

Automata and Formal Languages

Peter Wood

Department of Computer Science and Information Systems
Birkbeck, University of London
ptw@dcs.bbk.ac.uk

Outline

Motivation and Background

Motivation and
Background

Automata

Automata

Grammars

Grammars

Regular Expressions

Regular
Expressions

Example of Research

Example of
Research

Conclusion

Conclusion

Doing Research

- ▶ analysing problems/languages
- ▶ computability/solvability/decidability
 - is there an algorithm?
- ▶ computational complexity
 - is it practical?
- ▶ expressive power
 - are there things that cannot be expressed?
- ▶ formal languages provide well-studied models

- ▶ finite alphabet of symbols Σ
 - e.g., $\Sigma = \{0, 1\}$
- ▶ all finite strings over Σ denoted by Σ^*
 - e.g., $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, \dots\}$
- ▶ *language* L over Σ is just subset of Σ^*
 - e.g., L_1 : strings with an even number of 1's
 - e.g., L_2 : strings representing valid Java programs
- ▶ are there finite representations for infinite languages?

- ▶ finite alphabet of symbols Σ
 - e.g., $\Sigma = \{0, 1\}$
- ▶ all finite strings over Σ denoted by Σ^*
 - e.g., $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, \dots\}$
- ▶ *language* L over Σ is just subset of Σ^*
 - e.g., L_1 : strings with an even number of 1's
 - e.g., L_2 : strings representing valid Java programs
- ▶ are there finite representations for infinite languages?
- ▶ yes, *grammars* (generative) and *automata* (recognition)

- ▶ device (machine) for recognising (accepting) a language
- ▶ provide models of computation
- ▶ automaton comprises *states* and *transitions* between states
- ▶ automaton is given a string as input
- ▶ automaton M *accepts* a string w by halting in an accept state, when given w as input
- ▶ language $L(M)$ *accepted* by automaton M is set of strings which M accepts

Types of Automata

- ▶ finite state automaton
 - ▶ deterministic
 - ▶ nondeterministic
- ▶ pushdown automaton
- ▶ linear-bounded automaton
- ▶ Turing machine

- ▶ *grammars* generate languages using:
 - ▶ symbols from alphabet Σ (called *terminals*)
 - ▶ set N of *nonterminals* (one designated as *starting*)
 - ▶ set P of *productions*, each of the form

$$U \rightarrow V$$

where U and V are (loosely) strings over $\Sigma \cup N$

- ▶ a string (sequence of terminals) w is generated by G if there is a *derivation* of w using G , starting from the *starting* nonterminal of G
- ▶ language *generated* by grammar G , denoted $L(G)$, is the set of strings which can be derived using G

Grammar Example

- ▶ L_1 (strings with an even number of 1's) can be generated by grammar with productions

$$S \rightarrow \epsilon$$

$$S \rightarrow 0S$$

$$S \rightarrow 1T$$

$$T \rightarrow 0T$$

$$T \rightarrow 1S$$

where S is the *starting* nonterminal

Grammar Example

- ▶ L_1 (strings with an even number of 1's) can be generated by grammar with productions

$$S \rightarrow \epsilon$$

$$S \rightarrow 0S$$

$$S \rightarrow 1T$$

$$T \rightarrow 0T$$

$$T \rightarrow 1S$$

where S is the *starting* nonterminal

- ▶ a *derivation* of 01010 is given by

$$S \Rightarrow 0\underline{S} \Rightarrow 01\underline{T} \Rightarrow 010\underline{T} \Rightarrow 0101\underline{S} \Rightarrow 01010\underline{S} \Rightarrow 01010$$

Uses of Grammars

- ▶ to specify syntax of programming languages
- ▶ in natural language understanding
- ▶ in pattern recognition
- ▶ to specify schemas (types) for tree-structured data, e.g., XML
- ▶ ...

Hierarchy of Grammars and Languages

- ▶ restrictions on productions give different *types* of grammars
 - ▶ *regular* (type 3)
 - ▶ *context-free* (type 2)
 - ▶ *context-sensitive* (type 1)
 - ▶ *phrase-structure* (type 0)
- ▶ for context-free, e.g., left side must be single nonterminal
- ▶ no restrictions for phrase-structure
- ▶ language is of type i iff there is a grammar of type i which generates it

Examples of Language Hierarchy

- ▶ varying expressive power
- ▶ regular \subset context-free \subset context-sensitive \subset phrase-structure

Examples of Language Hierarchy

- ▶ varying expressive power
- ▶ regular \subset context-free \subset context-sensitive \subset phrase-structure
- ▶ L_1 (strings over $\{0, 1\}$ with an even number of 1's) is regular

Examples of Language Hierarchy

- ▶ varying expressive power
- ▶ regular \subset context-free \subset context-sensitive \subset phrase-structure
- ▶ L_1 (strings over $\{0, 1\}$ with an even number of 1's) is regular
- ▶ $L_2 = \{0^n 1^n \mid n \geq 0\}$ is context-free, but not regular

Examples of Language Hierarchy

- ▶ varying expressive power
- ▶ regular \subset context-free \subset context-sensitive \subset phrase-structure
- ▶ L_1 (strings over $\{0, 1\}$ with an even number of 1's) is regular
- ▶ $L_2 = \{0^n 1^n \mid n \geq 0\}$ is context-free, but not regular
- ▶ $L_3 = \{ww \mid w \in \{0, 1\}^*\}$ is context-sensitive, but not context-free

Examples of Language Hierarchy

- ▶ varying expressive power
- ▶ regular \subset context-free \subset context-sensitive \subset phrase-structure
- ▶ L_1 (strings over $\{0, 1\}$ with an even number of 1's) is regular
- ▶ $L_2 = \{0^n 1^n \mid n \geq 0\}$ is context-free, but not regular
- ▶ $L_3 = \{ww \mid w \in \{0, 1\}^*\}$ is context-sensitive, but not context-free
- ▶ there exists a phrase-structure (recursive) language which is not context-sensitive

Complexity of Grammar Problems

Problem	Type			
	3	2	1	0
Is $w \in L(G)$?	P	P	PSPACE	U
Is $L(G)$ empty?	P	P	U	U
Is $L(G_1) \equiv L(G_2)$?	PSPACE	U	U	U

- ▶ P: decidable in polynomial time
- ▶ PSPACE: decidable in polynomial space (and *complete* for PSPACE: at least as hard as NP-complete)
- ▶ U: undecidable
- ▶ so type of grammar has significant effect on complexity

Relationships between Languages and Automata

A language is

<p>regular context-free context-sensitive phrase-structure</p>	<p>} iff accepted by</p>	<p>{ finite-state pushdown linear-bounded Turing machine</p>
--	----------------------------------	--

Regular Expressions

- ▶ algebraic notation for denoting regular languages
- ▶ use \circ (concatenation), \cup (union) and $*$ (closure) operators
- ▶ L_1 denoted by RE $0^* \cup (0^* \circ 1 \circ 0^* \circ 1 \circ 0^*)^*$
- ▶ given RE R , the set of strings it denotes is $L(R)$
- ▶ pattern matching in text
- ▶ query languages for XML or RDF

Using Regular Expressions to Query Graphs

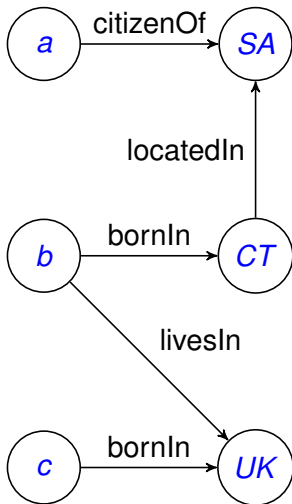
Graphs (networks) are widely used for representing data

- ▶ social networks
- ▶ transportation and other networks
- ▶ geographical information
- ▶ semistructured data
- ▶ (hyper)document structure
- ▶ semantic associations in criminal investigations
- ▶ bibliographic citation analysis
- ▶ pathways in biological processes
- ▶ knowledge representation (e.g. semantic web)
- ▶ program analysis
- ▶ workflow systems
- ▶ data provenance
- ▶ ...

Using Regular Expressions to Query Graphs

- ▶ (my PhD thesis!)
- ▶ usually regular expressions used for string search
- ▶ consider data represented by a directed graph of labelled nodes and labelled edges
- ▶ regular expressions can express *paths* we are interested in
- ▶ sequence of edge labels rather than sequence of symbols (characters)
- ▶ a query using regular expression R can ask for all nodes connected by a path whose concatenation of edge labels is in $L(R)$

Graph G (where nodes represent people and places):



Regular expression

$R = \text{citizenOf} \mid ((\text{bornIn} \mid \text{livesIn}) \cdot \text{locatedIn}^*)$

asks for paths of edges between a person x and a place y such that

- ▶ x is a citizenOf y , or
- ▶ x is bornIn or livesIn y , or
- ▶ x is bornIn or livesIn a place that is locatedIn y

Regular path query evaluation

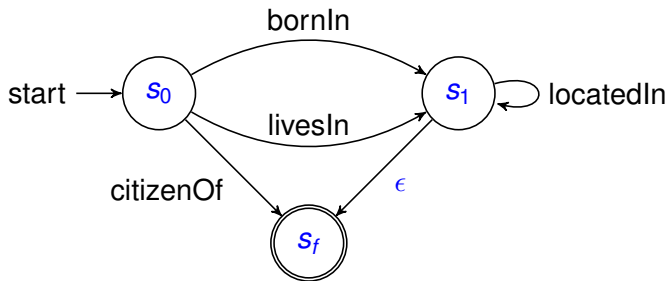
▶ REGULAR PATH PROBLEM

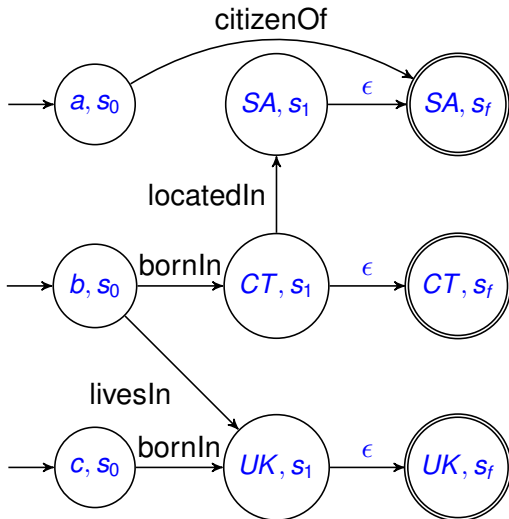
Given graph G , pair of nodes x and y and regular expression R , is there a path from x to y satisfying R ?

▶ algorithm:

- ▶ construct a nondeterministic finite automaton (NFA) M accepting $L(R)$
 - ▶ assume M has initial state s_0 and final state s_f
 - ▶ consider G as an NFA with initial state x and final state y
 - ▶ form the “intersection” I of M and G
 - ▶ check if there is a path from (s_0, x) to (s_f, y)
- ▶ Each step can be done in PTIME, so REGULAR PATH PROBLEM has PTIME complexity

NFA M for $R = \text{citizenOf} \mid ((\text{bornIn} \mid \text{livesIn}) \cdot \text{locatedIn}^*)$



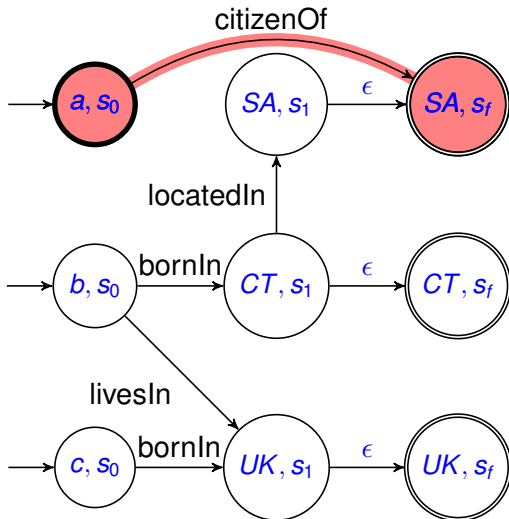
Intersection of G and M Motivation and
Background

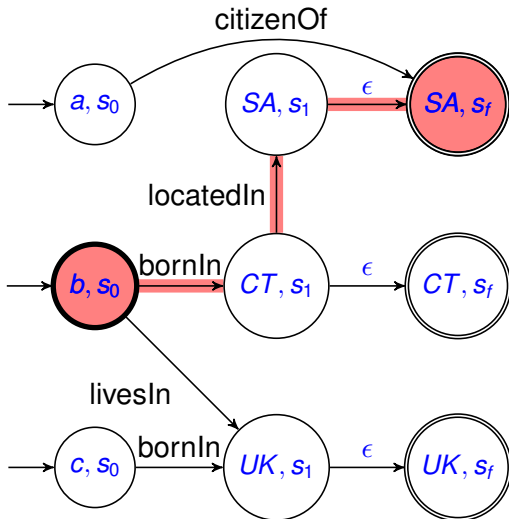
Automata

Grammars

Regular
ExpressionsExample of
Research

Conclusion

Intersection of G and M 

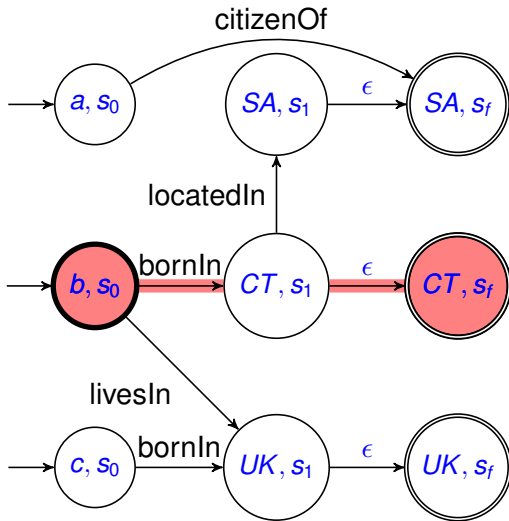
Intersection of G and M Motivation and
Background

Automata

Grammars

Regular
ExpressionsExample of
Research

Conclusion

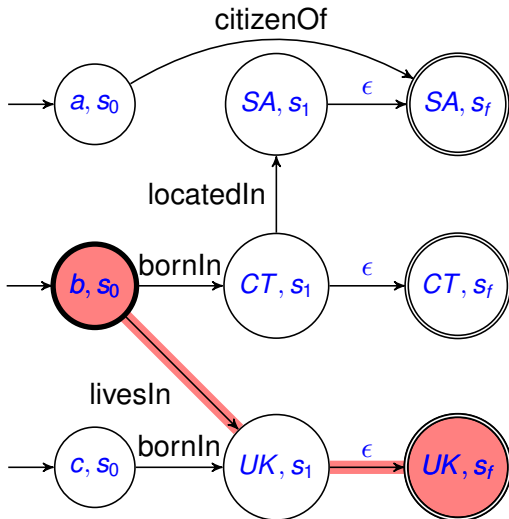
Intersection of G and M Motivation and
Background

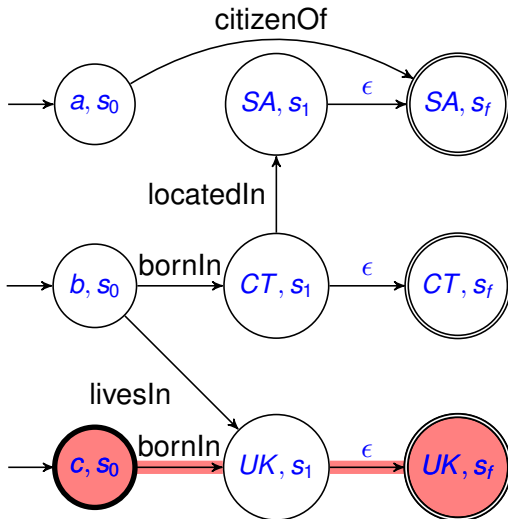
Automata

Grammars

Regular
ExpressionsExample of
Research

Conclusion

Intersection of G and M 

Intersection of G and M Motivation and
Background

Automata

Grammars

Regular
ExpressionsExample of
Research

Conclusion

Regular simple path queries

▶ path p is *simple* if no node is repeated on p

▶ **REGULAR SIMPLE PATH PROBLEM**

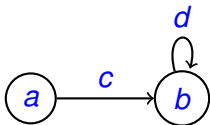
Given graph G , pair of nodes x and y and regular expression R , is there a *simple* path from x to y satisfying R ?

Regular simple path queries

- ▶ path p is *simple* if no node is repeated on p
- ▶ **REGULAR SIMPLE PATH PROBLEM**
Given graph G , pair of nodes x and y and regular expression R , is there a *simple* path from x to y satisfying R ?
- ▶ **REGULAR SIMPLE PATH PROBLEM** is NP-complete [Mendelzon & Wood (1989)]

Regular simple path queries

- ▶ path p is *simple* if no node is repeated on p
- ▶ **REGULAR SIMPLE PATH PROBLEM**
Given graph G , pair of nodes x and y and regular expression R , is there a *simple* path from x to y satisfying R ?
- ▶ **REGULAR SIMPLE PATH PROBLEM** is NP-complete [Mendelzon & Wood (1989)]
- ▶ there can be a path from x to y satisfying R but no simple path satisfying R , e.g., $R = (c \cdot d)^*$



Approaches to deal with this problem

- ▶ what causes the problem?

Approaches to deal with this problem

- ▶ what causes the problem?
- ▶ the presence of cycles

Approaches to deal with this problem

- ▶ what causes the problem?
- ▶ the presence of cycles
- ▶ obvious first step is to consider graphs without cycles—DAGs

Approaches to deal with this problem

- ▶ what causes the problem?
- ▶ the presence of cycles
- ▶ obvious first step is to consider graphs without cycles—DAGs
- ▶ then might look at restricted forms of REs—those corresponding to languages closed under *abbreviations*

Approaches to deal with this problem

- ▶ what causes the problem?
- ▶ the presence of cycles
- ▶ obvious first step is to consider graphs without cycles—DAGs
- ▶ then might look at restricted forms of REs—those corresponding to languages closed under *abbreviations*
- ▶ then one might consider a combination of graphs and REs—those graphs whose cycle structure does not *conflict* with RE

Approaches to deal with this problem

- ▶ what causes the problem?
- ▶ the presence of cycles
- ▶ obvious first step is to consider graphs without cycles—DAGs
- ▶ then might look at restricted forms of REs—those corresponding to languages closed under *abbreviations*
- ▶ then one might consider a combination of graphs and REs—those graphs whose cycle structure does not *conflict* with RE
- ▶ finally show that conflict-freedom is a generalisation:
 - ▶ no RE conflicts with any DAG
 - ▶ RE closed under abbreviations never conflicts with any graph

Other approaches

- ▶ in general, may also run experiments to measure actual running times

Other approaches

- ▶ in general, may also run experiments to measure actual running times
- ▶ may also develop *approximation* algorithms
 - ▶ can sometimes find a PTIME algorithm with a performance guarantee (e.g. for TSP, finds a tour at most twice the optimal distance)
 - ▶ other times this problem itself is NP-hard

Conclusion

- ▶ is my system/language more *powerful* than others?
- ▶ is my system/language more *efficient* than others?
- ▶ expressive power or computational complexity can be studied by relating them to
 - ▶ formal language theory: languages, grammars, automata, ...
- ▶ tradeoff between expressive power and computational complexity
- ▶ consider restrictions of difficult problems or giving up exact solutions

- ▶ Aho, Hopcroft and Ullman, “*The Design and Analysis of Computer Algorithms*,” Addison-Wesley, 1974
- ▶ Garey and Johnson, “*Computers and Intractability: A Guide to the Theory of NP-Completeness*,” W. H. Freeman and Company, 1979
- ▶ Lewis and Papadimitriou, “*Elements of the Theory of Computation*,” Prentice-Hall, 1981
- ▶ Mendelzon and Wood, “Finding Regular Simple Paths in Graph Databases,” *SIAM J. Computing*, Vol. 24. No. 6, 1995, pp. 1235–1258
- ▶ Sipser, “*Introduction to the Theory of Computation*,” PWS Publishing Company, 1997