < p$ for only finitely many j.

This follows from the fact that if p is hit infinitely often then $p \geq m$ and item 2 guarantees that $f_j < m$ only finitely many times. Thus, beyond some point $f_j \geq m$ and thus if F_i is hit infinitely often then so is E_i. The two conditions above can be directly translated as a Rabin pair and thus we can translate any Streett automaton into an equivalent Rabin automaton with $O(n.r!)$ states and $O(r)$ accepting pairs.

Theorem 1 *Let $A = (Q, \Sigma, \delta, s, ((E_1, F_1), (E_2, F_2) \ldots (E_k, F_k))$ be a Streett automaton. Then the automaton*

$$\mathsf{IAR}(A) = (Q \times \mathsf{Perm}(k) \times \{1, \ldots k\} \times \{1, \ldots, k\}, \Sigma, \Delta, (s, (1, 2, 3, \ldots, k), k, k), ((E'_1, F'_1), \ldots (E'_k, F'_k))$$

where the transition relation Δ is as defined in the above discussion and

$$\begin{array}{rcl} F'_i & = & \{(q, I, e, f) \mid e = i\} \\ E'_i & = & \{(q, I, e, f) \mid f < i\} \end{array}$$

accepts the same language as A. Further, A' is deterministic whenever A is deterministic. Thus any Streett automaton can be translated into and equivalent Rabin automaton, preserving determinism, whose size is bounded by $O(n.r!)$ and which has at the most r accepting pairs.

Can we extend this to a transformation from Streett automata to Parity/Rabin-Chain automata? Here is how: As discussed above a run ρ is accepting if the leftmost position that appears as e_j for infinitely many j is to the left of the leftmost position that appears as f_j for infinitely many j. This smacks of a parity condition! Note that e_j can take values among $1, 2, \ldots k$ and similarly for f_j. Let us assign the value $2i$ to e instead of i (so that e now takes values from the set $\{2, 4, \ldots 2k\}$) and the value $2i + 1$ to f instead of i (so that f takes values from the set $\{3, 5, \ldots, 2k + 1\}$).

Suppose that in the original run, the leftmost position taken infinitely often by e is j and that taken by f is l. With the new values the smallest value taken by e along this infinite run is $2j$, while the smallest value taken by f along the run is $2l + 1$. Notice that $j \leq l$ if and only if $2j < 2l + 1$. Equivalently, $j \leq l$ if and only if the smallest value taken infinitely often by either e or f is an even number. We use this to define the parity automaton. We do not bother to replace the values of e and f by $2e$ and $2f + 1$, but use the rank function to capture this.

Theorem 2 *Let $A = (Q, \Sigma, \delta, s, ((E_1, F_1), (E_2, F_2) \ldots (E_k, F_k))$ be a Streett automaton. Consider the automaton*

$$\mathsf{IAR}(A) = (Q \times \mathsf{Perm}(k) \times \{1, 2, \ldots k\} \times \{1, 2, \ldots, k\}, \Sigma, \Delta, (s, (1, 2, 3, \ldots, k), 2k, 2k+1), \sigma)$$

where the transition relation Δ is as in theorem 1 and the rank function σ is given by

$$\sigma((q, I, e, f)) \;=\; \mathsf{Min}(2e, 2f+1)$$

This parity automaton accepts the same language as A, more over this automatonis deterministic whenver A is deterministic. The size blowup is bounded by $O(n.r!)$ and the number of levels in the rank function is bounded by $O(r)$.

Rabin to Streett

Recall that a Rabin automaton can be transformed into an equivalent Nondeterministic Büchi automaton of size $O(n.r)$. And since NBAs are also NSAs, this gives a efficient way to tranform NRAs into NSAs. But this does not preserve determinism. However, given a DRA, we can complement it to get a DSA (without any blowup), transform the DSA into an equivalent Deterministic Parity automaton (of size $O(n.r!)$ and with at the most $O(r)$ rank levels) and finally complement this parity automaton (without any blowup) to get a DPA automaton equivalent to the original Rabin automaton. Thus, we also have transformations from Rabin automata to Streett automata and Parity automata with a size blow up at at most $O(n.r!)$ and with at the most $O(r)$ size for the acceptance condition.

A Lowerbound on transformations from Streett Automata

In this section we shall show that the transformation from deterministic Streett automata to deterministic Rabin automata described in the previous section is optimal. This result (and the following proof) is due to Christof Löding ([**?**]).

The technique used to prove this result will be almost identical to that used to proving the optimality of Safra's construction. Recall, that the proof there proceeded using the following steps:(where L_n is the language used in that proof.)

1. Assume that there is a NSA of size $< n!$ that accepted the language L_n.
2. Pick words $\sigma_1, \sigma_2 \ldots \sigma_k$ in L_n and corresponding accepting runs $\rho_1, \rho_2, \ldots \rho_k$.
3. Construct a new word σ and a run ρ on σ by splicing together parts of $\rho_1, \rho_2, \ldots$ such that $inf(\rho) = \bigcup_{1 \leq i \leq k} Inf(\rho_i)$.
4. Show that $\sigma \notin L_n$ and use the fact that if X and Y satisfy a Streett condition then so does $X \cup Y$ to conclude that we have an accepting run on a word not in L_n and arrive at a contradiction to 1.

A similar technique would not work for Nondeterministic Rabin automata. For Rabin conditions the analogue of the property used in item 3 above is: if X and Y do not satisfy a Rabin condition then $X \cup Y$ also does not satisfy the Rabin condition. To use this property we have dualize the entire argument and get:

1. Assume that there is a NRA of size $< n!$ that accepted the language L_n.
2. Pick words $\sigma_1, \sigma_2, \ldots \sigma_k$ outside L_n and corresponding non-accepting runs $\rho_1, \rho_2, \ldots \rho_k$.
3. Construct a new word σ and a run ρ on σ by splicing together parts of $\rho_1, \rho_2, \ldots$ such that $\mathit{inf}(\rho) = \bigcup_{1 \leq i \leq k} Inf(\rho_i)$.
4. Show that $\sigma \in L_n$ and use the fact that if X and Y do not satisfy a Rabin condition then $X \cup Y$ also does not satisfy it to conclude that ρ is not an accepting run.

But we have no contradiction since there might be other runs that accept σ. As a matter of fact, I do not know any technique to show a lowerbound on the translation from Streett automata to Nondeterministic Rabin automata.

However if we work with Deterministic Rabin automata the above technique works, because, ρ is the unique run for the automaton on σ and if it is rejecting then there is no accepting run and so it rejects a word in L_n and thereby contradicts 1. This is the scheme we shall follow here. Consider the family of automata $(A_n)_{n \geq 1}$ described below.

$$A_n \;=\; (\{i, -i \mid 1 \leq i \leq n\}, \{1, 2, \ldots n\}, \delta, s, ((\{1\}, \{-1\}), (\{2\}, \{-2\}), \ldots, (\{n\}, \{-n\})))$$

where $\delta(i, j) = -j$ and $\delta(-i, j) = j$ for all $1 \leq i, j \leq n$. The automaton is in negative states after even number of moves and in positive states after odd number of moves. Moreover, the state reached after reading a word w depends only on the last letter of w (and the parity of the length of w).

Let $\rho = a_0 a_1 \ldots$ be a word over Σ. The set of positive states entered during this run is completely determined by $a_0 a_2 a_4 \ldots$ and similarly the set of negative states entered is determined by $a_1 a_3 \ldots$. In particular if $\mathsf{even}(\rho)$ is the set of letters that appear infinitely often at positions $0, 2, \ldots$ and $\mathsf{odd}(\rho)$ is the set of letters that appear infinitely often at positions $1, 3, \ldots$ then, the word ρ is accepted if and only if $\mathsf{odd}(\rho) \subseteq \mathsf{even}(\rho)$. This leads us to the following observation:

Observation 1: Let u be a word of even length over Σ_n. For any ρ, ρ is in the language L_n if and only if $u\rho$ is in L_n.

The proof proceeds by induction on n. We shall actually prove a slightly stronger result. Namely:

The Hypothesis: Any deterministic Rabin automaton A accepting the language L_n must contain a strongly connected component with at least $n!$ states.

If n is 1, the result follows immediately as any automaton accepting L_1 must have at least a scc with one state.

For the induction step, pick an automaton A with minimum size that accepts L_n. If we omit the letter i from the alphabet of A, it accepts the language L_{n-1} (modulo some renaming of letters) and therefore, by the induction hypothesis, we have the following observation:

Observation 2: The automaton A has an scc with least $(n-1)!$ states. As a matter of fact, A restricted to the set of letters $\Sigma - \{i\}$ has a scc with at least $(n-1)!$ states.

We can say something interesting about the structure of A:

Observation 3: The automaton A considered as a graph is strongly connected.

Suppose A has multiple sccs. These sccs themselves form a directed acyclic graph. Pick any scc at the leaf of this DAG. Now, there is a word of even length u from s to some state q in this scc. But $\sigma \in L_n$ if and only if $u\sigma \in L_n$. Thus, the automaton A with q as the start state accepts exactly the same language as A. But the states reachable from q are only its scc (since this scc forms a leaf in the DAG on sccs). Thus, we might as well restrict ourselves to this scc and use q as the start state to obtain an automaton that accepts L_n. But we had assumed that A was the smallest automaton that accepts L_n. Therefore, A must consist of just a single strongly connected component.

Observation 4: Any automaton A accepting L_n has a sub scc that accepts L_n.

This just follows from the argument given above to prove Observation 3.

Now, for each i we construct a word σ_i that $\mathsf{even}(\sigma_i) = \Sigma_n - \{i\}$ and $\mathsf{odd}(\sigma_i) = \Sigma_n$. Thus none of these words are accepted by A. The word σ_i and the run of A on σ_i are constructed in unison as follows: Let the word $u_s^i = 112233\ldots(i-1)(i-1)(i+1)(i+1)\ldots nn$. This word has every letter in $\Sigma_n - \{i\}$ in both odd and even numbered positions. Suppose this word leads us from s to some state q. Now, A with q as starting state accepts L_n. Thus, by Observation 2, there is an scc of size at least $(n-1)!$ reachable from q even when the alphabet is restricted to $\Sigma_n - \{i\}$. Let w be word over $\Sigma_n - \{i\}$ that labels a path from q to this scc and visits at least $(n-1)!$ states in this scc. We set $v_s^i = u_s^i.w.ik$ where $k \in \Sigma_n - \{i\}$. We point out two important properties of this word:

1. The set of letters appearing at odd positions is Σ_n and the set of letters appearing at even numbered positions is $\Sigma_n - \{i\}$.

2. The run ρ_s^i of the automaton A on this word (starting at s) has atleast $(n-1)!$ distinct states.

We could do this for any state q. That is, we could construct a word v_q^i such that the two properties listed above are satisified (with q in place of s.) The word $\sigma_i = v_s^i v_{q_1}^i v_{q_2}^i \ldots$, where q_{i+1} is the state reached on reading v_q^i starting at state q_i. The run ρ_i on σ_i is $\rho_s^i \rho_{q_1}^i \ldots$.

We now show that $\mathit{inf}(\rho_i) \cap \mathit{inf}(\rho_j)$ must be empty whenever $i \neq j$. Suppose that a state q appears infinitely often in both these runs. Let $\rho_i = r_1.r_2.\rho_i'$ where

1. r_1 leads to the state q
2. r_2 (begins and) ends in the state q
3. r_2 includes at least one of the ρ_p^i entirely (and therefore the word read along r_2 contains some v_p^i entirely)
4. r_2 visits exactly the set of states in $\mathit{inf}(\rho_i)$.

This can always be ensured since q is visited infinitely often.

Similarly we can write $\rho_j = t_1.t_2.\rho_j'$ satisfying similar properties. Now consider the run $\rho = r_1(r_2t_2)^\omega$. This run reads a word σ that includes infinitely many copies of some ρ_p^i and infinitely many copies of some $\rho_{p'}^j$. Thus, $\mathsf{even}(\sigma) = \mathsf{odd}(\sigma) = \Sigma_n$. Thus, the word is in L_n. However, the set of states hit infinitely often along this run is $\mathit{inf}(\rho_i) \cup \mathit{inf}(\rho_j)$. Thus the run ρ cannot satisfy the Rabin condition. However, the automaton is deterministic and ρ is the only run of this automaton on σ and this contradicts the assumption that $L(A) = L_n$. Hence, we may conclude that $\mathit{inf}(\rho_i) \cap \mathit{inf}(\rho_j)$ is empty whenver $i \neq j$. Notice that there are at least $(n-1)!$ states in each $\mathit{inf}(\rho_i)$. Thus, there are at least $n!$ states in A.

Now, we can apply Observation 4 to conclude that any automaton accepting L_n has an scc with at least $n!$ states.

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

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