Logique

Introduction

Florent Madelaine et Mamadou Kanté

M2 Clermont

Logique et Graphes

Année 2013 - 2014

Outline

Motivation

Elements of Complexity Theory

- Computing and measuring the efficiency of a computation
- Turing Machine
- Complexity Classes

Logics

- First Order Logic
- Second Order Logic

Motivation

Elements of Complexity Theory

Logics

Complexity

We classify computational problems according to the amount of various resources needed to solve them by an optimal algorithm. For example, being able to solve a problem in polynomial time (class P), or being able to solve a problem using only polynomial space (class Pspace)..
The evident aim is to model whether one can improve on a known algorithm or not. Complexity theory offers few established lower bounds and relies on conjectures, For example on the famous $P \neq NP$ conjecture.
In practice, we often show that a problem is complete for a complexity class. That is for a suitable notion of reduction (*e.g.* polynomial-time many-one reduction) every computational problem in this complexity class reduces to our problem.

Limits of this approach

First issue

- The computational model is defined for words
- Our computational problems are more often than not on more elaborate structures such as graphs, hypergraphs or relational structures.
- One can of course encode such an elaborate structure by a word (adjacency list, enumeration of the lines of an adjacency matrix, ...)
- But this induces an artificial order on the data.

Example

A database is not ordered in particular in a distributed context.

Limits of this approach

Second issue

- Complexity classes may change if the computational model is amended slightly
  - Additional tapes on a Turing machine.
  - One-dimensional vs Two dimensional Turing machine
  - Turing Machine vs RAM Machine [1]

Example

Linear Time is not the same on a Turing machine and on a RAM machine.

$$Time_{Turing}(n) \subsetneq Time_{RAM}(n) = DLIN$$

[1] *Random Access Machine*

What we would like

- a model that is fit for mote elaborate datastructures than words.
- a model that is machine independent.

The key idea

- We shall not describe the computation (algorithm)
- But the question (problem specification)
- For this we shall use Logic

... or more precisely the logics

- We shall measure the complexity of a computational problem by the richness of the logical language needed to specify it.

This research area is known as Descriptive Complexity

The key idea

- We shall not describe the computation (algorithm)
- But the question (problem specification)
- For this we shall use Logic

  ... or more precisely the logics
- We shall measure the complexity of a computational problem by the richness of the logical language needed to specify it.

This research area is known as Descriptive Complexity

The key idea

- We shall not describe the computation (algorithm)
- But the question (problem specification)
- For this we shall use Logic

  ... or more precisely the logics

- We shall measure the complexity of a computational problem by the richness of the logical language needed to specify it.

This research area is known as Descriptive Complexity

The key idea

- We shall not describe the computation (algorithm)
- But the question (problem specification)
- For this we shall use Logic

... or more precisely the logics

- We shall measure the complexity of a computational problem by the richness of the logical language needed to specify it.

This research area is known as Descriptive Complexity

The key idea

- We shall not describe the computation (algorithm)
- But the question (problem specification)
- For this we shall use Logic

  ... or more precisely the logics
- We shall measure the complexity of a computational problem by the richness of the logical language needed to specify it.

This research area is known as Descriptive Complexity

The key idea

- We shall not describe the computation (algorithm)
- But the question (problem specification)
- For this we shall use Logic

  ... or more precisely the logics
- We shall measure the complexity of a computational problem by the richness of the logical language needed to specify it.

This research area is known as Descriptive Complexity

The key idea

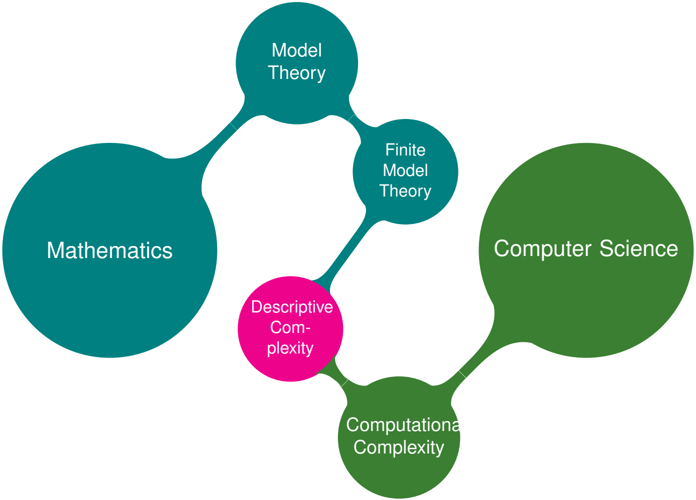
- We shall not describe the computation (algorithm)
- But the question (problem specification)
- For this we shall use Logic

  ... or more precisely the logics
- We shall measure the complexity of a computational problem by the richness of the logical language needed to specify it.

This research area is known as Descriptive Complexity

Descriptive Complexity

Descriptive Complexity establishes a link between finite model theory (the study of logic on finite structures) and Complexity theory.



Descriptive Complexity

We shall discover the fascinating interplay between specification and computation, between the richness of a language necessary to specify a problem and the computational resources necessary to solve this problem.

Motivation

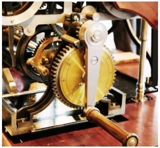
Elements of Complexity Theory

Logics

What is computation?

How does one measure its efficiency?

Input → Machine → Output



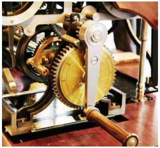
- Machine model: finite automaton, possibly with a stack (pushdown automaton), more generally a Turing machine, a RAM machine, etc
- Typically the input is a word
- What is efficiency?
  Typically [a]
  - parameter: the input size n
  - evaluation: amount of resources such as Time and Space as a function of this parameter.

[a] for enumeration problems where the output may be fairly large compared to the input, one often choses the sum of not input and outputs sizes

What is computation?

How does one measure its efficiency?

Input ⇒ Machine ⇒ Output

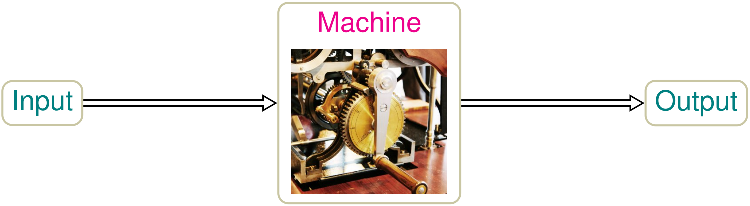


- Machine model: finite automaton, possibly with a stack (pushdown automaton), more generally a Turing machine, a RAM machine, etc
- Typically the input is a word
- What is efficiency?
  Typically [a]
  - parameter: the input size n
  - evaluation: amount of resources such as Time and Space as a function of this parameter.

[a] for enumeration problems where the output may be fairly large compared to the input, one often choses the sum of not input and outputs sizes

What is computation?

How does one measure its efficiency?



- Machine model: finite automaton, possibly with a stack (pushdown automaton), more generally a Turing machine, a RAM machine, etc
- Typically the input is a word
- What is efficiency?

  Typically [a]

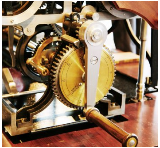
  - parameter: the input size n
  - evaluation: amount of resources such as Time and Space as a function of this parameter.

[a] for enumeration problems where the output may be fairly large compared to the input, one often choses the sum of not input and outputs sizes

What is computation?

How does one measure its efficiency?

Input ⇒ Machine ⇒ Output

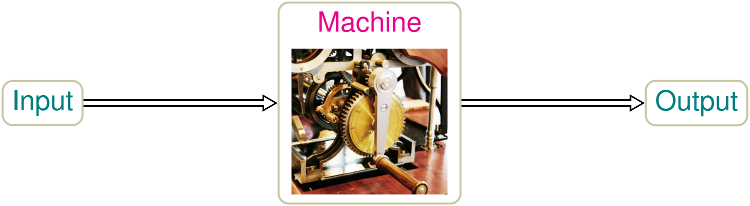


- Machine model: finite automaton, possibly with a stack (pushdown automaton), more generally a Turing machine, a RAM machine, etc
- Typically the input is a word
- What is efficiency?
  Typically [a]
  - parameter: the input size n
  - evaluation: amount of resources such as Time and Space as a function of this parameter.

[a] for enumeration problems where the output may be fairly large compared to the input, one often choses the sum of not input and outputs sizes

What is computation?

How does one measure its efficiency?

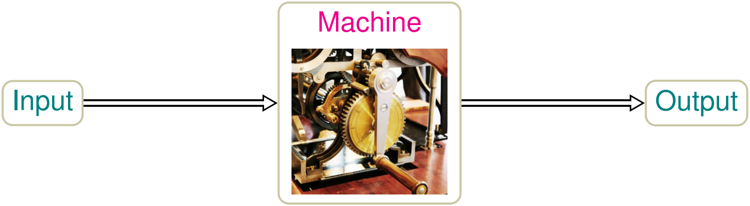


- Machine model: finite automaton, possibly with a stack (pushdown automaton), more generally a Turing machine, a RAM machine, etc
- Typically the input is a word
- What is efficiency?
  Typically [a]
  - parameter: the input size *n*
  - evaluation: amount of resources such as Time and Space as a function of this parameter.

[a] for enumeration problems where the output may be fairly large compared to the input, one often choses the sum of not input and outputs sizes

What is computation?

How does one measure its efficiency?



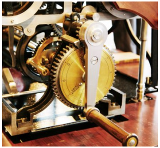
- Machine model: finite automaton, possibly with a stack (pushdown automaton), more generally a Turing machine, a RAM machine, etc
- Typically the input is a word
- What is efficiency?
  Typically [a]
  - parameter: the input size n
  - evaluation: amount of resources such as Time and Space as a function of this parameter.

[a] for enumeration problems where the output may be fairly large compared to the input, one often choses the sum of not input and outputs sizes

Decision problems and Turing Machines

Input a word

Turing Machine



Output yes/no

Associated Problem/Language: set of accepted words.
Vocabulary. We say that the machine decides this language.

Worst Case Complexity

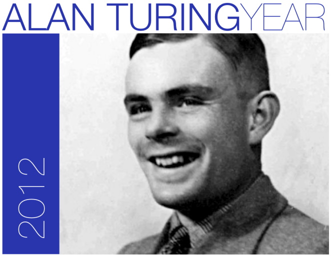
- parameter: the input size n
- evaluation: amount of resources such as Time and Space as a function of this parameter in the worst case.

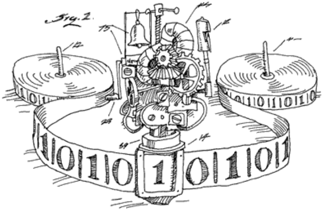
Example

For the binary alphabet $\{0, 1\}$. For all words but the word that consists only of 1s, the machine scans the input word and accepts (time $O(n)$). For the word that consists only of 1s, the machine goes back to the first letter and replace it by 0, then it goes back to the last letter and replaces it by a 0, and so on and so forth until all letter are 0s (time $O(n^2)$).
$\rightarrow$ (worst case) Time Complexity $O(n^2)$.

There are other complexity models such as average case complexity, or Kolmogorov Complexity.

Turing Machine

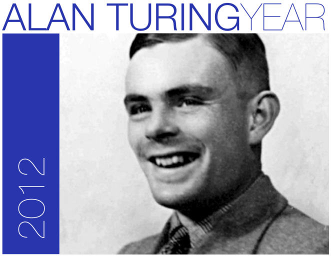


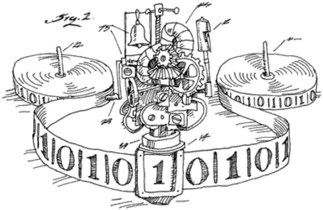


- Time unit: 1 head movement
- Space unit?

  For a finer measure of space complexity we add a working tape and we count 1 space unit per cell of this tape.

Turing Machine

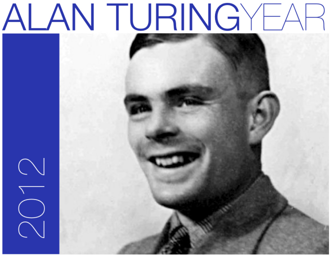


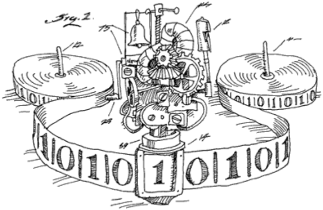


- Time unit: 1 head movement
- Space unit?

  For a finer measure of space complexity we add a working tape and we count 1 space unit per cell of this tape.

Turing Machine

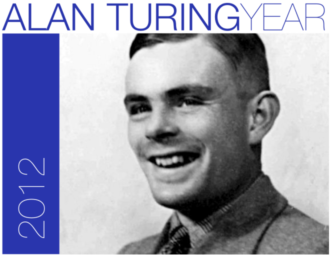


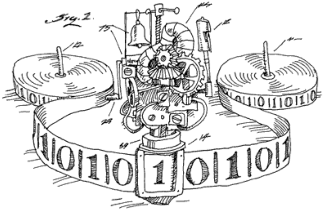


- Time unit: 1 head movement
- Space unit?

  For a finer measure of space complexity we add a working tape and we count 1 space unit per cell of this tape.

Turing Machine





- Time unit: 1 head movement
- Space unit?

  For a finer measure of space complexity we add a working tape and we count 1 space unit per cell of this tape.

Non deterministic Turing Machine

Non-determinism

- The transition function becomes a transition relation
- accepts iff one computation accepts
- Rejects iff all computations reject

For polynomial time, one can pitch the deterministic machine that computes an answer in polynomial-time to the non deterministic one which checks an answer in polynomial time

Non deterministic Turing Machine

Non-determinism

- The transition function becomes a transition relation
- accepts iff one computation accepts
- Rejects iff all computations reject

For polynomial time, one can pitch the deterministic machine that computes an answer in polynomial-time to the non deterministic one which checks an answer in polynomial time

Non deterministic Turing Machine

Non-determinism

- The transition function becomes a transition relation
- accepts iff one computation accepts
- Rejects iff all computations reject

For polynomial time, one can pitch the deterministic machine that computes an answer in polynomial-time to the non deterministic one which checks an answer in polynomial time

Non deterministic Turing Machine

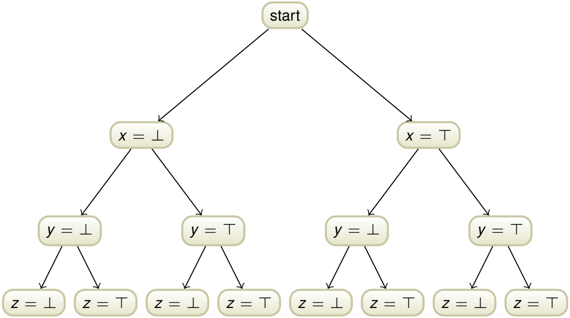
Non-determinism

- The transition function becomes a transition relation
- accepts iff one computation accepts
- Rejects iff all computations reject

For polynomial time, one can pitch the deterministic machine that computes an answer in polynomial-time to the non deterministic one which checks an answer in polynomial time

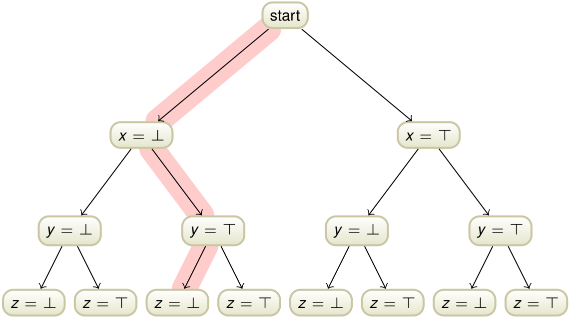
What is reasonable time?

P	vs.	NP
polynomial solving	vs.	polynomial checking



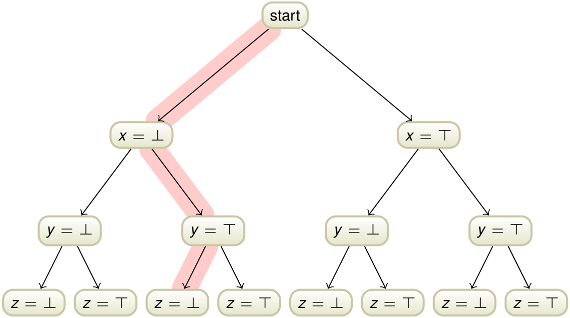
What is reasonable time?

P	vs.	NP
polynomial solving	vs.	polynomial checking



What is reasonable time?

P	vs.	NP
polynomial solving	vs.	polynomial checking



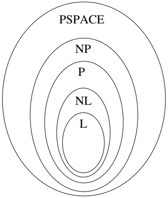
Fo SAT, 2^n possibilities for n variables.
Brute Force Search $\Rightarrow$ exponential time.

Complexity Classes

Class: set of decision problems (languages).
Typically, for which there is an algorithm (a Turing machine) that works with a limited resource (Time or Space).

Well known Classes

Logarithmic Space	L	(Logspace)
(non-det. version)	NL	(NLogspace)
Polynomial Time	NP	(NPtime)
(non-det. version)	P	(Ptime)
Polynomial Space	Pspace	



Reductions

There are several notions of a reduction from a problem Ω_1 to another problem Ω_2.

- The reduction is simply a mean of producing a program for solving Ω_1 given one for Ω_2.
- The most general (Turing reduction) allows several calls to the program for Ω_2.
- The least general (Karp reduction) allows only one and at the end of the reduction process
- We restrict the resources of the reduction according to the computational class under study[2]

[2] We attempt to work with reductions under which the complexity class is invariant.

Reductions

For the complexity class we will be working with (P and NP), we shall use polynomial time Karp reductions.

For this classes (P,NP) and classes of lower coplexity (L,NL), one uses weaker reductions in the literature, the so-called logspace reductions.

Complete Problemes

A problem is complete for a class if all other problems reduce to it[3].

Classe			Problèmes complets
L		2-Col	Graph accessibility in undirected graph
NL	2-Sat		Graph accessibility in directed graphs
P	Horn-Sat		Accessibility in 3-Hypergraphs
NP	Sat	3-Col	
Pspace	QSat		Accessibility game (Hex)

Consequence: Sat being in P would imply P=NP.

[3] To be more precise, one should indicate the notion of a reduction under use

Structure

- We fix names of symbols. For example for graphs, we use a binary symbol E to denote the edge relation.
- We could also use K to denote a vertex subset.
- One can also have relation symbols of higher arity (tables in databases, hypergraphs), and even several relations (graphs with several types of edges, say "coloured edges")
- We call signature of the structure the set of these symbol names and their arity.
- We may also use constant symbols (= 0-ary relation).

Example

To model graph accessibility, we need a graph and two special vertices (2 constants) s and t. signature: E of arity 2, s of arity 0 and t of arity 0.

We may also consider signatures with function symbols and structures with functions. We shall restrict ourselves to relational structures in this lecture.

Structure

We fix names of symbols. For example for graphs, we use a binary symbol E to denote the edge relation.

We could also use K to denote a vertex subset.

One can also have relation symbols of higher arity (tables in databases, hypergraphs), and even several relations (graphs with several types of edges, say “coloured edges”)

We call signature of the structure the set of these symbol names and their arity.

We may also use constant symbols (= 0-ary relation).

Example

To model graph accessibility, we need a graph and two special vertices (2 constants) s and t. signature: E of arity 2, s of arity 0 and t of arity 0.

We may also consider signatures with function symbols and structures with functions. We shall restrict ourselves to relational structures in this lecture.

structure

We have just defined the notion of a signature. A structure (over such a signature) is given by a set (the universe or domain) and an interpretation of the correct arity for each symbol of the signature.

Example

For the signature E of arity 2, z of arity 0 and t of arity 0, we consider a structure S that consists of

- a domain V (); and,
- an interpretation for E which is a relation $E^S \subseteq V \times V$
- an interpretation for s which is an element s^S of V
- an interpretation for t which is an element t^S of V

E^S denotes the concrete relation and E the relation symbol. When it is clear from context, we shall use the latter to denote both.

structure

We have just defined the notion of a signature. A structure (over such a signature) is given by a set (the universe or domain) and an interpretation of the correct arity for each symbol of the signature.

Example

For the signature E of arity 2, z of arity 0 and t of arity 0, we consider a structure S that consists of

- a domain V (for us almost always finite); and,
- an interpretation for E which is a relation $E^S \subseteq V \times V$
- an interpretation for s which is an element s^S of V
- an interpretation for t which is an element t^S of V

E^S denotes the concrete relation and E the relation symbol. When it is clear from context, we shall use the latter to denote both.

First Order Logic

We consider an infinite set of variables $x, y, z, \ldots$ (intuition: the value of a variable will be a graph vertex).

- The terms are built from the relation symbols, variables and constant symbols (example: $E(x, y)$ to denote an edge from x to y).
- We use the connectors

  $\wedge$ and
  $\vee$ or
  $\neg$ negation
- and the quantifiers

  $\forall$ universal
  $\exists$ existential

First Order Logic

Definition (Formulae)

- atomic formulae
  - (relational) term that is an r-ary symbol from the signature, and r variables or constants; or,
  - an identity $x = y$.
- inductive definition
  - If φ_1 are φ_2 are formulae then $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$ and $\neg\varphi_1$ are also formulae
  - If φ is a formula then $\exists x \varphi$ and $\forall x \varphi$ are formulae.

Vocabulary

- A formula is quantifier-free if it does not contain the quantifier symbols $\exists$ or $\forall$.
- A variable x is bound if it appears in the scope of a quantifier, that is it appears in a subformula which is preceded by $\forall x$ or $\exists x$.
- A formula that is not bound is said to be free.
- We write $\varphi(\bar{x})$ to denote that the variables $\bar{x}$ are free.

Example

In the settings of directed graphs that may have loops.
Signature: E binary.

Property: every vertex has a successor distinct from itself

$$\forall x \exists y E(x, y) \wedge \neg(x = y)$$

We shall also use some abbreviations for the sake of readability such as $x \neq y$ instead of $\neg(x = y)$.

$$\forall x \exists y E(x, y) \wedge x \neq y$$

Semantic

A graph G (encoded as a structure over the signature E binary) satisfies the formula

$$\varphi := \forall x \exists y E(x, y) \wedge x \neq y$$

iff
every vertex of G has a successor distinct from itself.

Notation and vocabulary

$G \models \varphi$
G is a model of φ
G satisfies φ

Semantic

We defined the meaning of $S \models \varphi(\bar{a})$ inductively.
(here $\bar{a}$ represents a tuple of values from the of S, one value for each variable of $\bar{x}$).

- We interpret a (constant symbol) c by c^S and a free variable $x_i(\bar{a})$ by a_i.
- An identity holds iff the two terms either side of the = symbol have the same value in S
- A relational term $R(t_1, t_2, \dots, t_r)$ holds in S if the tuple $R(t_1^S(\bar{a}), t_2^S(\bar{a}), \dots, t_2^S(\bar{a}))$ belongs to the relation R^S of S.
- $S \models \neg\varphi_1(\bar{a})$ if it is not the case that $S \models \varphi_1(\bar{a})$
- $S \models \varphi_1(\bar{a}) \wedge \varphi_2(\bar{a})$ iff we have both $S \models \varphi_1(\bar{a})$ and $S \models \varphi_2(\bar{a})$
- $S \models \varphi_1(\bar{a}) \vee \varphi_2(\bar{a})$ iff we have $S \models \varphi_1(\bar{a})$ or $S \models \varphi_2(\bar{a})$
- $S \models \exists y \varphi(y, \bar{a})$ iff there exists a domain value a' for y such that $S \models \varphi(a', \bar{a})$
- $S \models \forall y \varphi(y, \bar{a})$ iff for all value of the domain a' for y, we have $S \models \varphi(a', \bar{a})$

Examples

Property: a vertex has a predecessor, or a successor

$$\forall x(\exists y E(x,y) \wedge x \neq y) \vee (\exists z E(z,x) \wedge x \neq z)$$

Remark: we may reuse variable names.

$$\forall x(\exists y E(x,y) \wedge x \neq y) \vee (\exists y E(y,x) \wedge x \neq y)$$

Examples

Property: The binary relation E is symmetric and non reflexive (in short it represents the edge relation of a loopless undirected graph).

$$\forall x \neg E(x,x) \wedge \forall y E(x,y) \implies E(y,x)$$

Notation

We us the abbreviation $\varphi \implies \psi$ for $\neg\varphi \vee \psi$.
We use also $\varphi \iff \psi$ for $(\varphi \implies \psi) \wedge (\psi \implies \varphi)$

Examples

Property: diameter at most 2 (without taking orientation into account).

$$\begin{aligned}\forall x \forall y\, x = y \; \vee (E(x,y) \vee E(y,x)) \\ \vee \exists z (E(x,z) \vee E(z,x)) \wedge (E(z,y) \vee E(y,z))\end{aligned}$$

As to keep formulae readable, when working over undirected graphs, we'll assume E to be symmetric and write:

$$\forall x \forall y\, x = y \; \vee E(x,y) \vee \exists z E(x,z) \wedge E(z,y)$$

Example: other signature

Structure: Graph (edge E) + a vertex set K
Property: K is a Kernel of G (independent set that covers all vertex).

Exercise

FO formula?

Representing words

- Alphabet $\Sigma = \{a, b\}$.

  Word: $s_1 s_2 \dots s_n$
- Encode by a structure with domain $\{1, 2, \dots, n\}$ that has a binary relation $<$ (such that $i < i + 1$) and two unary predicates A and B. We have $A(i)$ iff $s_i = a$ and $B(i)$ iff $s_i = b$.
- FO formula for the following language ab^*a?

  There is at least 2 letters and

  There is a a at the beginning and a a at the end; and,

  There is no a among the other letters.

Representing words

- Alphabet $\Sigma = \{a, b\}$.

  Word: $s_1 s_2 \ldots s_n$
- Encode by a structure with domain $\{1, 2, \ldots, n\}$ that has a binary relation $<$ (such that $i < i + 1$) and two unary predicates A and B. We have $A(i)$ iff $s_i = a$ and $B(i)$ iff $s_i = b$.
- FO formula for the following language ab^*a?

  There is at least 2 letters and

  There is a a at the beginning and a a at the end; and,

  There is no a among the other letters.

Weakness of FO

We can prove that we can not express simple property such as:

- The universe has even size
- Accessibility

In fact, in general FO may only express local properties (in a precise sense).

Second Order Logic

Idea

We allow to quantify over new relations symbols.

Example

We may “guess” a binary relation L and state that it is a linear order.

$$\varphi_L := \exists L \forall x \neg (L(x,x) \wedge \forall x,y,z\big((x,y) \wedge L(y,z) \implies L(x,z)\big) \\ \wedge \forall x,y L(x,y) \vee L(y,x))$$

Successor

We may deduce from a linear order, the underlying successor relation.

$$\varphi_S := \varphi_L \wedge \exists S$$
$$\forall x, y S(x, y) \iff \big(L(x, y) \wedge \neg L(x, z) \wedge L(z, y)\big)$$

Once we have an order, we may easily “walk”along the path defined by S and check that we indeed have an even numbe of elements.

Example: 3 colorability

We want to describe the following property.
It is possible to colour each vertex of a graph in red, green or blue, such that adjacent vertices have different colours.

3 colorability

$$\exists R \exists B \exists V \forall x \big(R(x) \vee B(x) \vee V(x) \big) \wedge$$

$$\forall x \Big(\neg \big(R(x) \wedge B(x) \big) \wedge \neg \big(R(x) \wedge V(x) \big) \wedge \neg \big(B(x) \wedge V(x) \big) \Big) \wedge$$

$$\forall x \forall y \neg \big(E(x,y) \wedge R(x) \wedge R(y) \big) \wedge$$

$$\neg \big(E(x,y) \wedge B(x) \wedge B(y) \big) \wedge \neg \big(E(x,y) \wedge V(x) \wedge V(y) \big)$$

- In the example of 3-colorability, the second order predicates are all sets
- In this restricted case, we speak of Monadic Second Order logic (MSO)
- Note also that unlike in this example, in general in MSO we may quantify universally over a set.
- MSO plays a central role in language theory.

Syntax and Semantic

Syntax

Similar to FO, but we allow terms using the new symbols[4] and we allow their quantification, whether existential $\exists R$ or universal $\forall R$.

Semantic

Similarly to FO, we interpret a new symbol R of arity r as a new relation R^S or arity r over the structure.

Notation

Traditionally we reserve uppercase letters to predicates (SO variables or signature symbols) and use lower case letters for FO variables or constant symbols.

[4] We allow for infinitely many symbols of each arity.

Various fragments of ESO

SO	Second Order Logic	extension of First Order Logic with new symbols representing relations
MSO	Monadic Second Order Logic	restriction of Second Order Logic where the new relation symbols are necessarily sets (arity 1)
ESO	Existential Second Order Logic	of the form $\exists$ second order symbols followed by a first-order formula
FO	First Order logic	No new symbols, only those from the signature

Teaser

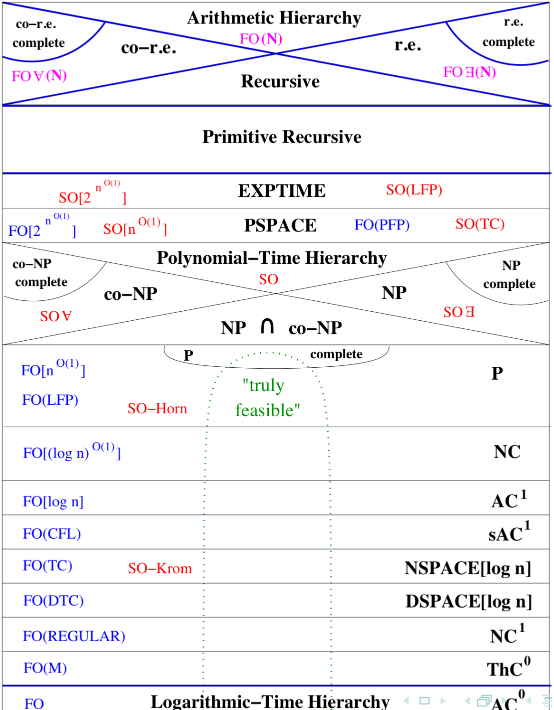
Theorem (Büchi)

For words: Regular Languages = MSO

Theorem (Fagin)

For structures: NP = ESO

There are many other logics of interest in the context of Complexity Theory


Arithmetic Hierarchy
co–r.e. complete
co–r.e.
FO (N)
r.e.
r.e. complete
FO ∀(N)
Recursive
FO ∃(N)
Primitive Recursive
SO[2^{n^{O(1)}}]
EXPTIME
SO(LFP)
FO[2^{n^{O(1)}}]
SO[n^{O(1)}]
PSPACE
FO(PFP)
SO(TC)
Polynomial–Time Hierarchy
co–NP complete
SO
NP complete
co–NP
NP
SO ∀
SO ∃
NP ∩ co–NP
P complete
FO[n^{O(1)}]
"truly feasible"
P
FO(LFP)
SO–Horn
FO[(log n)^{O(1)}]
NC
FO[log n]
AC^1
FO(CFL)
sAC^1
FO(TC)
SO–Krom
NSPACE[log n]
FO(DTC)
DSPACE[log n]
FO(REGULAR)
NC^1
FO(M)
ThC^0
FO
Logarithmic–Time Hierarchy
AC^0

The graal: a logic for P

One of the famous open question from Descriptive Complexity asks for a logic suitable for P.

Motivation: to have a query language in database theory that is both efficient and as expressive as possible.

Next Lectures

5 Last lecture on graphs (Anne Berry)

6 Logic et Automaton (Mamadou Kanté)

7 Fagin's theorem (me)

8,9 MSO and tree decomposition
(Courcelle's theorem) (Mamadou Kanté)

10 Constraint Satisfaction problems and decomposition (me)