Plan of the talk

- What is descriptive complexity. Main result of the chapter (Fagin's theorem).
- First order logic
- Second order existential logic ($SO\exists$)
- Spectral problem
- NP
- $SO\exists \subseteq NP$ (all queries expressed in $SO\exists$ can be evaluated on an NP machine).
- $NP \subseteq SO\exists$ (all NP problems can be expressed in $SO\exists$).

What is descriptive complexity

Descriptive complexity is about capturing complexity classes in logic.

Instead of using computer resources to characterise the difficulty of a problem, use expressive power of a logic.

Fagin's theorem: a problem is in NP if, and only if, it is expressible in existential second order logic (Fagin, 1974).

Relational structures

A relational structure $\mathbf{A}$ consists of

- a set called the domain of $\mathbf{A}$, A.
- a set of relations on A: an r-ary relation on $\mathbf{A}$ is a subset of A^r ($r > 0$). Relations are *named* by relation symbols.

Signature of a structure is a set of symbols for relations and constants of the structure.

Note that anything can be represented as a relational structure. A binary string $s = s_1 \dots s_n$, for example, corresponds to a relational structure $\langle A, <, BIT \rangle$ where $|A| = n$, $<$ is a total order and $BIT(a_i)$ holds if the ith bit of s is 1.

Language of first order logic

Logical symbols

- variables $x, y, x_1, \dots$
- equality $=$
- boolean connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$
- quantifiers $\forall$, $\exists$

plus some signature τ (non-logical symbols):

- relation symbols.

Atomic formulas

- if x and y are variables, then $x = y$ is an atomic formula
- if $x_1, \ldots, x_n$ are variables and R is an n-ary relation symbol, then $R(x_1, \ldots, x_n)$ is an atomic formula.

Formulas

- An atomic formula is a formula;
- If φ, ψ are formulas, then so are $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \to \psi$.
- If φ is a formula, and x a variable, then $\forall x\varphi$ and $\exists x\varphi$ are formulas. Occurrences of x in the scope of the quantifier are bound by this quantifier (unless they are already bound by another quantifier). Variables which are not bound are called free.

A formula with no free variables is called a sentence.

Meaning of formulas

φ is satisfied by assignment s on $\mathbf{A}$ (s is a finction from variables into A), $\mathbf{A}, s \models \varphi$:

- $\mathbf{A}, s \models x = y$ iff $s(x) = s(y)$
- $\mathbf{A}, s \models R(x_1, \ldots, x_r)$ iff $\langle s(x_1), \ldots, s(x_r)\rangle \in R^A$
- $\mathbf{A}, s \models \neg\varphi$ iff $A, s \not\models \varphi$
- $\mathbf{A}, s \models \varphi \wedge \psi$ iff $\mathbf{A}, s \models \varphi$ and $A, s \models \psi$
- $\mathbf{A}, s \models \exists x\varphi$ iff for some assignment s' which differs from s at most in the value assigned to x, $\mathbf{A}, s' \models \varphi$

Evaluating second order formulas

Second order formulas are evaluated on relational structures in the same way as first order formulas.

$\mathbf{A}, s \models \exists X^r\varphi$ iff there is an interpretation of X such that under this interpretation $\mathbf{A}, s \models \varphi$. (There is a set R of r-tuples from A such that if X is interpreted as R, $\mathbf{A}, s \models \varphi$ in the usual first-order sense.)

Existential second order logic

Only existential quantifiers over relational variables are allowed. All formulas in $SO\exists$ can be written as

$$\exists X_1 \ldots \exists X_n \phi$$

where $X_1, \ldots, X_n$ are relational variables and ϕ a formula which does not contain second order quantifiers (quantifiers over relational variables).

Spectral problem

Given a first order formula ϕ,

$Spectrum(\phi) = \{n : \exists \mathbf{A}(\mathbf{A} \models \phi \text{ and } |A| = n)\}$

Spectral problem: characterise such sets; are they closed under complement?

Given a $SO\exists$ formula ψ,

$Models(\psi) = \{\mathbf{A} : \mathbf{A} \models \phi \text{ and } A \text{ is finite}\}$

is called *generalised spectrum* of ψ.

How hard is it to answer, whether for a $SO\exists$ formula ψ, a particular relational structure is in $Models(\psi)$?

Turing machine

Abstraction of a computing device; used to define complexity classes.

A Turing machine has:

1. a tape (unbounded to the right)
2. finite work alphabet Σ
3. a head which can read a symbol from Σ in the cell it is facing, write a symbol from Σ in the cell it is facing, and move one cell to the left or to the right or stay in the same cell
4. a finite set of states, which includes initial state s_0, accepting state s_+, rejecting state s_-.
5. a finite set of instructions of the form: 'if you are in state s reading a symbol a then replace a with b, move left (or right, or don't move) and go into state s''.

Deterministic vs non-deterministic Turing machines

- Deterministic TM: at most one instruction for any state and symbol pair.
- Non-deterministic TM: more than one instruction may be applicable in some situations.

A deterministic TM accepts an input string if its computation started with the input on its tape halts in an accepting state. A non-deterministic TM accepts an input string if one of its computations starting with this input halts in an accepting state.

Time and space bounds

Obviously we can write a set of instructions for a TM so that it never stops and keeps using more and more cells on the tape.

In general, it is undecidable whether a TM stops (halting problem) and after how many steps it halts on a given input.

However, for some problems, we *can* say how many steps the TM will need to make and how many cells on the work tape it will need.

Those problems can be guaranteed to be solved by a TM with a bounded tape and a clock which stops it after so many steps (both bounds depend on the size of the input).

Problems

What do we mean by 'problems computable by such and such machine'?

Since inputs to Turing machines are strings, problems are languages (classes of strings).

However, we can represent binary strings by relational structures.

We can also encode relational structures as strings (provided we assume some order on elements).

So, in what follows, we will identify problems with properties of relational structures.

Space and time complexity classes

$DTIME(t(n))$ the set of problems computable by a deterministic TM in $O(t(n))$ steps for inputs of size n. (The number of steps is bounded from above by a function of the form $t(n)$, for all n which are larger than some n_0.)

$NTIME(t(n))$ the set of problems computable by a non-deterministic TM in $O(t(n))$ steps for inputs of size n.

$DSPACE(s(n))$ the set of problems computable by a deterministic TM using $O(s(n))$ cells for inputs of size n.

$NSPACE(s(n))$ the set of problems computable by a non-deterministic TM using $O(s(n))$ cells for inputs of size n.

Some complexity classes

$L = DSPACE(log(n))$ logarithmic space (logspace)

$NL = NSPACE(log(n))$ nondeterministic logspace

$P = \bigcup_{k=1}^{\infty} DTIME(n^k)$ polynomial time

$NP = \bigcup_{k=1}^{\infty} NTIME(n^k)$ nondeterministic polynomial time

$PSPACE = \bigcup_{k=1}^{\infty} DSPACE(n^k) =$
$= \bigcup_{k=1}^{\infty} NSPACE(n^k)$

$EXPTIME = \bigcup_{k=1}^{\infty} DTIME(2^{n^k})$

$L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE \subseteq EXPTIME$

$NL \neq PSPACE$

$SO\exists \subseteq NP$

A very informal argument that for every $SO\exists$ formula ψ there is a non-deterministic TM which given a structure of size n answers whether it satisfies ψ in $O(n^k)$ steps:

1. Fix $\psi = \exists X_1^{r_1} \dots \exists X_k^{r_k} \phi$. Recall that ϕ is like a first order formula only it may contain atomic subformulas of the form $X_i^{r_i}(x_1, \dots, x_{r_i}))$ as well as atomic subformulas of the form $R^m(x_1, \dots, x_m)$.

2. Given a structure $\mathbf{A}$ of size n, and some fixed interpretations of $X_1^{r_1}, \dots, X_k^{r_k}$, checking whether $\mathbf{A} \models \phi$ takes polynomial time in the size of n. For example, to evaluate $\exists x_1 \exists x_2 \exists x_3 R(x_1, x_2, x_3)$, we need to check all triples of elements from A: the polynomial is n^3.

3. An NP machine can 'guess' an interpretation of relational variables and then evaluate ϕ in polynomial time.

$NP \subseteq SO\exists$

We have shown that if a problem (a property of relational structures) can be expressed in $SO\exists$, then it can be solved (evaluated) in NP.

To prove the other direction (if some problem is in NP, it can be expressed in $SO\exists$), we reason as follows.

1. Suppose a problem X is in NP

2. Then there is an NP machine M which accepts only structures in X

3. Let us describe M by an $SO\exists$ formula ϕ_M so that a structure $\mathbf{A} \in X$ ($\mathbf{A}$ is accepted by M) iff $\mathbf{A} \models \phi_M$.

4. This means that we have written a $SO\exists$ formula which is true only on structures in X, that is the problem X is expressible in $SO\exists$.

Encoding M by a formula 1

It is crucial that M is an NP machine, that is, for an input of size n M makes at most n^k steps and uses at most n^k cells on the tape.

Assumptions made in the proof:

- M never visits the cells to the left of the input;
- input in in alphabet $\{0, 1, \_\} \subseteq \Sigma$ (Σ is the work alphabet of M)
- Z is the set of states of M. The halting state s_+ is reached always in the leftmost position and reading a blank symbol (this is just for convenience; any machine can be made to do this: instead of accepting, first go to the left, erase the symbol there and then accept).
- after the machine accepts, it cycles in accepting state;

Encoding M by a formula 2

Configuration of M at every step in the computation is given by:

- what is the state of M;
- which cell is the head scanning;
- which symbol is in every of the n^k cells on the tape used in the computation.

Configurations of M can be described by strings of length n^k over the alphabet $\Gamma = \Sigma \cup (Z \times \Sigma)$:

$$a_1\, a_2\, \dots\, a_{i-1}\, (z, a_i)\, a_{i+1}\, \dots\, a_{n^k}$$

says that the tape contents

$$a_1\, a_2\, \dots\, a_{i-1}\, a_i\, a_{i+1}\, \dots\, a_{n^k}$$

the head is in cell i and the state is z.

Encoding M by a formula 3

M's accepting computation on an input which is an encoding of structure $\mathbf{A}$ with domain of size n (encoding is longer, since an r-ary relation will be encoded as a string of length n^r. We assume that the sum of arities of all relations is less than k) is represented by a $n^k \times n^k$ matrix:

$(s_0, a_1)\ \ a_2 \ldots a_m$ _

$\ldots \qquad\qquad\qquad\qquad \ldots$

$\ldots \qquad\qquad\qquad\qquad \ldots$

$\ldots \qquad\qquad\qquad\qquad \ldots$

$(s_+, \_)\ \ b_2 \ldots \qquad\qquad \ldots\ b_{n^k}$

Encoding M by a formula 4

To encode this computation, we should be able to encode every string (which Γ symbol is at each position) and the transition relation (how next configuration is computed).

Then the formula ϕ_M will say that the initial configuration is such and such, each subsequent configuration is in the transition relation, and the n^kth configuration is an accepting one.

Encoding M by a formula 5

To encode a configuration, we should be able to encode n^k different positions. We cannot use n^k variables because then our formula will depend on the size of the structure. Instead we use k-ary tuples to encode different positions. If a structure has n elements, there are n^k different tuples.

If we have a total order relation $<$ we can define a successor relation S_1:

$$S_1(x, y) = (x < y) \wedge \forall z((x < z) \wedge (y \neq z) \rightarrow (y < z))$$

Similarly, we can define a successor relation S_k between k-tuples in the lexicographic ordering of tuples.

$x_1 ... x_n < y_1 ... y_n$ if
$x_1 ... x_i = y_1 ... y_i$ and $x_{i+1} < y_{i+1}$
for some i where $0 \leq i \leq n - 1$.

Encoding M by a formula 6

For every symbol $a \in \Gamma$ we introduce a new $2k$-ary relational symbol P_a.

$P_a(x_1, ..., x_k, y_1, ..., y_k)$ is intended to be true if and only if symbol a is in position $(x_1, ..., x_k, y_1, ..., y_k)$ in the matrix ($\bar{x} = x_1, \ldots, x_k$ encodes the cell number, $\bar{y} = y_1, \ldots, y_k$ the configuration or time step number).

Now we can express instructions. For example, if there is an instruction which says that 'in state s reading 0, replace 0 by 1, stay in the same cell and change the state to z', we can write

$$\forall \bar{x}, \bar{y}, \bar{u}(P_{(s,0)}(\bar{x}, \bar{y}) \wedge S(\bar{y}, \bar{u}) \rightarrow P_{(s',1)}(\bar{x}, \bar{u}))$$

Encoding M by a formula 7

In general, symbol in the position $(i+1, j+1)$ in the matrix depends on what is at positions (i,j), $(i+1,j)$ and $(i+2,j)$ and on non-deterministic choice of instruction:

$$\begin{array}{ccc} (i,j) & (i+1,j) & (i+2,j) \\ & (i+1,j+1) & \end{array}$$

Assuming that the machine can make only two choices each time (harmless assumption), this can be encoded as two transition relations Δ_0 and Δ_1 on symbols from Γ. Δ_0 list one set of choices

$$\begin{array}{ccc} a & b & c \\ & d & \end{array}$$

and Δ_1 the other.

We use a k-ary predicate $C(y_1, ..., y_k)$ to encode the fact that at time step $y_1, ..., y_k$ an instruction from Δ_1.

Encoding M by a formula 8

The general form of transition relation for configurations (provided we are inside the matrix) is

$$\forall \bar{x}, \bar{y}, \bar{x'}, \bar{x''}, \bar{y'} \bigwedge_{(a,b,c,d) \in \Delta_1}$$

$$(P_a(\bar{x}, \bar{y}) \wedge P_b(\bar{x'}, \bar{y}) \wedge P_c(\bar{x''}, \bar{y}) \wedge$$

$$\wedge S(\bar{x}, \bar{x'}) \wedge S(\bar{x'}, \bar{x''}) \wedge S(\bar{y}, \bar{y'}) \wedge$$

$$\wedge C(\bar{y}) \rightarrow R_d(\bar{x'}, \bar{y'})))$$

Encoding M by a formula 9

We also need to be able to encode the first time moment/cell position (least element in the ordering of k-tuples) and last time moment/cell position (the greatest element, or n^k).

To do this, it is enough to define a unary relation Min which holds of the least element in $<$ and Max which holds of the greatest. Then the least tuple in the ordering of K-tuples is going to have all k elements in Min, same for the greatest tuple and Max.

Encoding M by a formula 10

Finally, ϕ_M is going to be of the form

$$\exists < \exists\ Min\ \exists\ Max\ \exists\ S_k \exists\ P_{a_1} \ldots \exists\ P_{a_g} \exists\ C\ \ \psi$$

where $a_1, \ldots a_g$ are all the symbols in Γ and ψ contains

- definitions of $<$, Min, Max, and S_k;
- encodings of instructions;
- description of the start configuration
- description of the n^kth configuration which has state s_+:
  $\exists \bar{x} \exists \bar{y} (\bigwedge_{x_i} Min(x_i) \wedge \bigwedge_{y_i} Max(y_i) \wedge P_{(s_+,-)}(\bar{x}, \bar{y}))$