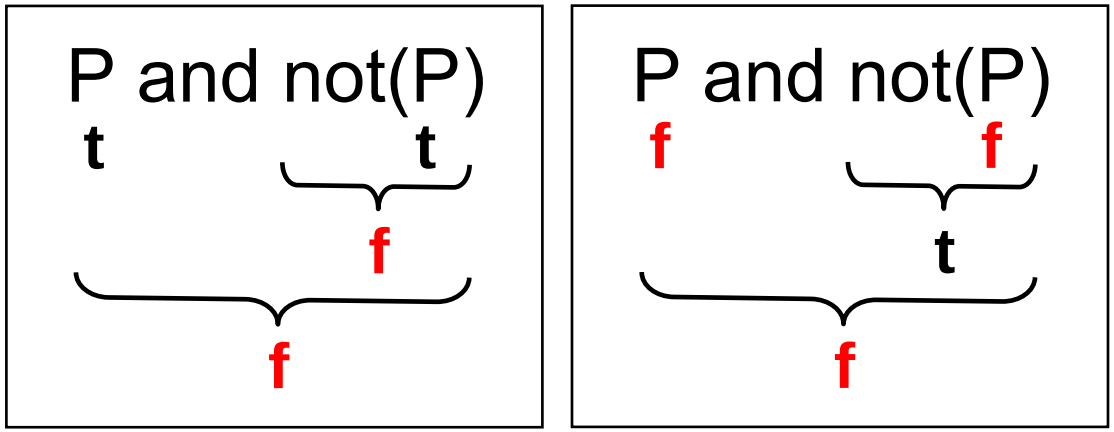
Review of Informatics 1 Computation & Logic

Basics for the exam

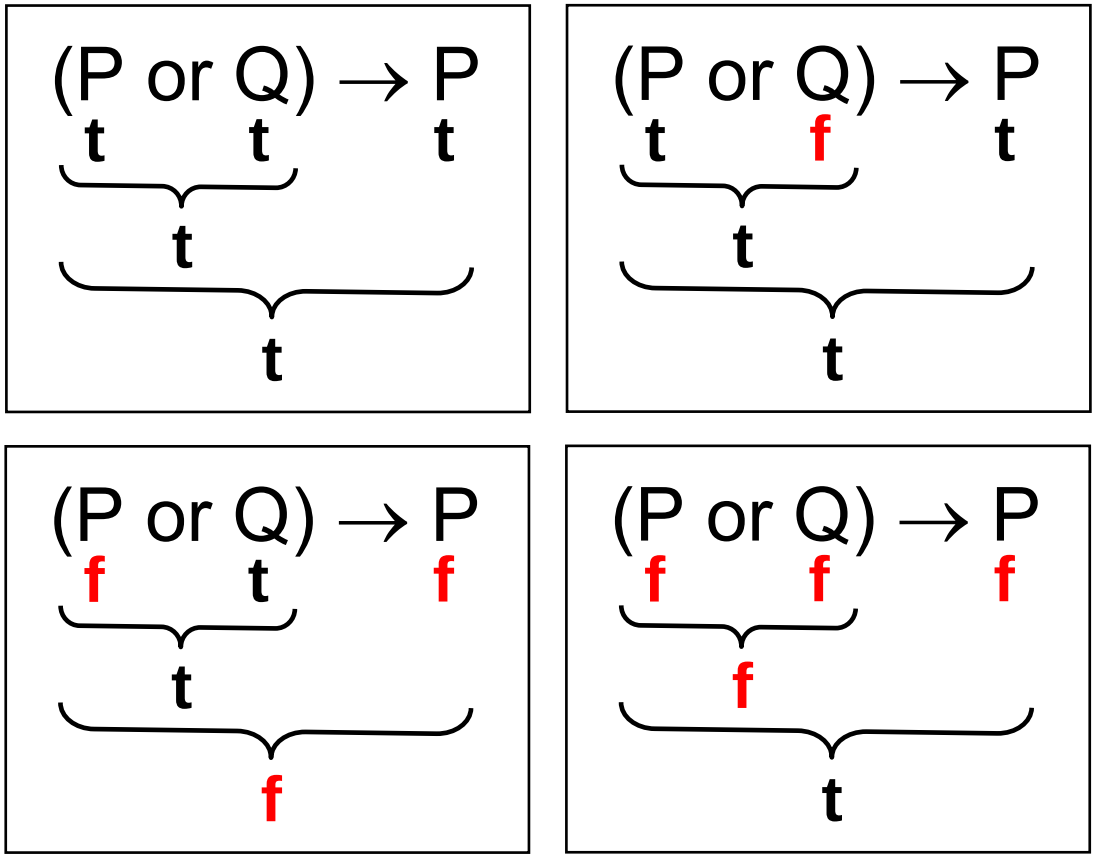
Truth Tables

P	Q	not(P)	P and Q	P or Q	$P \rightarrow Q$	$P \leftrightarrow Q$
t	t	f	t	t	t	t
t	f	f	f	t	f	f
f	t	t	f	t	t	f
f	f	t	f	f	t	t

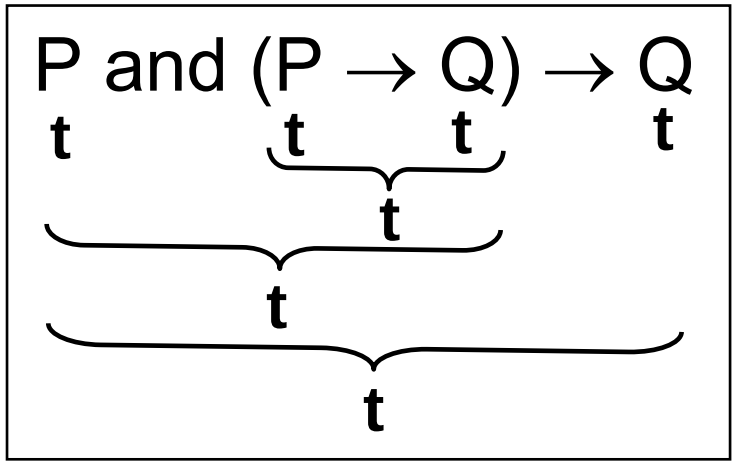
Inconsistencies

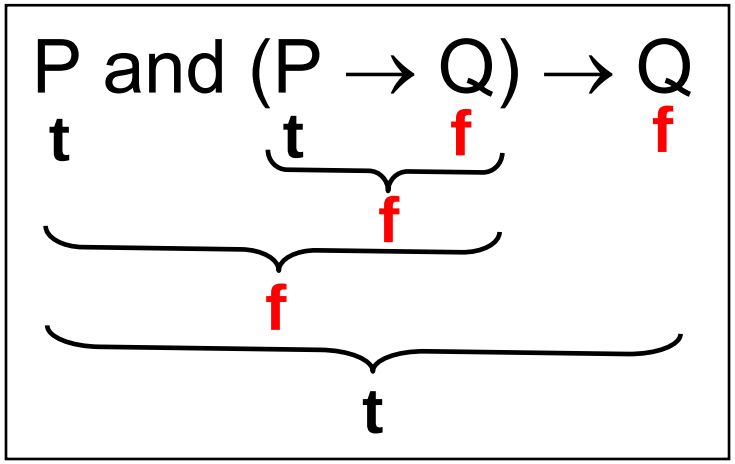


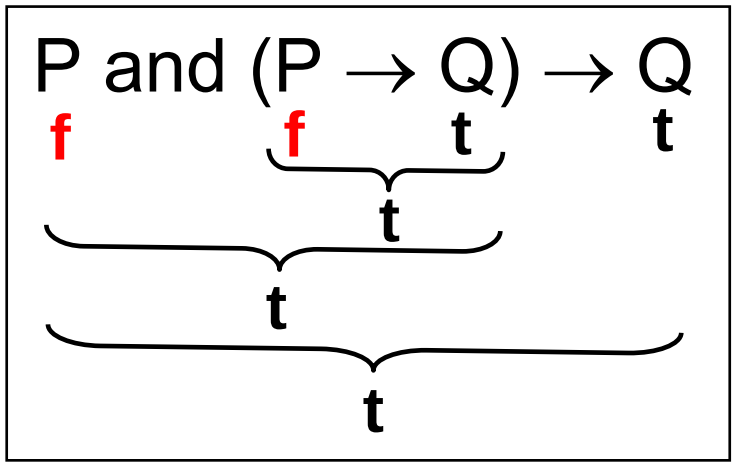
Contingencies

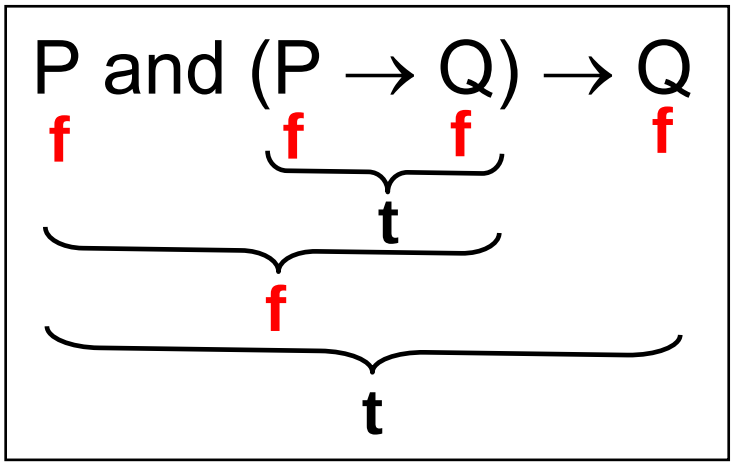


Tautologies

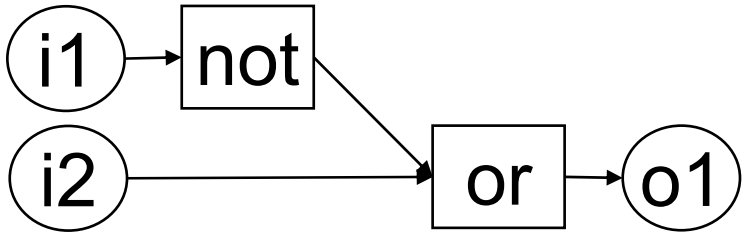




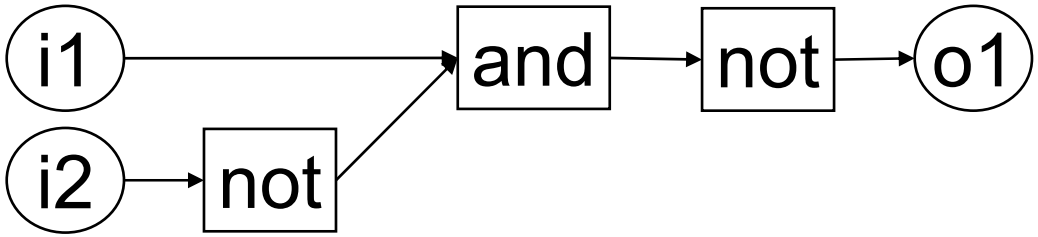




Typical Problem



equivalent ?



not(i1) or i2

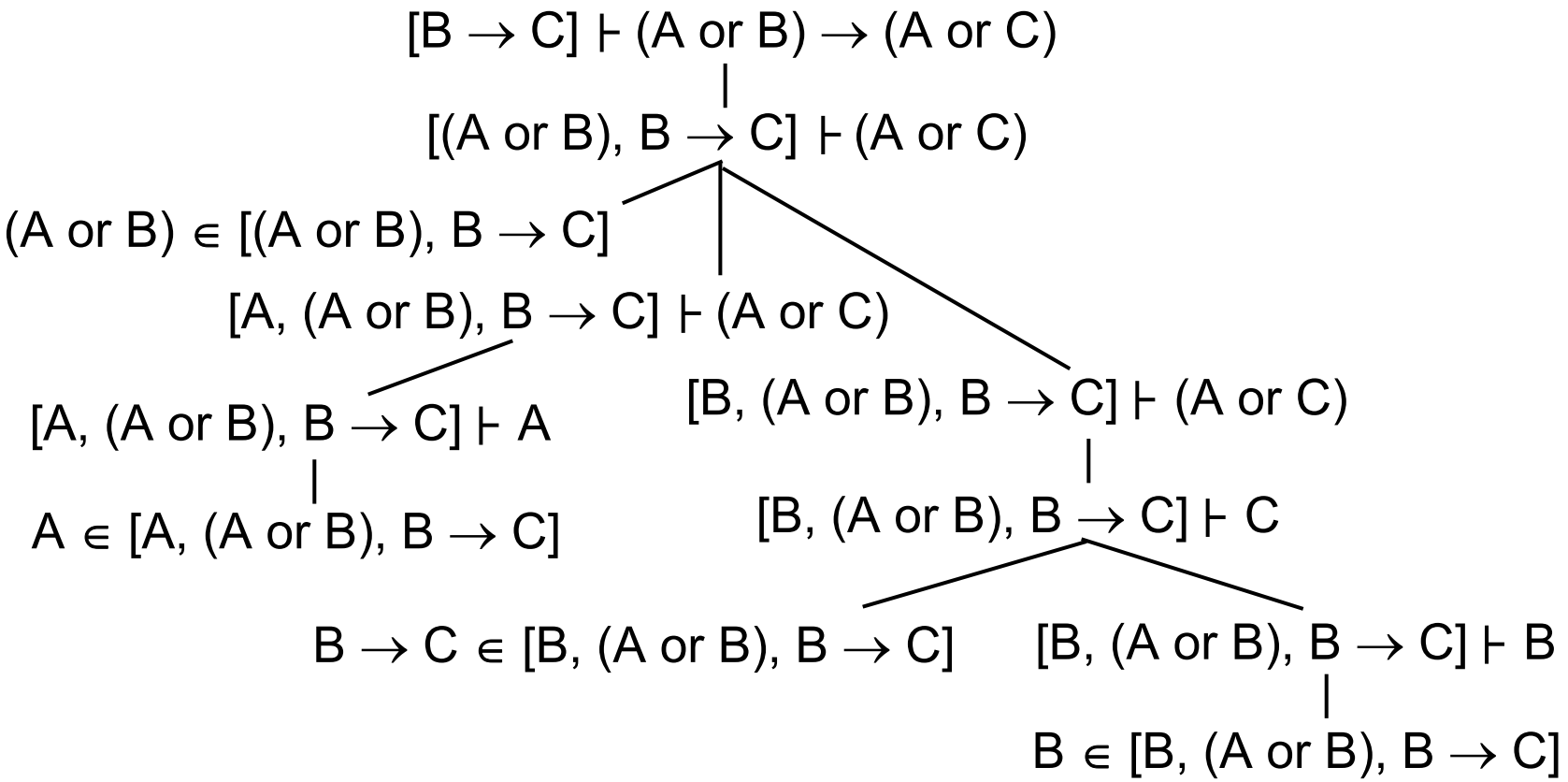
equivalent ?

not(I1 and not(i2))

Proof Rules

Proof	Sub-proofs		
$F \vdash A$	$A \in F$		
$F \vdash A$ and B	$F \vdash A$	$F \vdash B$	
$F \vdash A$ or B	$F \vdash A$		
$F \vdash A$ or B	$F \vdash B$		
$F \vdash C$	A or $B \in F$	$[A\|F] \vdash C$	$[B\|F] \vdash C$
$F \vdash B$	$A \to B \in F$	$F \vdash A$	
$F \vdash A \to B$	$[A\|F] \vdash B$		

Proofs (and Proof Trees)



Proof Rules for Negation

Proof	Sub-proofs
$F \vdash A$	$F \vdash \text{false}$
$F \vdash \text{not}(A)$	$[\, A \mid F \,] \vdash \text{false}$
$F \vdash B$	$\text{not}(A) \in F$ | $F \vdash A$
$F \vdash A$	$F \vdash \text{not}(\text{not}(A))$

The purpose of these rules is to provide positive evidence that an expression is false.

Negation as Failure

Replace <u>all</u> the previous negation rules with:

Proof	Sub-proofs
$F \vdash \text{not}(A)$	$F \nvdash A$

Where $F \nvdash A$ means a proof can't be found for A from F

This makes the closed world assumption that F contains all the axioms pertinent to the problem and that the proof search is complete.

Normal Forms

It is not strictly necessary to use all the logical operators. For example:

$A \rightarrow B$ is equivalent to not(A) or B
$A \rightarrow B$ is equivalent to not(A and not(B))
A or B is equivalent to not(not(A) and not(B))
and so on….

We can use this to convert any expression into An equivalent one with fewer operator types.

Conversion to Clausal Form

(a and not(b) $\rightarrow$ c) and a and not(c)

P $\rightarrow$ Q equivalent to not(P) or Q

(not(a and not(b)) or c) and a and not(c)

not(P and Q) equivalent to not(P) or not(Q)

(not(a) or not(not(b)) or c) and a and not(c)

not(not(P)) equivalent to P

(not(a) or b or c) and a and not(c)

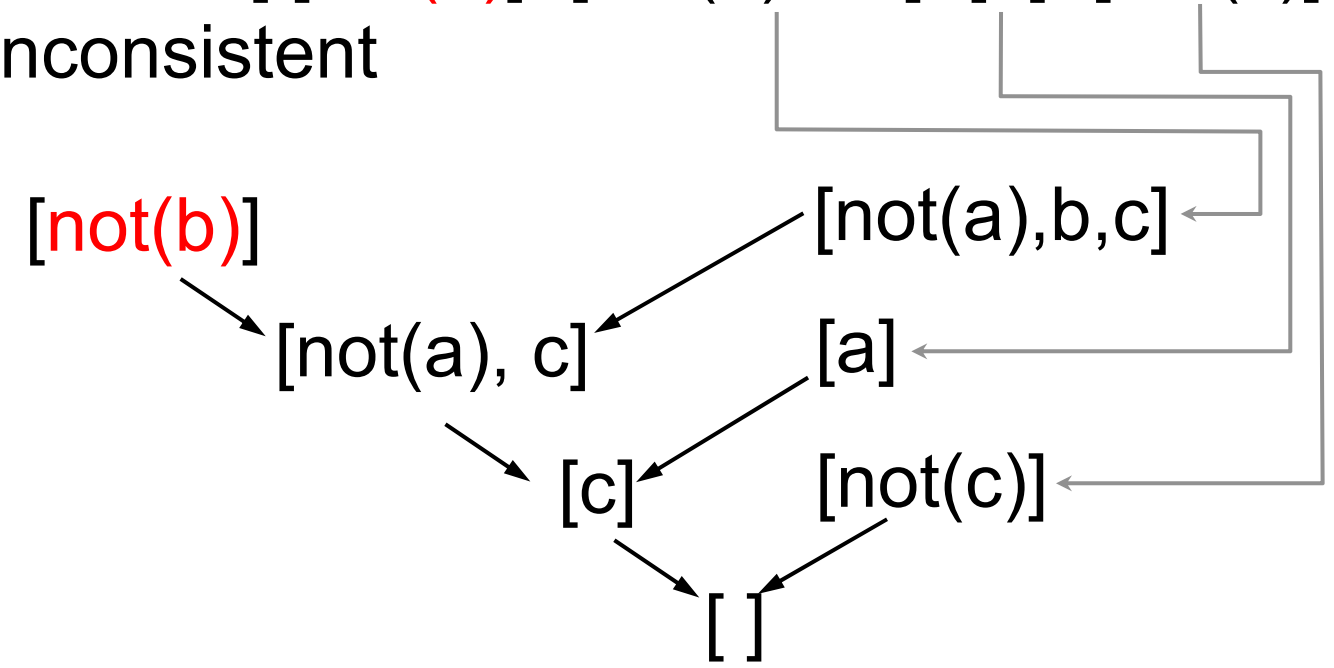
(P or …) and … to [[P,…],…]

[[not(a),b,c], [a], [not(c)]]

A Resolution Proof

To prove b from [[not(a),b,c], [a], [not(c)]]

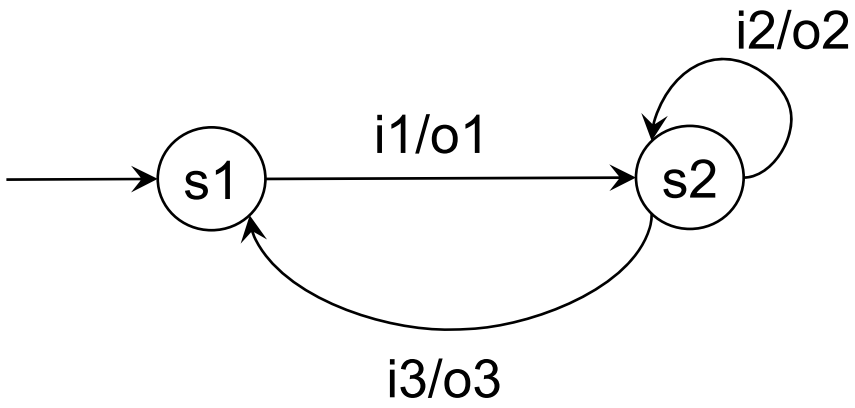
show that [[not(b)], [not(a),b,c], [a], [not(c)]]
is inconsistent



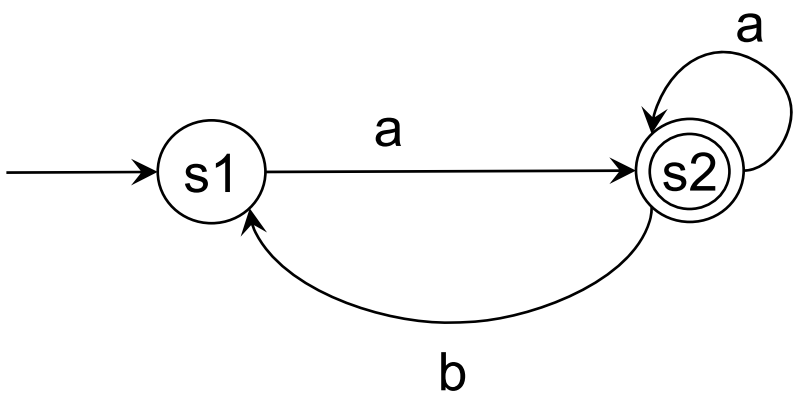
Temporal Proof Rules

Proof	Sub-proofs
(*S*,J) ├ A	access(J, *S*, *F*) *F* ├ A
(*S*,J) ├ A and B	(*S*,J) ├ A (*S*,J) ├ B
(*S*,J) ├ A or B	(*S*,J) ├ A
(*S*,J) ├ A or B	(*S*,J) ├ B
(*S*,J) ├ not(A)	(*S*,J) ⊬ A
(*S*,J) ├ next(A)	(*S*,J+1) ├ A
(*S*,J) ├ prev(A)	(*S*,J-1) ├ A
(*S*,J) ├ e_future(A)	(*S*,K) ├ A for some K > J
(*S*,J) ├ e_past(A)	(*S*,K) ├ B for some K < J
(*S*,J) ├ a_future(A)	(*S*,K) ├ A for all K > J
(*S*,J) ├ a_past(A)	(*S*,K) ├ A for all K < J

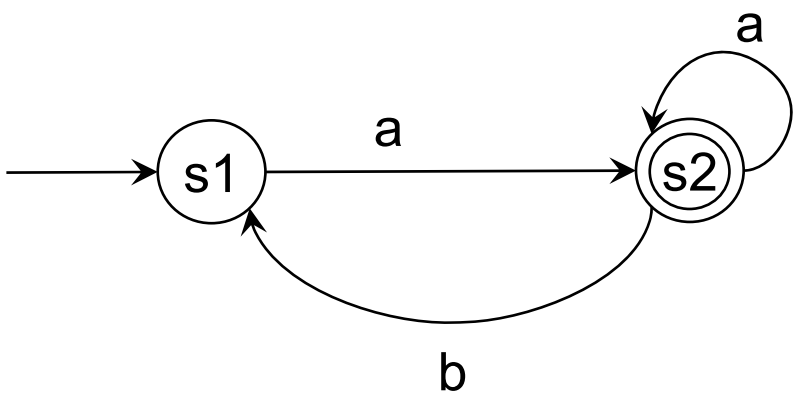
Transducer FSMs



Acceptor FSMs

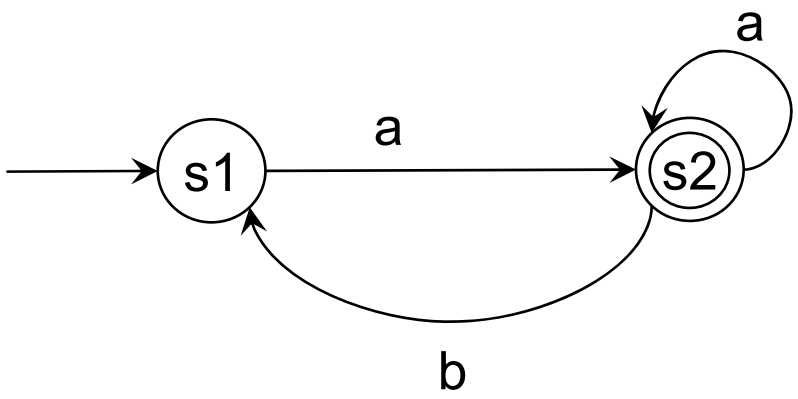


Traces



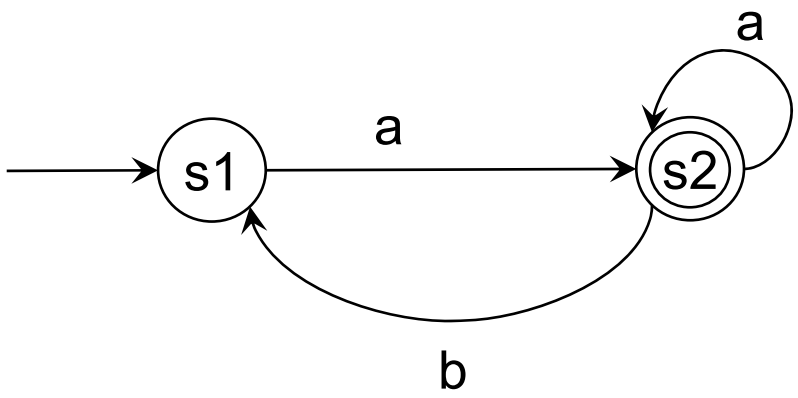
s1, a, s2, a, s2, b, s1, a, s2

Transition Function

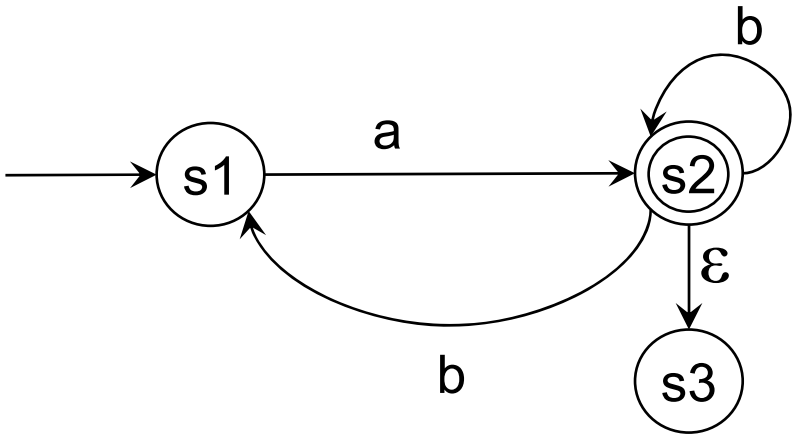


	a	b
s1	s2	-
s2	s2	s1

Deterministic v Nondeterministic

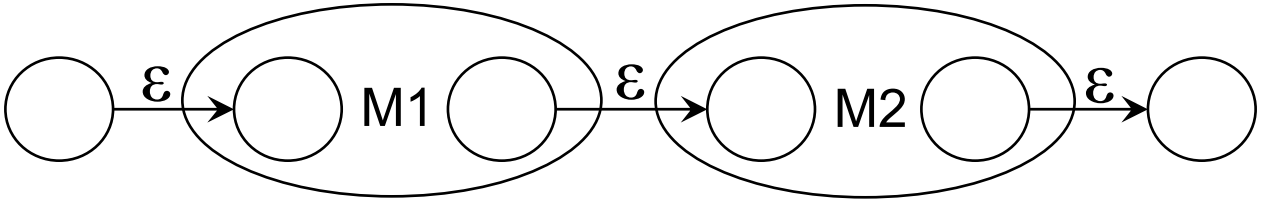


Deterministic

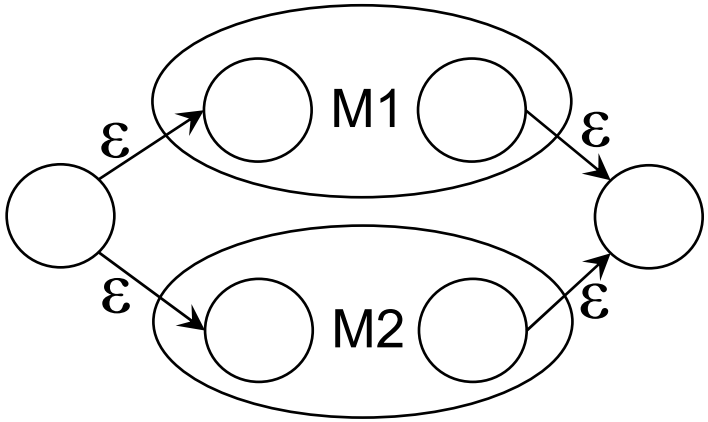


Nondeterministic

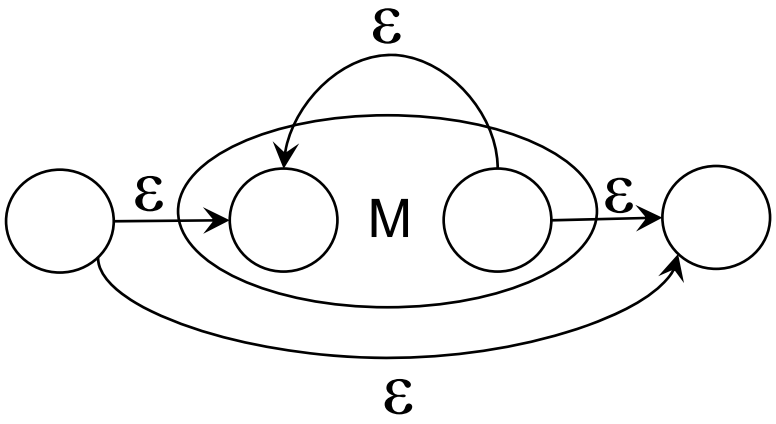
Structured FSM Design



sequence



choice



repetition

Regular Expressions

E_1E_2	Sequence
$E_1 \mid E_2$	Choice
E^*	Repetition

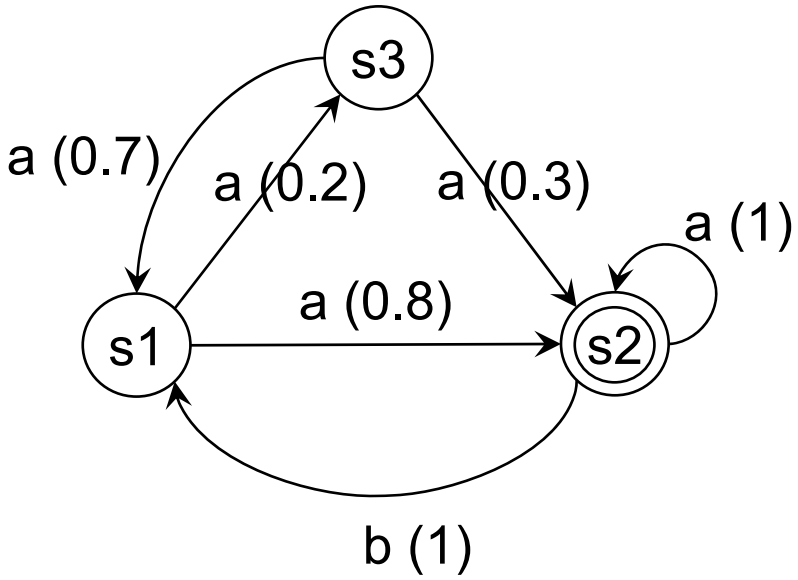
Half of Kleene's Theorem

For every regular expression we can build a FSM to accept the language defined by it.

Limits of FSMs and Regular Expressions

Some languages can't be defined by FSMs or regular expressions - languages that require us to count up to arbitrarily high numbers for example.

Probabilistic FSMs



Practical Things to Practice

- **Describing simple problems in logic (see problems tackled in each logic lecture)**
- **Describing simple systems as FSMs (see system engineering half of 6th computation lecture)**