

Introduction

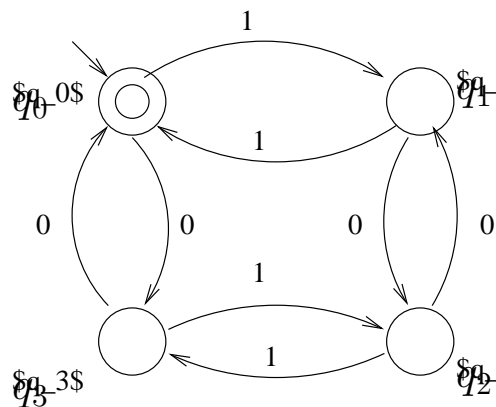
- Finite Automata
 - DFA, regular languages
 - Nondeterminism, NFA, subset construction
- Regular Expressions
 - Syntax, Semantics
 - Relationship to regular languages
- Properties of regular languages
 - Pumping lemma
 - Minimizing DFA, Myhill-Nerode theorem
- Machines
 - Modeling hardware in terms of machines
 - Semantics of interaction, complexity issues

References:

1. “Introduction to Automaton Theory, Languages, and Computation”, Hopcroft and Ullman, 1979, Chapters 2,3
2. “Elements of the Theory of Computation”, Lewis and Papadimitrou, 1981, Chapters 1,2

Finite Automaton

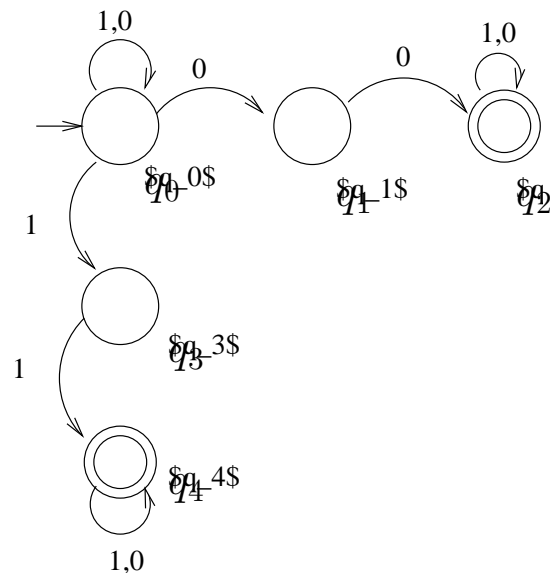
- Arise in various contexts: Text Editors, Neuron modelling, Hardware
- **Notation:** Given an alphabet Σ , Σ^* is the set of all finite strings on Σ , a language is any $L \subset \Sigma^*$, $a \cdot b$ denotes the concatenation of string a with b , ϵ is the empty string
- Deterministic Finite Automaton : characterised by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$
 1. Q - finite set of states
 2. Σ - finite input alphabet
 3. δ - transition function, maps $Q \times \Sigma \mapsto Q$ (can assume total)
 4. q_0 - initial state
 5. F - subset of S , the set of final states
- Example



- Behavior of DFA on input strings:
Extend δ to $\hat{\delta} : Q \times \Sigma^* \mapsto Q$ in natural way
 1. $\hat{\delta}(q, \epsilon) = q$
 2. $\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$
- **Language** of DFA A : $L(A) = \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) \in F\}$
- Language L is said to be regular if \exists DFA M such that $L = L(M)$

Nondeterminism

- Less obvious, useful for theoretical analysis, modelling initial stages of hardware designs, can be more compact
- Nondeterministic Finite Automaton : characterised a 5-tuple $(Q, \Sigma, \Delta, q_0, F)$
 1. Q - finite set of states
 2. Σ - finite input alphabet
 3. Δ - transition function, maps $Q \times \Sigma \mapsto 2^Q$ (variously $\Delta \subset Q \times \Sigma \times Q$)
 4. q_0 - initial state
 5. F - subset of S , the set of final states
- Example



- Behavior of NFA on input strings:

Extend Δ to $\hat{\Delta} : Q \times \Sigma^* \mapsto 2^Q$ in natural way

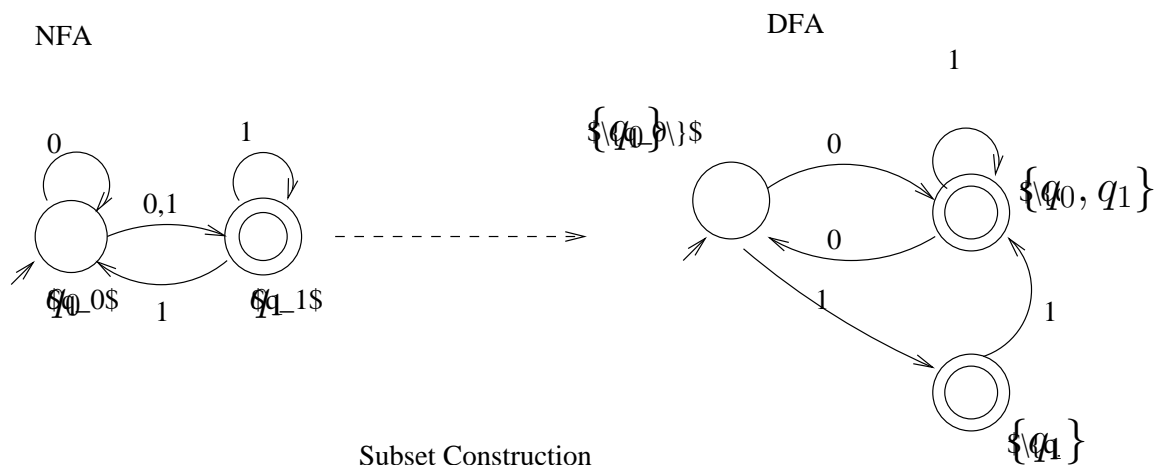
1. $\hat{\Delta}(q, \epsilon) = \{q\}$

2. $\hat{\Delta}(q, wa) = \{p \mid \exists r \in \hat{\Delta}(q, w), p \in \Delta(r, a)\}$

- **Language** of NFA A : $L(A) = \{x \in \Sigma^* \mid \hat{\Delta}(q_0, x) \cap F \neq \phi\}$

Nondeterminism - II

- Obvious fact : Class of languages accepted by NFA \supseteq Class of languages accepted by DFA ; Less obvious fact : Equality holds
- Proof : by the subset construction
- Intuition : Nondeterminism leads to parallel paths \Rightarrow simulate NFA to determinize
- Example



- **Remark:** Potential blowup; generate subsets incrementally; Average case increase ?

Regular Expressions

- **Idea:** Formulae that define certain subsets of Σ^*
- not commonly used to describe systems; describe properties
eg “3 REQs never occur consecutively”
- **Notation:** Let $L, L_1, L_2 \subset \Sigma^*$
 1. Concatenation of L_1 and $L_2 = \{x \cdot y \mid x \in L_1 \wedge y \in L_2\}$
 2. Define $L^0 = \{\epsilon\}$ and $L^i = L \cdot L^{i-1}$ for $i \geq 1$

$$\begin{aligned}\text{Kleene closure } L^* &= \bigcup_{i=0}^{\infty} L^i \\ \text{positive closure } L^+ &= \bigcup_{i=1}^{\infty} L^i\end{aligned}$$

- **Example:** Σ - finite alphabet

$$\begin{aligned}L_1 &= \{x \mid x \text{ has 3 consecutive 0's}\} \\ L_2 &= \{y \mid y \text{ has 2 consecutive 1's}\} \\ L_1 \cdot L_2 &= \{z \mid z \text{ has 3 consecutive 0's} \\ &\quad \text{and 2 consecutive 1's which follow the 3 0's}\} \\ L &= \{x \mid x \text{ has an even number of 1's}\} \\ L^* &= L\end{aligned}$$

Regular Expressions - II

- Syntax and Semantics
 1. ϕ is a regular expression, denotes the empty set
 2. ϵ is a regular expression, denotes the set $\{\epsilon\}$
 3. $\forall a \in \Sigma$, \mathbf{a} is a regular expression, denotes the set $\{a\}$
 4. let r, s be regular expressions denoting the languages R and S , then $(r + s)$, $(r \cdot s)$, $(r)^*$ are regular expressions denoting $R \cup S$, $R \cdot S$, and R^* respectively
- **Example:** $0^* + (0^* \cdot 1 \cdot 0^* 1 \cdot 0^*)^*$
- **Remark:** Can extend above definition to include other operators such as \wedge - intersection, \neg - complement, $()^2$ - squaring, $()^i$ - exponentiation where i is encoded in binary. The resultant expressions do not define any new languages, but can be exponentially more succinct. Interesting fact : deciding universality of regular expressions drawn from $+, \cdot, *, ()^i$ is EXP-space complete.

Equivalence of FA and Regular Expressions

- **Theorem:** Given any regular expression r , there exists an FA accepting $L(r)$

Proof: by construction, use ϵ transitions

- **Theorem:** [Kleene, 1956] Given a regular language L , there is a regular expression r such that $L = L(r)$.

Proof: Let L be a regular set. There is a DFA which accepts L . Let the DFA be $M = (\{q_1, \dots, q_n\}, \Sigma, \delta, q_1, F)$. We will inductively construct regular expressions that drive the DFA from one state to another.

Let R_{ij}^k denote all strings x such that $\delta(q_1, x) = q_j$ and if $\delta(q_i, y) = q_l$ for some y which is a prefix of x , other than x or ϵ , then $l \leq k$.

Clearly :

$$\begin{aligned} R_{ij}^0 &= \{a \mid \delta(q_1, x) = q_j\} \text{ if } i \neq j \\ &= \{a \mid \delta(q_1, x) = q_j\} \cup \{\epsilon\} \text{ if } i = j \\ R_{ij}^k &= R_{ij}^{k-1} \cup R_{ik}^{k-1} \cdot (R_{kk}^{k-1})^* \cdot R_{kj}^{k-1}, \quad k \geq 1 \end{aligned}$$

Claim: $\forall i, j, k$ R_{ij}^k is defined by a regular expression r_{ij}^k

Proof: By induction on k

Base case: R_{ij}^0 is a finite set \Rightarrow

$$\begin{aligned} r_{ij}^0 &= (a_1 + \dots + a_p) \text{ if } i \neq j \\ &= (a_1 + \dots + a_p + \epsilon) \text{ if } i = j \end{aligned}$$

Induction:

$$r_{ij}^k = r_{ij}^{k-1} \cup r_{ik}^{k-1} \cdot (r_{kk}^{k-1})^* \cdot r_{kj}^{k-1}, k \geq 1$$

Conclusion:

$$L(M) = \bigcup_{q_j \in F} R_{ij}^n \Rightarrow r^L = r_{ij_1}^n + \dots + r_{ij_p}^n$$

where $F = \{q_{j_1}, \dots, q_{j_p}\}$

Equivalence of FA and Regular Expressions

- **Theorem:** Given any regular expression r , there exists an FA accepting $L(r)$

Proof: by construction, use ϵ transitions

- **Theorem:** [Kleene, 1956] Given a regular language L , there is a regular expression r such that $L = L(r)$.

Proof: Let L be a set accepted by the DFA

$M = (\{q_1, \dots, q_n\}, \Sigma, \delta, q_1, F)$ Let R_{ij}^k denote all strings x such that $\delta(q_1, x) = q_j$ and if $\delta(q_i, y) = q_l$ for some y which is a prefix of x , other than x or ϵ , then $l \leq k$.

Clearly :

$$\begin{aligned} R_{ij}^0 &= \{a \mid \delta(q_1, x) = q_j\} \text{ if } i \neq j \\ &= \{a \mid \delta(q_1, x) = q_j\} \cup \{\epsilon\} \text{ if } i = j \\ R_{ij}^k &= R_{ij}^{k-1} \cup R_{ik}^{k-1} \cdot (R_{kk}^{k-1})^* \cdot R_{kj}^{k-1}, k \geq 1 \end{aligned}$$

Lemma: $\forall i, j, k$ R_{ij}^k is defined by a regular expression r_{ij}^k

Proof: By induction on k

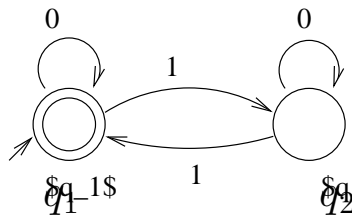
Base case: R_{ij}^0 is a finite set $\Rightarrow r_{ij}^0 = (a_1 + \dots + a_p)$ or $= (a_1 + \dots + a_p + \epsilon)$ if $i = j$ Induction: $r_{ij}^k = r_{ij}^{k-1} \cup r_{ik}^{k-1} \cdot (r_{kk}^{k-1})^* \cdot r_{kj}^{k-1}, k \geq 1$ **Conclusion:**

$$L(M) = \bigcup_{q_j \in F} R_{ij}^n \Rightarrow r^L = r_{ij_1}^n + \dots + r_{ij_p}^n$$

where $F = \{q_{j_1}, \dots, q_{j_p}\}$

Equivalence of FA and Regular Expressions - II

- **Example:**



- Step 1

$$r_{11}^0 = \epsilon + 0, r_{22}^0 = \epsilon + 0, r_{12}^0 = r_{21}^0 = \phi$$

- Step 2

$$r_{11}^1 = 0^*, r_{22}^1 = 0 + 1 \cdot 0^* \cdot 1, r_{12}^1 = r_{21}^1 = 1$$

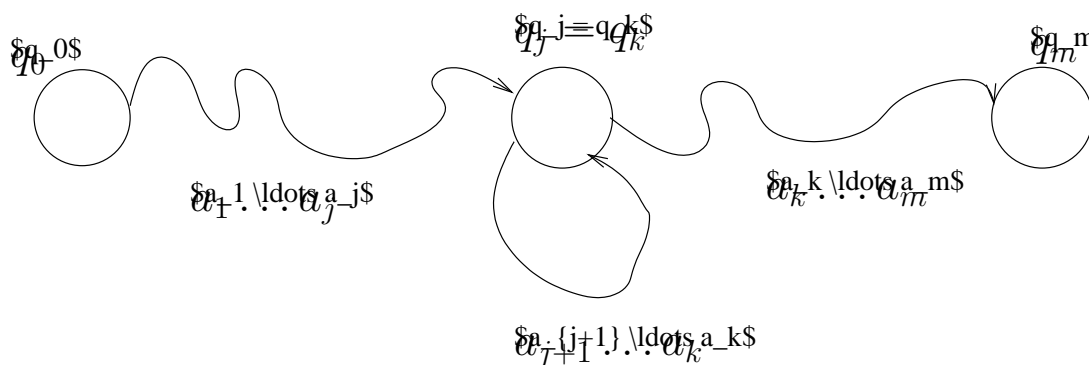
- Step 3

$$r_{11}^2 = 0^* + (0 + 1 \cdot 0^* \cdot 1)^*$$

Remark: Associating formula with states is a common theme in verification, eg can characterize states in a Kripke structure upto bisimulation using a CTL formula

Regular Language Properties - The Pumping Lemma

- Let L be a regular language \Rightarrow defined by DFA $M = (Q, \Sigma, \delta, q_0, F)$
- Consider an input of n or more symbols a_1, a_2, \dots, a_m $m \geq n$
 for $i = 1, 2, \dots, m$ let $\delta(q_0, a_1, \dots, a_i) = q_i$
 Observe: Cant have q_0, q_1, \dots, q_n all distinct
 $\Rightarrow \exists j, k$ $0 \leq j < k \leq n \wedge q_j = q_k$
- Example



- $a_m \in F$ iff $a_1 a_2 \dots a_m \in L(M) \Rightarrow$ so is the string $a_1 \dots a_j a_{k+1} \dots a_m$
 in fact can loop around as many times as desired:

$$a_1 \dots a_j (a_{j+1} \dots a_k)^i a_{k+1} \dots a_m \in L(M) \forall i \geq 0$$

- If FA accepts arbitrarily long strings \Rightarrow must have a string s with the property \exists a substring of s that can be repeated

Pumping Lemma: Let L be a regular set. Then there exists a constant n such that if $z \in L$ and $|z| \geq n$, can write $z = u \cdot v \cdot w$ in such a way that $|u \cdot v| \leq n$, $|v| \geq 1$ and $\forall i \geq 0 u \cdot v^i \cdot w \in L$. Also $n \leq \#$ number of states in smallest DFA for L

Regular Language Properties - Closure, Minimization

- **Closure**

- Closure under union, concatenation, and Kleene closure : follows from equivalence of regular expression and FA
- Closure under complementation, intersection : trivial to complement DFA
- Deciding Emptiness, Universality, Equivalence of DFA
- Deciding Emptiness, Universality, Equivalence of NFA

- **Minimizing DFA**

- The Myhill-Nerode theorem : Given a regular language L , there is a unique minimum state DFA accepting the language. Proof based on partitioning Σ^* , under the equivalence relation

$$xR_Ly \text{ iff } \forall z \ x \cdot z \in L \leftrightarrow y \cdot z \in L$$

- Simple Algorithm : Merge states in a DFA if there is no input string that can differentiate them

Hardware

- **Scenario:**

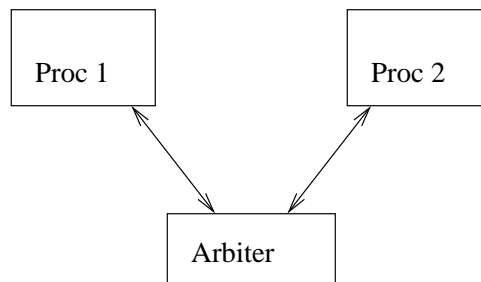
- Two processors, single disk drive
- Only one processor should be allowed to access the drive at any time
- Design arbiter to decide which processor gets access
- Processors and arbiter communicate through

$$\Sigma_{P_1} = \{Idle1, Req1, Busy1\},$$

$$\Sigma_{P_2} = \{Idle2, Req2, Busy2\},$$

$$\Sigma_{Arbit} = \{Stop, Go1, Go2\}$$

- **Block Diagram**

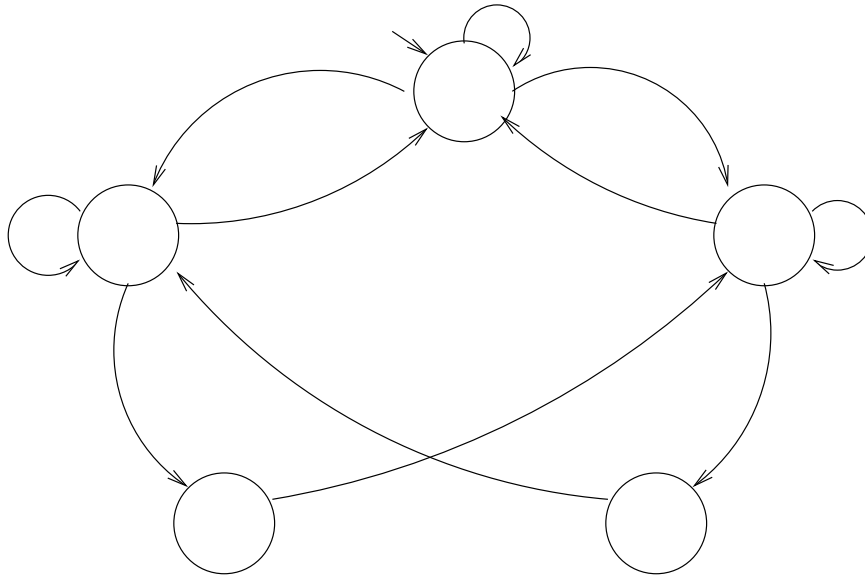


$$\text{Output}_{Proc1} \equiv \{Idle1, Request1, Busy1\}$$

$$\text{Output}_{Proc2} \equiv \{Idle2, Request2, Busy2\}$$

$$\text{Output}_{Arbiter} \equiv \{Stop, Go1, Go2\}$$

• Controller State Diagram



• Observe:

- Nondeterminism in the controller
- Inputs, Outputs, States : Symbolic
- Transition Conditions (“Predicates”) are described by formulae
- Outputs on state : ‘Moore machine’

• Properties:

- Safety : Never give both processors access to drive at same time
- Liveness : Processor makes request \Rightarrow eventually gets access

• Relationship to Regular Languages:

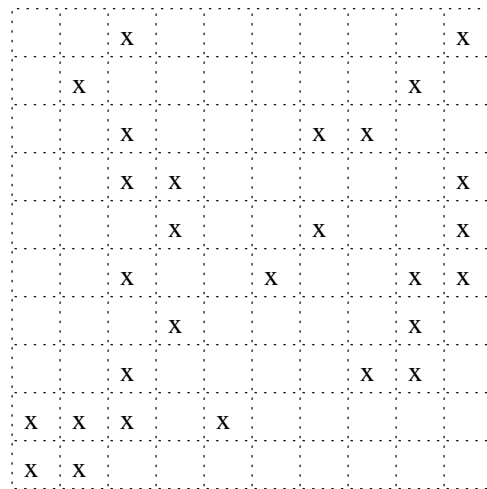
View behavior of arbiter as map from $(\Sigma_{P1} \times \Sigma_{P2})^*$ to $(\Sigma_{Arbit})^*$.
Can naturally associate a regular language with the behavior.

Interacting Machines

- **Game of Life**

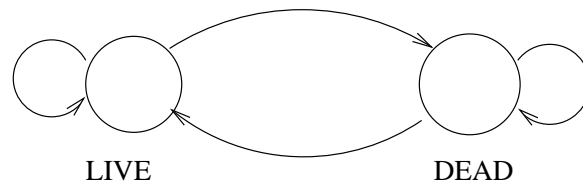
Single cellular organisms, atmost one on each square of a chess board. Only two states: $\{ Dead, Live \}$, simple transition rule

- **Example:**



Petri Dish

- **Transition rule:**



- **Definition:** The **product machine** $M = M_1 \times M_2 \times \dots \times M_n$ is the machine on state space $S = S_1 \times S_2 \times \dots \times S_n$ and output space $O = O_1 \times \dots \times O_n$ characterized by

– Output relation

$$\Theta(x, o) = \prod_{i=1}^n \Theta_i(x_i, o_i)$$

– Transition relation

$$T(x, y) = (\exists \vec{i}) \left[\prod_{i=1}^n T_i(x_i, i, y_i) \cdot \Theta_i(x_i, o_i) \cdot (i = [o_1 \dots o_n]) \right]$$

where the present state variable $s = [s_1 s_2 \dots s_n]$, and the next state variable $t = [t_1 t_2 \dots t_n]$.

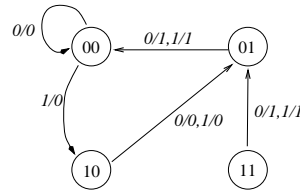
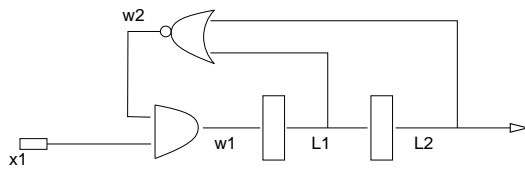
- **Complexity :** The computational complexity of verifying systems of interacting machines is very high; for example the following problem is P-space complete:

Reached State Analysis:

Given a system of interacting machines, and two states s and p , is a s reachable from p ?

Modeling Hardware

Recall — can go from netlist to FSM



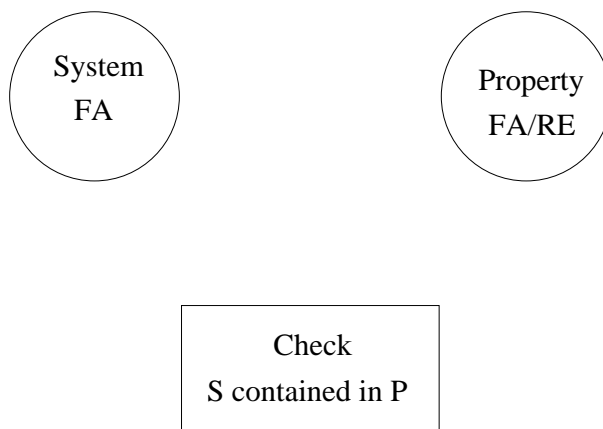
- Problem 1 — is “output behavior”

of N_1 at state s_0 equal to “output behavior” of N_2 at state s_1 ?

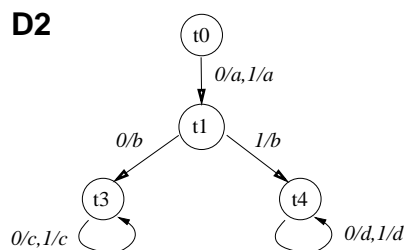
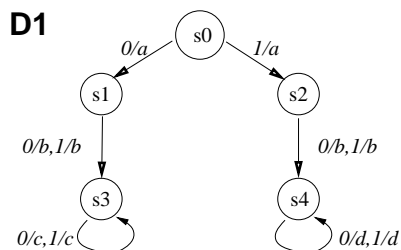
- Problem 2 — is N_1 a safe replacement for N_2 ?

Language Containment

Suggests another verification paradigm — **Language Containment**



- What properties can we express in this paradigm?



ω -Automata

Accept **infinite** sequences rather than finite sequences!!!

⇒ Contrast with Finite Automata (FA)

Motivation:

- Would like to be able to express properties like

$$\mathbf{AG} (\text{REQ}=1 \rightarrow (\mathbf{AF} \text{ACK}=1))$$

- Intuitively, such a property is a function of input/output behavior of design, but can not be captured using Finite Automata.
- Would like to capture designs which have fairness constraints
 - How to describe processor with fairness constraint which ensures $\text{REQ}=1$ infinite often?

Terminology

1. Recall $\omega = \{0, 1, 2, \dots\}$; an ω sequence f over a set Σ is a function from ω to Σ .

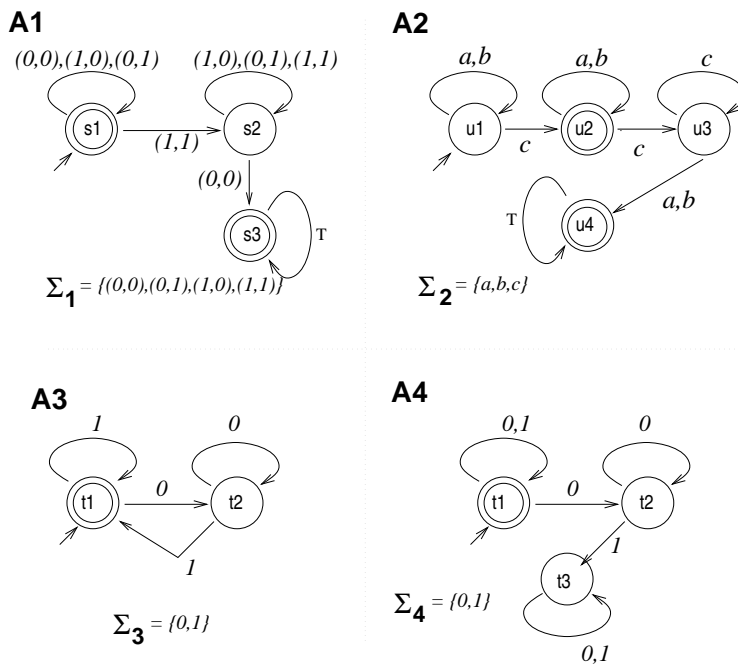
$$\langle f(0), f(1), f(2), \dots, \rangle$$

2. Given set Σ (alphabet), Σ^ω is set of all infinite sequences over Σ .
3. Subsets of Σ^ω are referred to as ω -languages.

How to build automata accepting ω -sequences???

Defn. Büchi automata — syntax and semantics

- Syntactically identical to Finite Automata
 - Same dichotomy between deterministic and nondeterministic
- Semantics — Accept **infinite** input sequence exactly when there is a resulting path on which an accepting state occurs infinitely often.



Defn. An ω -language L is said to be ω -regular if it is accepted (variously defined, recognized) by some Büchi automaton.

Closure Properties of ω -regular languages

1. Closed under **union** — could do exactly as for FA

Alternate approach — product construction. Let L_1 and L_2 be ω -regular languages over alphabet Σ .

ω -regular \Rightarrow must be accepted by BA.

Let $(S_1, \alpha, T_1, \Sigma, F_1)$ and $(S_2, \alpha, T_2, \Sigma, F_2)$ be BA for L_1 and L_2 .

Consider the following BA:

- States = $S_1 \times S_2$
- Initial state = (α, β)
- Transition Relation

$$T = \{((s_1, s_2), a, (t_1, t_2)) \mid (s_1, a, t_1) \in T_1 \wedge (s_2, a, t_2) \in T_2\}$$

- Alphabet = Σ
- Accepting States = $F_1 \times S_2 \cup S_1 \times F_2$

Why does this work?

2. Closed under **intersection** — for FA, could use complementation, union, and DeMorgans's laws.

Can we do directly?

First try: build product automata

(a) Accepting States = $F_1 \times F_2$

Q. Does this catch everything?

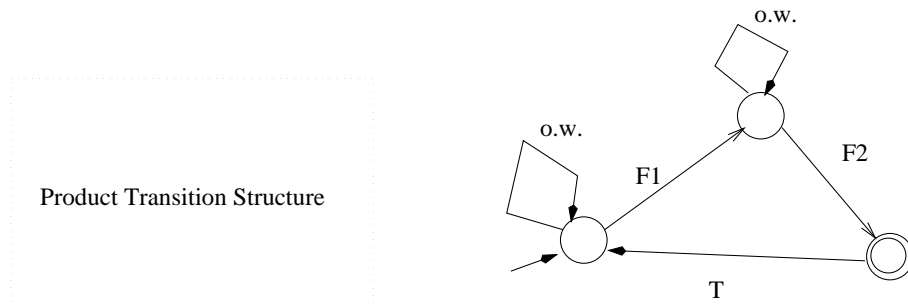


Figure 1: Monitor for Intersection

3. Closure under projection — eraser construction, exactly as before (May go from deterministic to nondeterministic.)

Closure of ω -regular languages under Complement

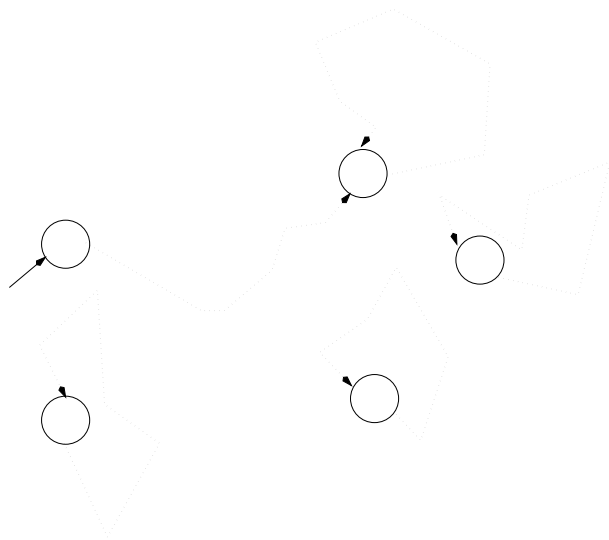
How did we achieve this for *-regular languages?

1. L is *-regular \rightarrow has an NFA N_L .
2. From N_L built DFA D_L for L through subset construction.
3. Complementing D_L — trivial.

First attempt for ω -regular languages — mimic above.

1. Problem: how to complement D_L ?
2. Not possible in general!!! Consider

$$L = \{x \in \{0, 1\}^\omega \mid 0 \text{ occurs a.a. in } x\}$$



Fundamental result: [Büchi 1969] ω -regular languages are closed under complement

- Idea: go from NBA to deterministic Müller automaton which are easy to complement; then go back to NBA

General Knowledge

Other types of ω -automata: states, transition structure remains the same. Differ in acceptance condition. Let σ be a path, $\text{inf}(\sigma)$ set of states occurring infinitely often in σ .

1. **Müller automata:** F = set of subset of state set.
 - σ is accepted iff $\text{inf}(\sigma)$ is a member of F
2. **Streett automata:** F = set of pairs of subsets of state space.
 - σ is accepted iff for every pair (A_i, B_i) in F , $A_i \cap \text{inf}(\sigma) = \emptyset$ or $B_i \cap \text{inf}(\sigma) \neq \emptyset$ in the path
3. **Rabin automata:** F = set of pairs of subsets of state space.
 - σ is accepted iff for some pair (A_i, B_i) in F , $A_i \subset \text{inf}(\sigma)$ or $B_i \cap \text{inf}(\sigma) \neq \emptyset$ in the path

All these accept exactly the class of ω -regular languages (even in the deterministic case). Differ in terms of succinctness — refer to Safra FOCS 1988.

- Language containment paradigm for ω -automata — basis for COSPAN.

Relationship to Temporal Logic

- Generally — can't convert CTL formula to Automata ($*$ or ω).
 - For the logic PLTL, can automatically build Büchi automata

- To analyse CTL (and other “branching time” logics) from automata point of view, need automata which accept **trees**.
 - Similar issues — complementation, expressiveness, etc.
 - Strong analogy to language containment