Logic and Automata

Lecture 1
Monadic Second-Order Logic on Graphs and Strings

Sebastian Maneth
NICTA & UNSW

Logic Summer School - Canberra, December 2006

Outline

1. Introduction to
   Monadic Second-Order Logic (MSO) on graphs

2. Decidability Questions

3. Introduction to Finite-State Automata

1. Monadic Second-Order Logic

First-Order Logic plus quantification over *monadic predicates* (sets)

Logical Connectives $\wedge$ $\vee$ $\neg$ $\rightarrow$

Quantifiers $\forall x$ $\exists x$ $\forall X$ $\exists X$

Atomic Formulas $x=y$ $x \in X$

Predicates of the form $p(x_1, \ldots, x_n)$

Σ and Δ finite sets of symbols / labels ("alphabets")

Graphs over Σ and Δ x, y, z, … node variables
X, Y, Z, … node set variables

predicates	lab-σ(x)	node x has label σ
	γ-edge(x, y)	there is a γ-labeled edge from x to y

Notation GR(Σ, Δ) and MSO(Σ, Δ, $\emptyset$)

1. Monadic Second-Order Logic

First-Order Logic plus quantification over *monadic predicates* (sets)

Logical Connectives $\wedge$ $\vee$ $\neg$ $\rightarrow$

Quantifiers $\forall x$ $\exists x$ $\forall X$ $\exists X$

Atomic Formulas $x=y$ $x \in X$

Predicates of the form $p(x_1, \ldots, x_n)$

In these lectures, we only deal with (finite) graphs, or subclasses thereof.

More general (finite) *relational structures*

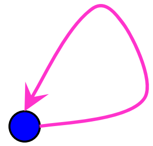
→ signature R (= finite set of relational symbols, each with an arity)
→ (finite) domain D
→ interpretation, maps relational symbol p of arity n
to a function of type $D^n \rightarrow \{ \text{true}, \text{false} \}$

Class of relational structures over R: STRUCT(R) MSO(R, $\emptyset$)

1. Monadic Second-Order Logic

First-Order Logic plus quantification over *monadic predicates* (sets)

Examples (here: no edge labels / Δ is singleton)

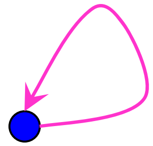
graphs without self-loops $(\forall x)\ \neg \text{edge}(x,x)$  (not allowed!)

1. Monadic Second-Order Logic

First-Order Logic plus quantification over *monadic predicates* (sets)

Examples

graphs without self-loops $(\forall\ x)\ \neg edge(x,x)$



(not allowed!)

---

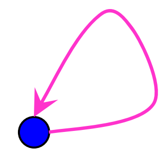
undirected graphs $(\forall\ x)(\forall\ y)\ edge(x,y) \leftrightarrow edge(x,y)$

1. Monadic Second-Order Logic

First-Order Logic plus quantification over *monadic predicates* (sets)

Examples

graphs without self-loops $(\forall x)\ \neg \mathrm{edge}(x,x)$



(not allowed!)

---

undirected graphs $(\forall x)(\forall y)\ \mathrm{edge}(x,y) \leftrightarrow \mathrm{edge}(x,y)$

---

circular graphs

$$\underbrace{\begin{array}{l} (\exists x_1)[\ \mathrm{edge}(x_1,x_1) \\ \vee\ (\exists x_2)[\ (\mathrm{edge}(x_1,x_2) \wedge \mathrm{edge}(x_2,x_1)) \\ \vee\ (\exists x_3)[\ (\mathrm{edge}(x_1,x_2) \wedge \mathrm{edge}(x_2,x_3) \wedge \mathrm{edge}(x_3,x_2)) \\ \ldots\ ]\ ]\ ] \end{array}}$$

$(\exists x_1)\ \mathrm{edge}^*(x_1, x_1)$

R^* = transitive closure of R ← *NOT* first-order!!

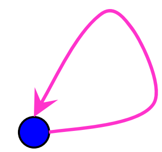
= everything reachable by R

1. Monadic Second-Order Logic

First-Order Logic plus quantification over *monadic predicates* (sets)

Examples

graphs without self-loops $(\forall x)\ \neg \text{edge}(x,x)$



(not allowed!)

undirected graphs $(\forall x)(\forall y)\ \text{edge}(x,y) \leftrightarrow \text{edge}(x,y)$

circular graphs

$$
\begin{aligned}
&(\exists x_1)[\ \text{edge}(x_1,x_1) \\
&\vee (\exists x_2)[\ (\text{edge}(x_1,x_2) \wedge \text{edge}(x_2,x_1)) \\
&\vee (\exists x_3)[\ (\text{edge}(x_1,x_2) \wedge \text{edge}(x_2,x_3) \wedge \text{edge}(x_3,x_2)) \\
&\ldots\ ]\ ]\ ]
\end{aligned}
$$

$$(\exists x_1)\ \text{edge}^*(x_1, x_1)$$

→#variables
→quantifier-depth

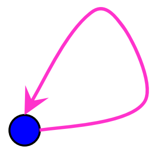
R^* = transitive closure of R ←*NOT* first-order!!
= everything reachable by R

1. Monadic Second-Order Logic

First-Order Logic plus quantification over *monadic predicates* (sets)

Examples

graphs without self-loops $(\forall x)\ \neg \text{edge}(x,x)$



(not allowed!)

---

undirected graphs $(\forall x)(\forall y)\ \text{edge}(x,y) \leftrightarrow \text{edge}(x,y)$

---

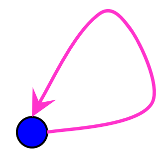
strings

1. Monadic Second-Order Logic

First-Order Logic plus quantification over *monadic predicates* (sets)

Examples

graphs without self-loops $(\forall x)\ \neg \text{edge}(x,x)$



(not allowed!)

undirected graphs $(\forall x)(\forall y)\ \text{edge}(x,y) \leftrightarrow \text{edge}(x,y)$

strings

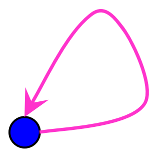
Not circular, outdegree is 1 or 0, and
and every node is reachable from one node

1. Monadic Second-Order Logic

First-Order Logic plus quantification over *monadic predicates* (sets)

Examples

graphs without self-loops $(\forall x)\ \neg \text{edge}(x,x)$



(not allowed!)

undirected graphs $(\forall x)(\forall y)\ \text{edge}(x,y) \leftrightarrow \text{edge}(x,y)$

strings

Not circular, outdegree is 1 or 0, and
and every node is reachable from one node

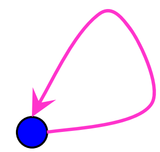
$$\neg(\exists x)\ \text{edge}^*(x,x)$$
$$\wedge\ (\forall x)(\forall y)(\forall z)[\ ((\text{edge}(x,y) \wedge \text{edge}(x,z)) \rightarrow y=z\ ]$$
$$\wedge\ (\exists x)(\forall y)\ \text{edge}^*(x,y)$$

1. Monadic Second-Order Logic

First-Order Logic plus quantification over *monadic predicates* (sets)

Examples

graphs without self-loops $(\forall x)\ \neg \mathrm{edge}(x,x)$



(not allowed!)

undirected graphs $(\forall x)(\forall y)\ \mathrm{edge}(x,y) \leftrightarrow \mathrm{edge}(x,y)$

strings

Not circular, outdegree is 1 or 0
and every node is reachable from one node

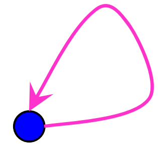
binary trees?

1. Monadic Second-Order Logic

First-Order Logic plus quantification over *monadic predicates* (sets)

Examples

graphs without self-loops $(\forall x)\ \neg \mathrm{edge}(x,x)$



(not allowed!)

---

undirected graphs $(\forall x)(\forall y)\ \mathrm{edge}(x,y) \leftrightarrow \mathrm{edge}(x,y)$

---

strings

Not circular, outdegree is 1 or 0
and every node is reachable from one node

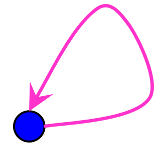
binary trees
outdegree is 2 or 1 or 0

1. Monadic Second-Order Logic

First-Order Logic plus quantification over *monadic predicates* (sets)

Examples

graphs without self-loops $(\forall x)\ \neg \mathrm{edge}(x,x)$



(not allowed!)

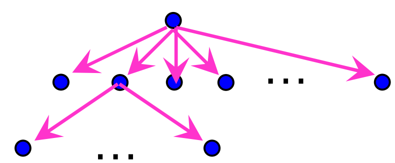
undirected graphs $(\forall x)(\forall y)\ \mathrm{edge}(x,y) \leftrightarrow \mathrm{edge}(x,y)$

strings

Not circular, outdegree is 1 or 0
and every node is reachable from one node

binary trees
outdegree is 2 or 1 or 0

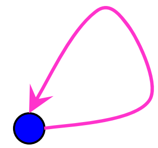
unranked trees?



1. Monadic Second-Order Logic

First-Order Logic plus quantification over *monadic predicates* (sets)

Examples

graphs without self-loops $(\forall x)\ \neg \text{edge}(x,x)$  (not allowed!)

---

undirected graphs $(\forall x)(\forall y)\ \text{edge}(x,y) \leftrightarrow \text{edge}(x,y)$ 

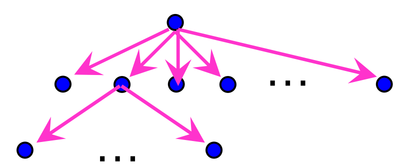
---

strings

Not circular, outdegree is 1 or 0
and every node is reachable from one node

binary trees
outdegree is 2 or 1 or 0

unranked trees
indegree is 1 or 0

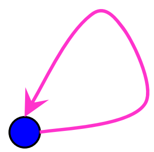


1. Monadic Second-Order Logic

First-Order Logic plus quantification over *monadic predicates* (sets)

Examples

graphs without self-loops $(\forall x)\ \neg edge(x,x)$



(not allowed!)

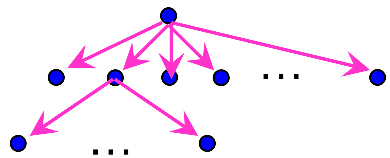
undirected graphs $(\forall x)(\forall y)\ edge(x,y) \leftrightarrow edge(x,y)$

strings

Not circular, outdegree is 1 or 0
and every node is reachable from one node

binary trees
outdegree is 2 or 1 or 0

unranked trees
indegree is 1 or 0



dags? …

1. Monadic Second-Order Logic

First-Order Logic plus quantification over *monadic predicates* (sets)

Example (First-Order)

"*friendship graph*": no labels, undirected graphs, no self-loops

every two distinct nodes have a unique common neighbour

$$(\forall x)\ (\forall y)\ ((\neg\ x=y) \rightarrow (\exists z)\ (z\neq x \wedge z\neq y \wedge \underbrace{edge(z,x) \wedge edge(z,y)}_{cnbor(z,x,y)} \wedge (\forall u)\ u\neq z \rightarrow \neg cnbor(u,x,y)))$$

1. Monadic Second-Order Logic

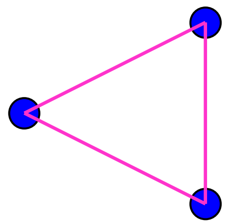
First-Order Logic plus quantification over *monadic predicates* (sets)

Example (First-Order)

"*friendship graph*": no labels, undirected graphs, no self-loops

every two distinct nodes have a unique common neighbour

$$(\forall x)\ (\forall y)\ ((\neg\ x{=}y) \rightarrow (\exists z)\ (z \neq x \land z \neq y \land \underbrace{\text{edge}(z,x) \land \text{edge}(z,y)}_{\text{cnbor}(z,x,y)} \land (\forall u)\ u \neq z \rightarrow \neg \text{cnbor}(u,x,y)))$$



A friendship graph

1. Monadic Second-Order Logic

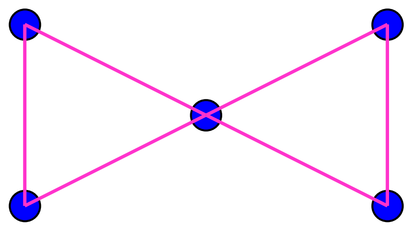
First-Order Logic plus quantification over *monadic predicates* (sets)

Example (First-Order)

"*friendship graph*": no labels, undirected graphs, no self-loops

every two distinct nodes have a unique common neighbour

$$(\forall x)\ (\forall y)\ ((\neg\ x=y) \ \rightarrow\ (\exists z)\ (z \neq x \wedge z \neq y \wedge \underbrace{edge(z,x) \wedge edge(z,y)}_{cnbor(z,x,y)} \wedge (\forall u)\ u \neq z \rightarrow \neg cnbor(u,x,y)))$$



A friendship graph

1. Monadic Second-Order Logic

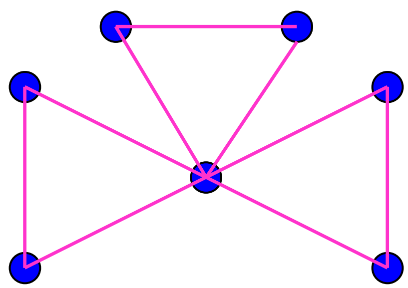
First-Order Logic plus quantification over *monadic predicates* (sets)

Example (First-Order)

"*friendship graph*": no labels, undirected graphs, no self-loops

every two distinct nodes have a unique common neighbour

$(\forall\ x)\ (\forall\ y)\ ((\neg\ x{=}y) \ \rightarrow\ (\exists\ z)\ (z{\neq}x \wedge z{\neq}y$
$\qquad \wedge\ \underbrace{edge(z,x) \wedge edge(z,y)}_{cnbor(z,x,y)} \wedge (\forall\ u)\ u{\neq}z \rightarrow \neg cnbor(u,x,y)))$



A friendship graph

1. Monadic Second-Order Logic

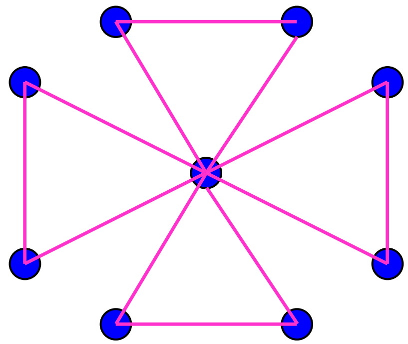
First-Order Logic plus quantification over *monadic predicates* (sets)

Example (First-Order)

"*friendship graph*": no labels, undirected graphs, no self-loops

every two distinct nodes have a unique common neighbour

$(\forall\ x)\ (\forall\ y)\ ((\neg\ x{=}y) \rightarrow (\exists\ z)\ (z{\neq}x \wedge z{\neq}y$
$\wedge\ \underbrace{edge(z,x) \wedge edge(z,y)}_{cnbor(z,x,y)} \wedge (\forall\ u)\ u{\neq}z \rightarrow \neg cnbor(u,x,y)))$



A friendship graph

windmills…

1. Monadic Second-Order Logic

First-Order Logic plus quantification over *monadic predicates* (sets)

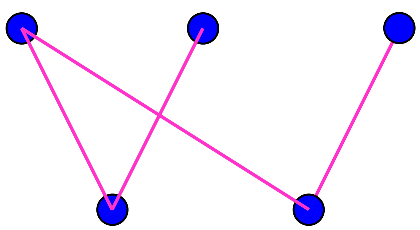
Example Second-Order

bipartite graphs: no labels, undirected graphs, no self-loops

set of nodes can be partitioned into sets X, Y such that
an edge between x and y implies that $x \in X$ and $y \in Y$ (or vice versa)

$$part(X, Y) \equiv (\forall z)\ (z \in X \lor z \in Y) \land \neg(z \in X \land z \in Y)$$

$$bipartite \equiv (\exists X)(\exists Y)\ part(X, Y) \land (\forall u)(\forall v)\ edge(u,v) \rightarrow (u \in X \land v \in Y) \lor (u \in Y \land v \in X)$$



A bipartite graph g

Notation: $g \models$ bipartite

Notation

MSO formula ϕ and graph g:

→ $g \models \phi$ means that ϕ holds for g (g is a model of ϕ)

→ ϕ is *closed*, if it contains no free variables

An MSO formula with one free variable describes a set. MSO(Σ,Δ,{x})

An MSO formula with two free variables describes a binary relation.

Etc.

1. Monadic Second-Order Logic (MSO)

First-Order Logic plus quantification over *monadic predicates* (sets)

Important Tool: Transitive Closure

Lemma
Let S be a structure which has a binary relation R.
The transitive closure of R is defined by the following MSO formula:

$$
\begin{aligned}
&(\forall X) \\
&[\\
&\quad \underbrace{[(\forall y)(\forall z)(y \in X \wedge R(y,z) \rightarrow z \in X)}_{X \text{ is ``R-closed''}} \wedge \underbrace{(\forall u)(R(x_1, u) \rightarrow u \in X)}_{\text{For transitive, reflexive closure, simply: } x_1 \in X}] \rightarrow x_2 \in X \\
&]
\end{aligned}
$$

(this formula has two *free variables* x_1, x_2)

$$\text{edge}^*(x,y) \equiv (\forall X)[\, [\, (\forall u)(\forall v)(u \in X \wedge \text{edge}(u,v) \rightarrow v \in X) \wedge x \in X \,] \rightarrow y \in X \,]$$

1. Monadic Second-Order Logic (MSO)

First-Order Logic plus quantification over *monadic predicates* (sets)

Example Second-Order

EVEN length strings

Can you construct MSO formula ψ such that

$g \models \psi \quad \Leftrightarrow \quad$ g is string with EVEN number of a's

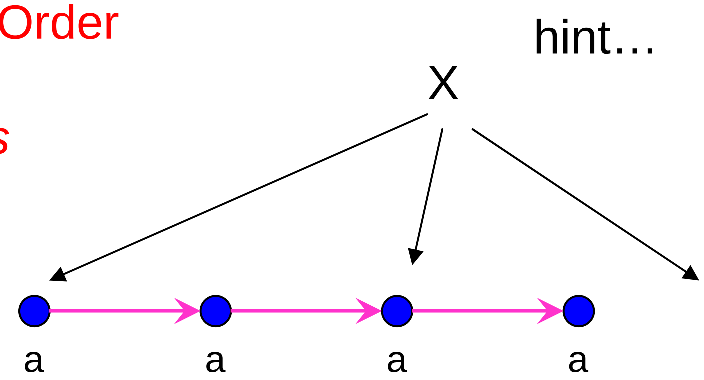
?

1. Monadic Second-Order Logic (MSO)

First-Order Logic plus quantification over *monadic predicates* (sets)

Example Second-Order

EVEN length strings



Can you construct MSO formula ψ such that

$g \models \psi \quad \Leftrightarrow \quad$ g is string with EVEN number of a's

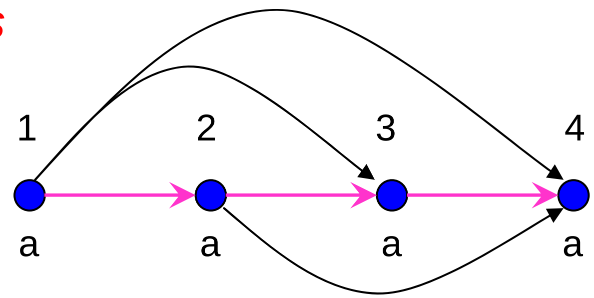
?

1. Monadic Second-Order Logic (MSO)

First-Order Logic plus quantification over *monadic predicates* (sets)

Example Second-Order

EVEN length strings



Can you construct MSO formula ψ such that

$g \models \psi \quad \Leftrightarrow$ g is string with EVEN number of a's

?

Btw instead of edge = suc = { (1,2), (2,3), (3,4) }

you can also use < = { (1,2), (1,3), (1,4), (2,3), (2,4), (3,4) }

Why? Because TC is in MSO! (in FO, edge and < give different expressivity!)

1. Monadic Second-Order Logic (MSO)

First-Order Logic (FO) plus quantification over *monadic predicates* (sets)

Examples of MSO definable graphs (properties)

→ connected

→ undirected Hamiltonian (= has Hamiltonian cycle)

→ k-colorable

→ string / tree / dag

→ planar (use Kuratowski's Theorem)

…

1. Monadic Second-Order Logic (MSO)

First-Order Logic (FO) plus quantification over *monadic predicates* (sets)

Questions

→ can you express bipartite / EVEN in FO?

→ how to prove that something *cannot* be expressed in FO/MSO?

Examples of graph properties NOT definable in MSO

→ two sets X and Y have equal cardinality

→ in a directed graph, X is the set of nodes of a path from x to y

→ a directed graph is Hamiltonian

2. Decidability Questions

R: finite set of relation symbols
L: class of closed formulas expressing properties of the structures in STRUCT(R)
Let $C \subseteq$ STRUCT(R).

L-Theory of C

$$Th_L(C) := \{ \phi \in L \mid g \models \phi \text{ for every } g \in C \}.$$

If $Th_L(C)$ is recursive then "the L-theory of C is decidable".

L-satisfiability problem for C

Decide whether a formula belongs to

$$Sat_L(C) := \{ \phi \in L \mid g \models \phi \text{ for some } g \in C \}.$$

2. Decidability Questions

R: finite set of relation symbols
L: class of closed formulas expressing properties of the structures in STRUCT(R)
Let C $\subseteq$ STRUCT(R).

L-Theory of C

$$Th_L(C) := \{ \phi \in L \mid g \models \phi \text{ for every } g \in C \}.$$

If $Th_L(C)$ is recursive then "the L-theory of C is decidable".

L-satisfiability problem for C

Decide whether a formula belongs to

$$Sat_L(C) := \{ \phi \in L \mid g \models \phi \text{ for some } g \in C \}.$$

If L is closed under negation: L-theory of C decidable $\Leftrightarrow$ L-sat. prob. decidable

2. Decidability Questions

$Th_L(C)$:= { $\phi \in L$ | $g \models \phi$ for every $g \in C$ }.

If $Th_L(C)$ is recursive then "the L-theory of C is decidable".

Theorem [Trakhtenbrot 1950]

The first-order theory of the class of finite graphs is undecidable.

That is:

There is NO ALGORITHM, that can tell for a given formula ϕ, whether or not ϕ holds for all finite graphs.

Corollary

The MSO theory of the class of finite graphs is undecidable.

2. Decidability Questions

$Sat_L(C)$:= { $\phi \in L$ | $g \models \phi$ for some $g \in C$ }.

Theorem
If C is a class of graphs containing infinitely many square grids,
Then the MSO-satisfiability problem for C is undecidable.

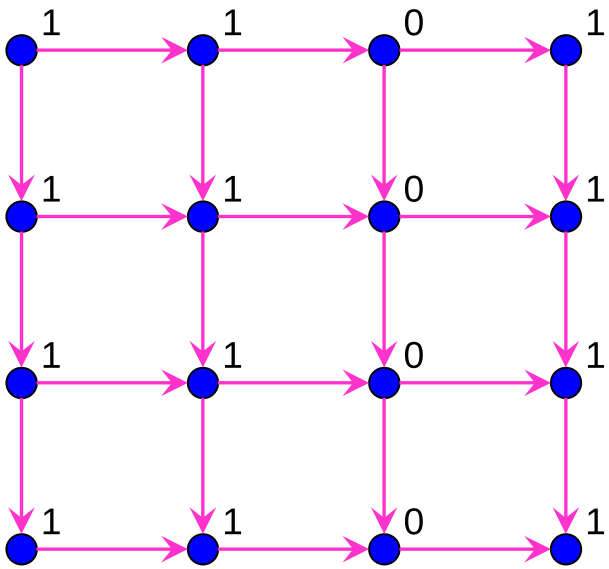
that is,

there is NO ALGORITHM, that can tell for a given formula ψ,
whether or not there is a $g \in C$ with $g \models \psi$.

Theorem [Seese 1975]
If C is a class of graphs containing infinitely many square grids,
Then the MSO-satisfiability problem for C is undecidable.

(given ψ, is there $g \in C$ with $g \models \psi$?)

Proof.

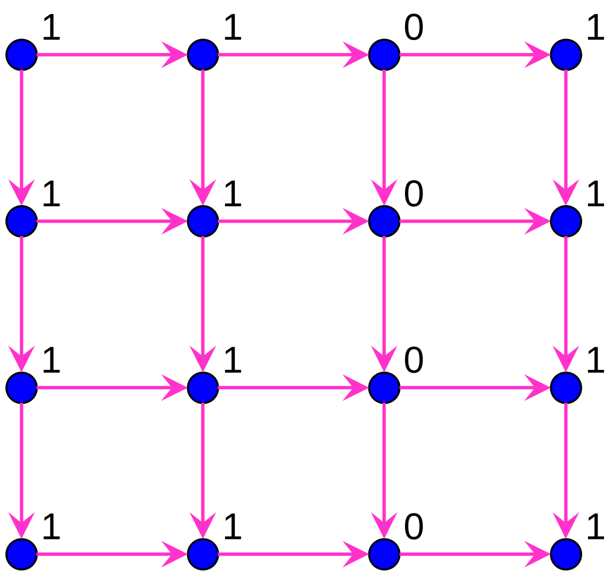


a square grid

Theorem [Seese 1975]
If C is a class of graphs containing infinitely many square grids,
Then the MSO-satisfiability problem for C is undecidable.

(given ψ, is there $g \in C$ with $g \models \psi$?)

Proof.



a square grid

there is $g \in C$ with $g \models \psi$

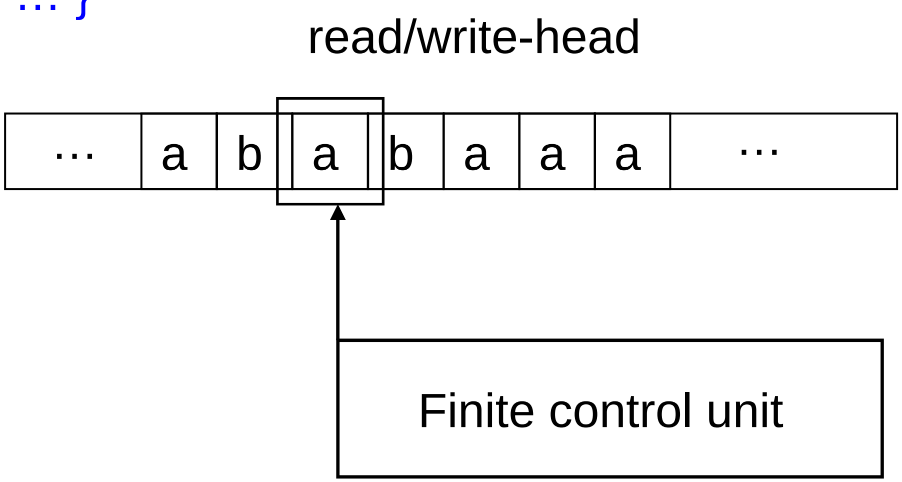
$\Leftrightarrow$ Turing Machine M halts on input w

A **Turing Machine M** consists of

→ finite set of states $Q = \{q, q', \ldots\}$
→ finite set of tape symbols $S = \{a, b, \ldots\}$
→ finite set of instruction of the form

$$\delta(q, a) = \{ (q', b, x), .. \}$$

where $x \in \{ \text{Left, Right, Stay} \}$.



Such an instruction means:

If **M** is in state q, and its read/write-head is on a tape position labeled by a, then the machine can
(1) change into state q'
(2) replace a by b, and
(3) make the move x (stay or move one position to Left/Right)

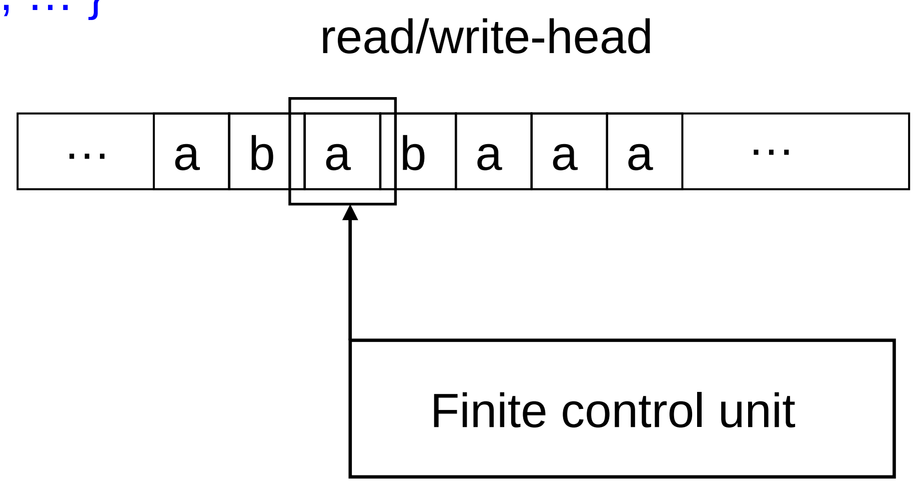
(**M** halts if no instruction can be applied)

A **Turing Machine M** consists of

→ finite set of states Q = {q, q', … }
→ finite set of tape symbols S = { a, b, … }
→ finite set of instruction of the form

$\delta(q, a) = \{ (q', b, x), .. \}$

where $x \in$ { Left, Right, Stay }.



Theorem [Turing 1936]

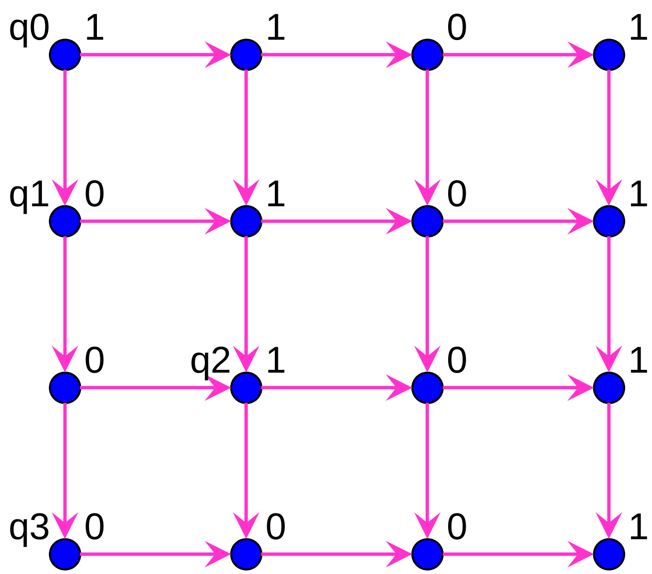
Given a TM M, it is undecidable, i.e., there is NO ALGORITHM to tell whether or not M halts for every/some input.

Proof, easy! (you've seen it…)

Theorem [Seese 1975]
If C is a class of graphs containing infinitely many square grids,
Then the MSO-satisfiability problem for C is undecidable.

(given ψ, is there $g \in C$ with $g \models \psi$?)

Proof.



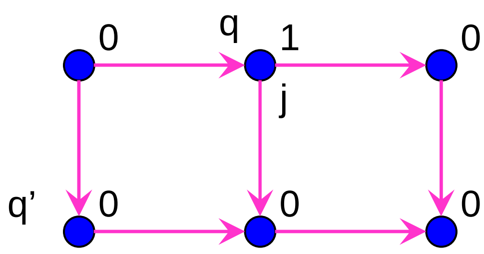
a square grid

there is $g \in C$ with $g \models \psi$

$\Leftrightarrow$ Turing Machine M halts on some input

Given M, construct MSO formula ψ
such that $g \models \psi$ if and only if

→ g is a square grid (non-trivial)
→ every line is a valid configuration
→ if a position j is labeled by a state q
and there is a next line, then



domino

is according to M

E.g.,
$\delta(q,1) = (q',0,L)$

2. Decidability Questions

We are looking for classes C of graphs such that the
MSO-theory of C is decidable.

(All we know: C *must not* contain infinitely many square grids..)

2. Decidability Questions

We are looking for classes C of graphs such that the
MSO-theory of C is decidable.

(All we know: C *must not* contain infinitely many square grids..)

STRING = class of string graphs

Theorem [Büchi1960, Elgot1961]

The MSO-theory of STRING is decidable.

2. Decidability Questions

We are looking for classes C of graphs such that the
MSO-theory of C is decidable.

(All we know: C *must not* contain infinitely many square grids..)

STRING = class of string graphs

Theorem [Büchi1960, Elgot1961]

The MSO-theory of STRING is decidable.

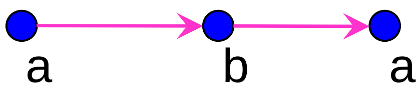
TREE = class of (binary) tree graphs
→The MSO-theory of TREE is decidable! [Thatcher/Wright1968]

BTW = class of graphs of bounded tree width
→The MSO-theory of BTW is decidable! [Courcelle1988]

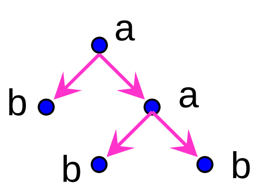
MSO & Formal Language Theory

Brief History

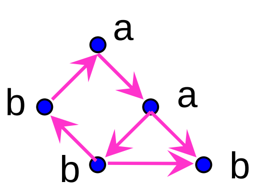
MSO definable string languages =
regular languages [Büchi1960, Elgot1961]



MSO definable tree languages =
regular tree languages [Doner1970, Thatcher/Wright1968]



MSO definable graph languages ⊆
recognizable graph languages [Courcelle1990]



3. Finite-State Automata

Fix an alphabet (= finite set) Σ of symbols.

A *string / word over* Σ is a sequence $a_1\ a_2 \ldots a_n$ of symbols in Σ.

A (formal) language (over Σ) is a set of strings over Σ.

Notation ε = the empty string
Σ^* = set of all strings over Σ

$\Sigma = \{ a, b \}$

abaa $\in \Sigma^*$, that is, abaa is a string over Σ

L1 = { a, aa, aaa, … } is a formal language over Σ.

L2 = $\{ w \in \Sigma^* \mid$ first letter of w is an a $\}$
is a formal language over Σ.

3. Finite-State Automata

→ define a formal language using a Turing Machine M. E.g.,

$L(M) := \{ w \in \Sigma^* \mid M \text{ halts on input } w \}$

Note: TM has unboundedly long tape
& can write arbitrary long computations

$\delta(q, a) = \{ (q', b, x) \}$

3. Finite-State Automata

→ define a formal language using a Turing Machine M. E.g.,

$L(M) := \{ w \in \Sigma^* \mid M \text{ halts on input } w \}$

Note: TM has unboundedly long tape
& can write arbitrary long computations

$\delta(q, a) = \{ (q', b, x) \}$

Consider TM's that canNOT write, and strictly move right in every transition.

$\delta(q, a) = \{ (q', a, R) \}$ or simply $\delta(q, a) = \{ q' \}$.

3. Finite-State Automata

→ define a formal language using a Turing Machine M. E.g.,

L(M) := $\{ w \in \Sigma^* \mid$ M halts on input w $\}$

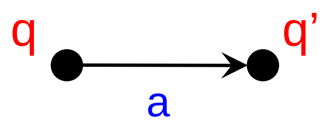
Note: TM has unboundedly long tape
& can write arbitrary long computations

$\delta(q, a) = \{ (q', b, x) \}$

Consider TM's that canNOT write, and strictly move right in every transition.

$\delta(q, a) = \{ (q', a, R) \}$ or simply $\delta(q, a) = \{ q' \}$.

➔ FINITE-STATE AUTOMATA



FINITE-STATE AUTOMATON

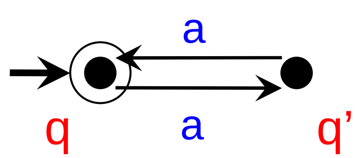
$$\mathbf{A} = (Q, \Sigma, q_0, F, \delta)$$

Q	finite set of states
Σ	input alphabet
$q_0 \in Q$	initial state
$F \subseteq Q$	set of final states
$\delta: Q \times \Sigma \rightarrow P(Q)$	transition function

---

Example $A = (\{q,q'\}, \{a\}, q, \{q\}, \delta)$ with $\delta(q,a) = \{q'\}$

$\delta(q',a) = \{q\}$



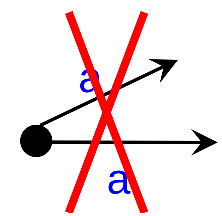
Extend δ to $Q\times\Sigma^*$

$$\underline{\delta}(q, a\,w) = \underline{\delta}(\delta(q, a), w)$$

$$\underline{\delta}(q, w\,a) = \delta(\underline{\delta}(q, w), a)$$

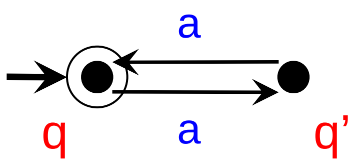
Language accepted by **A** $\quad L(\mathbf{A}) = \{ \ w\in\Sigma^* \ | \ \underline{\delta}(q_0, w) \cap F \neq \emptyset \ \}$

A is deterministic, if $|\delta(q,a)| \leq 1$ for all $q\in Q$ and $a\in\Sigma$.



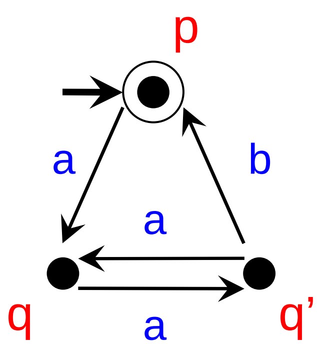
More FINITE-STATE AUTOMATA

A1



$L(A1) = \{ \varepsilon, aa, aaaa, a^6, a^8, \ldots \}$
$= \{ a^n \mid n \geq 0 \text{ is even} \} = (aa)^*$

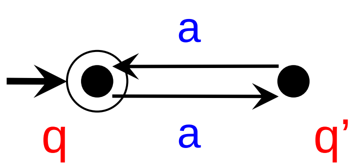
A2



L(A2) =

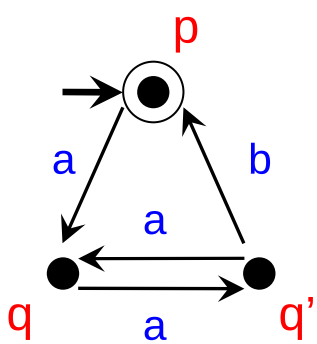
More FINITE-STATE AUTOMATA

A1



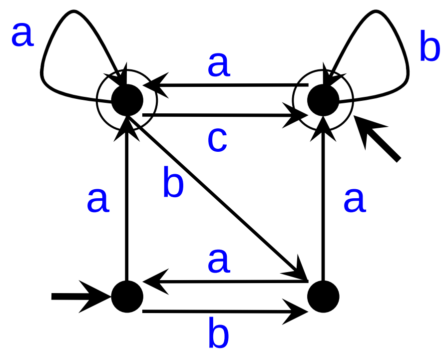
L(A1) = { ε, aa, aaaa, a^6, a^8, … }
= { a^n | $n \geq 0$ is even } = (aa)*

A2



L(A2) = { ε, aab, … } = ((aa)*b)*

A3



L(A3) = … ?

FINITE-STATE AUTOMATA

→ can be *determinized*   (but, exponential blow-up ☹)
    (and, ∃ unique *minimal deterministic* automaton for every automaton)
→ allow constant memory scanning (→ stream-processing ☺)
→ accept the regular languages

Regular Languages have nice properties

Characterized by
→ Finite-State Automata
→ Regular Expressions
→ Regular Grammars
→ MSO sentences, …

Closed under
→ Intersection
→ Union
→ Complement, …

Decidable emptiness/equivalence/
inclusion etc

FINITE-STATE AUTOMATA

→ can be *determinized* (but, exponential blow-up ☹)
(and, ∃ unique *minimal deterministic* automaton for every automaton)
→ allow constant memory scanning (→ stream-processing ☺)
→ accept the regular languages

Regular Languages have nice properties

Characterized by
- → Finite-State Automata
- → Regular Expressions
- → Regular Grammars
- → MSO sentences, …

Closed under
- → Intersection
- → Union
- → Complement, …

Decidable emptiness/equivalence/inclusion etc

Tool to prove NON-regularity

→ pumping lemma

$(\forall L)(\exists n)$: for all words x longer than n,
$x = u\,v\,w$, for words u,v,w
and $u\,(v)^k\,w \in L$ for all $k \geq 0$.

FINITE-STATE AUTOMATA can be *determinized*

Lemma For every FSA A there effectively exists a deterministic FSA det(A) such that L(det(A)) = L(A).

A = $(Q, \Sigma, q_0, F, \delta)$ arbitrary finite-state automaton

“powerset / subset construction”

Det(A) = $(P(Q), \Sigma, \{ q_0 \}, \{ S \in P(Q) \mid S \cap F \neq \emptyset \}, \delta')$

for $S \in P(Q)$ and $a \in \Sigma$: $\delta'(S, a) = \{ \delta(p,a) \mid p \in S \}$

Lets prove that L(Det(A)) = L(A).

Induction on the length of w. (w → w a)

(1) If $q' \in \underline{\delta}(q, w)$, then for all S with $q \in S$: $q' \in \underline{\delta'}(S, w)$

(2) If $S' = \delta'(S, w)$, then for every $q \in S'$ there is $p \in S$ such that $\delta(p, w) = q$

→ In the worst case, size of Det(A) = O($2^{size(A)}$)

Stronger

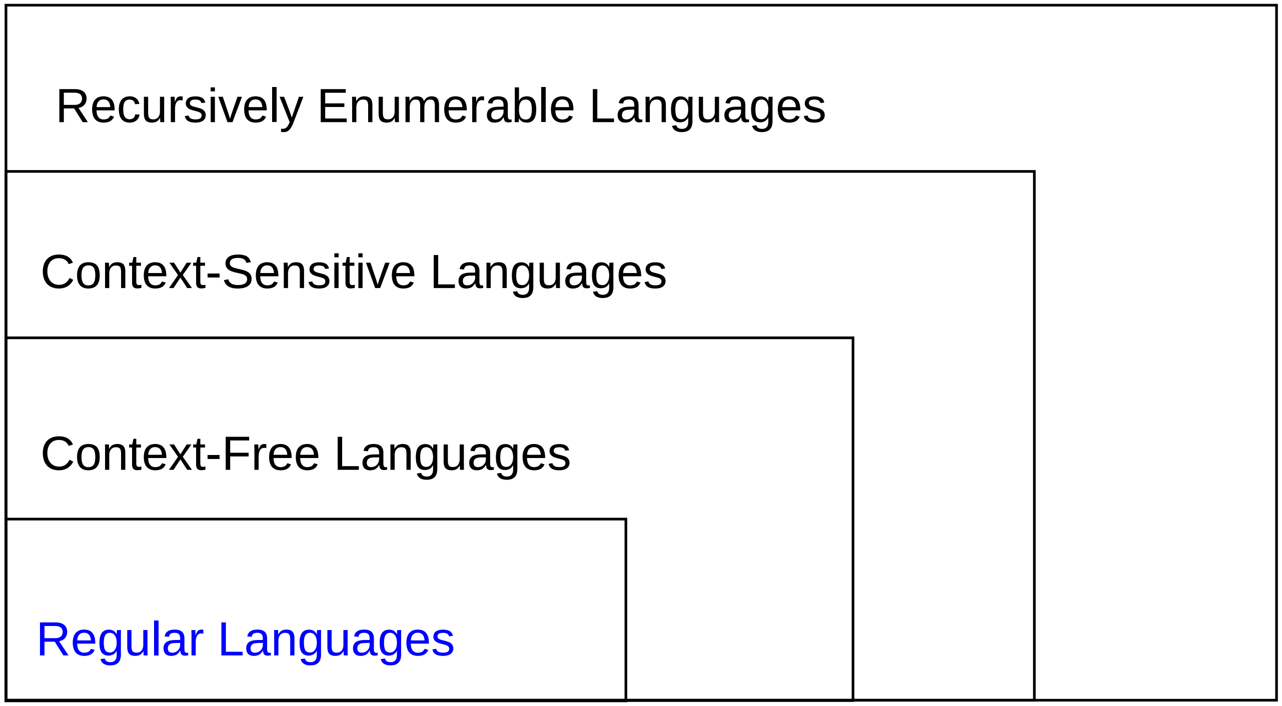
FS automata are exponentially more succinct
than deterministic FS automata.

There is an automaton A of size n such that *any*
(or, alternatively, the minimal one..)
deterministic automaton for L(A) has size O(2^n).

QUESTION How does such an A look like?

Formal Language Primer

Chomsky Hierarchy

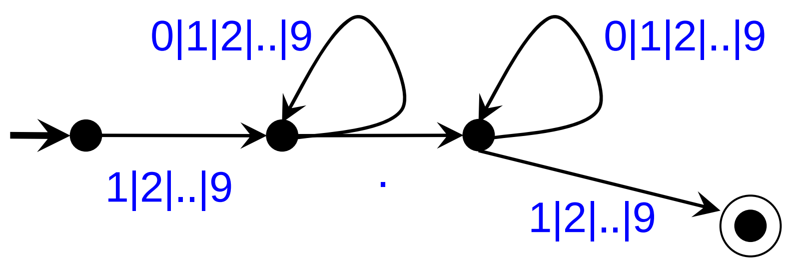


---

a *regular grammar*

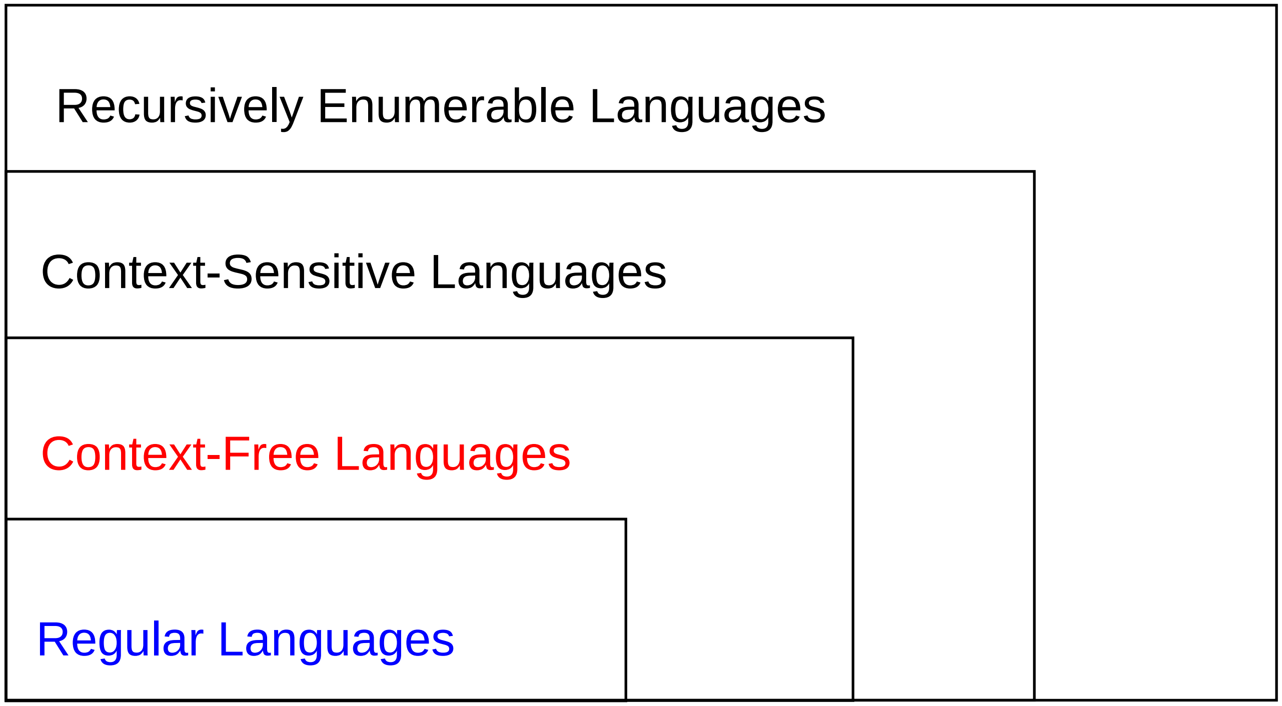
S → 1A | 2A | … | 9A
A → 0A | 1A | … | 9A | ε | .B
B → 0B | 1B | … | 9B | C
C → 1C | 2C | … | 9C | ε

FINITE-STATE AUTOMATON



Formal Language Primer

Chomsky Hierarchy



---

a *regular grammar*

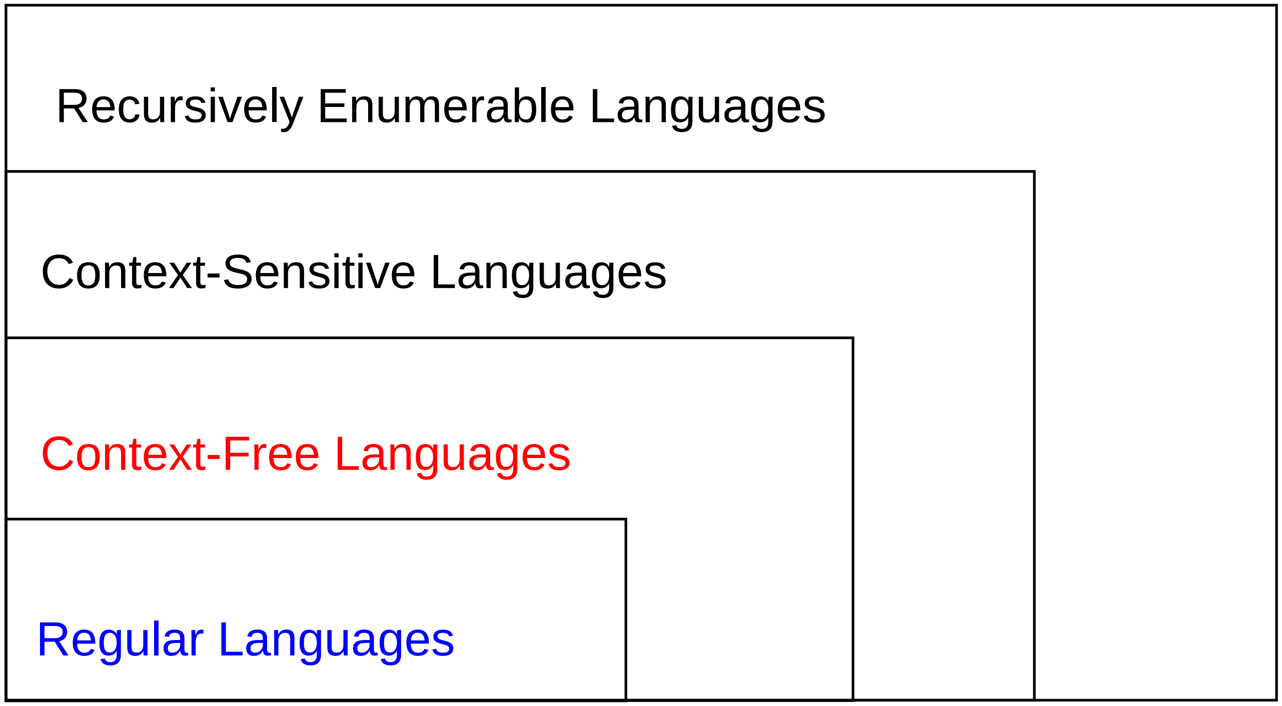
S → 1A | 2A | … | 9A
A → 0A | 1A | … | 9A | ε | .B
B → 0B | 1B | … | 9B | C
C → 1C | 2C | … | 9C | ε

a *context-free grammar*

S → aSb
S → ε

Formal Language Primer

Chomsky Hierarchy



---

a *regular grammar*

S → 1A | 2A | … | 9A
A → 0A | 1A | … | 9A | ε | .B
B → 0B | 1B | … | 9B | C
C → 1C | 2C | … | 9C | ε

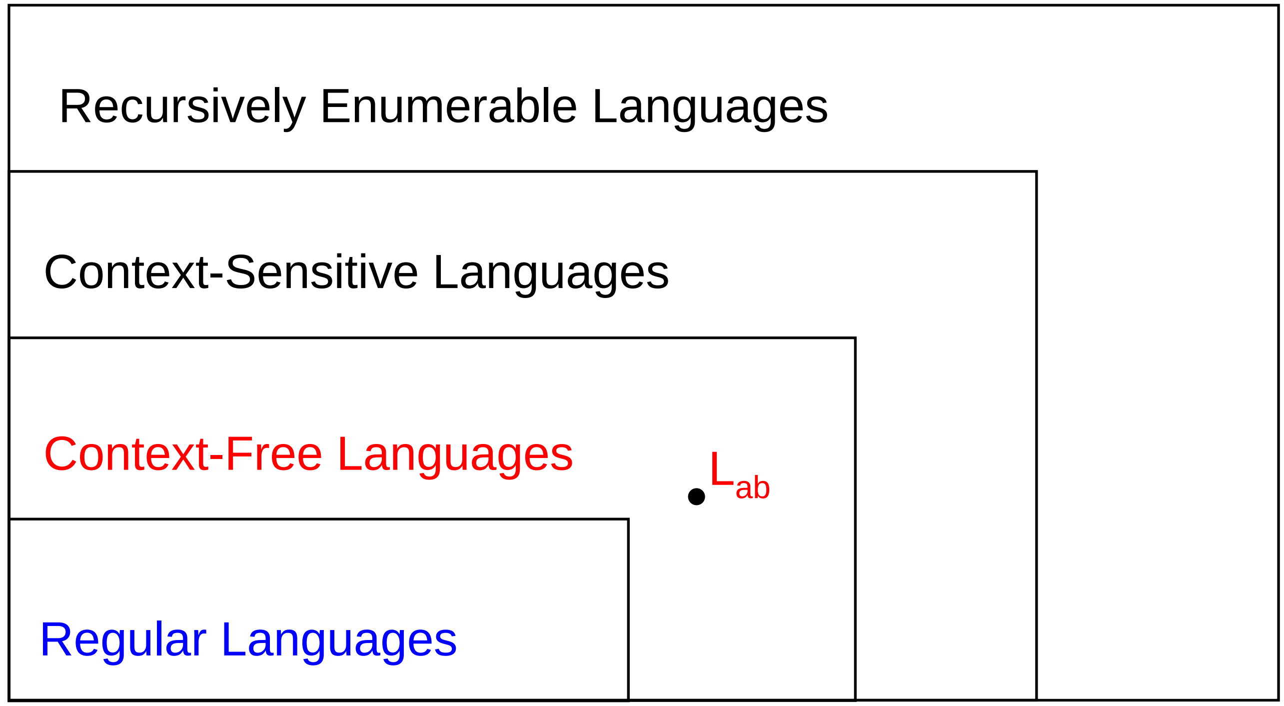
a *context-free grammar*

S → aSb
S → ε

$$L_{ab} = \{ \underbrace{a \ldots a}_{n}\underbrace{b \ldots b}_{n} \mid n \geq 0 \}$$

Formal Language Primer

Chomsky Hierarchy



---

a *regular grammar*

$S \rightarrow 1A \mid 2A \mid \ldots \mid 9A$
$A \rightarrow 0A \mid 1A \mid \ldots \mid 9A \mid \varepsilon \mid .B$
$B \rightarrow 0B \mid 1B \mid \ldots \mid 9B \mid C$
$C \rightarrow 1C \mid 2C \mid \ldots \mid 9C \mid \varepsilon$

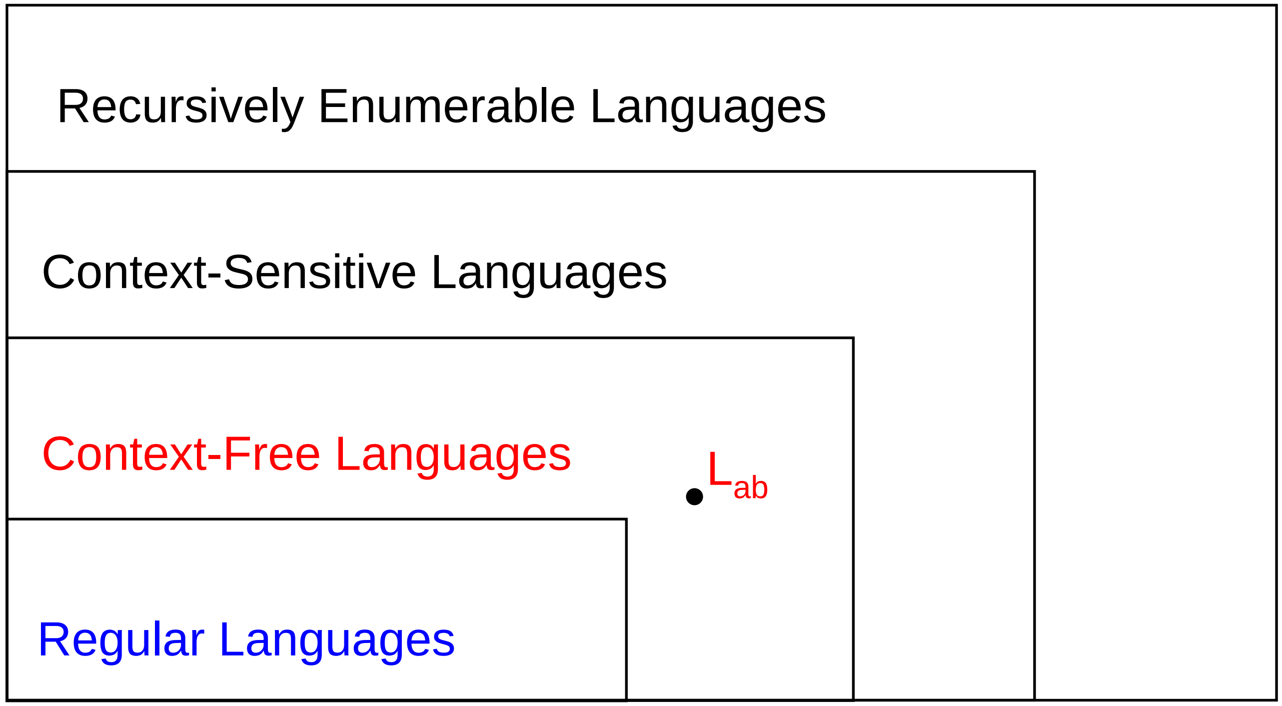
a *context-free grammar*

$S \rightarrow aSb$
$S \rightarrow \varepsilon$

$$L_{ab} = \{ \underbrace{a\ldots a}_{n}\underbrace{b\ldots b}_{n} \mid n \geq 0 \}$$

Formal Language Primer

Chomsky Hierarchy



---

a *regular grammar*

S → 1A | 2A | … | 9A
A → 0A | 1A | … | 9A | ε | .B
B → 0B | 1B | … | 9B | C
C → 1C | 2C | … | 9C | ε

a *context-free grammar*

S → aSb
S → ε

$$L_{ab} = \{ \underbrace{a\ldots a}_{n}\underbrace{b\ldots b}_{n} \mid n \geq 0 \}$$

Proof that L_{ab} not regular ➔ *Lecture 3*!