UCT

Descriptive Complexity

Klaus Sutner

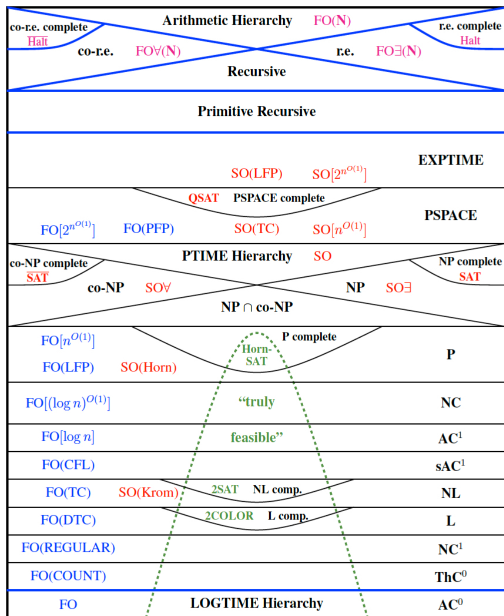
Carnegie Mellon University

Spring 2024

1 **Descriptive Complexity**

2 **Words as Structures**

3 **Existential SOL**



abstract computation

civilized computation

feasible computation

We have covered all the major classes in Immerman's little chart, from the arithmetical hierarchy at the top all the way down to $\mathbb{L}$ and circuits.

What's missing:

The descriptive classes in blue on the bottom left, and red further up.

The blue labels refer to first-order logic, and the red ones to second-order logic.

There are lots and lots of plausible models of computation:

> λ-calculus, Gödel-Herbrand equations, primitive recursive functions, Turing machines (one-tape, k-tape, separate input/output tape, oblivious, deterministic, nondeterministic, probabilistic, alternating), finite state machines, pushdown automata, linear bounded automata, counter machines, random access machines, Kolmogorov-Uspenskii machines, pointer machines, circuits, straight line programs, branching programs, ...

At the very least, this is all a bit confusing. More importantly, is there any real substance to the results obtained by invoking all these models? Or are they all more or less accidental?

One way to separate oneself from vexing definitional details of machine models is to recast everything in terms of **logic**.

Big Idea:

Measure the complexity of a problem by the complexity of the logic that is necessary to express it. In other words, write down a careful description of your problem in a as weak a formal system as you can manage, and declare the complexity of the problem to be the complexity of that system.

This is in stark contrast to the standard approach where everything is coded up in Dedekind-Peano arithmetic or Zermelo-Fraenkel set theory (typically using first-order logic): these are both sledge hammers, very convenient and powerful, but not subtle. Appropriate for recursion theory, real analysis, differential equations and the like, but not so much for complexity theory.

A logic or logical system has the following parts:

- a formal language (syntax)
- a class of structures (semantics)
- a formal notion of proof
- effectiveness requirements

The effectiveness requirements depend a bit on the system in question, minimally we would want that it is decidable whether a string is a formula. Also, it should be decidable whether an object is a valid proof (this says nothing about proof search).

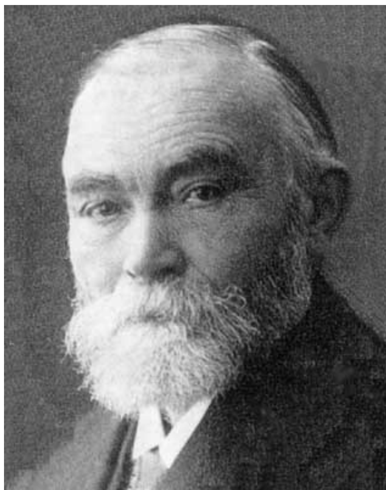
At any rate, we are here not at all interested in proof theory.

Typical Examples

- propositional logic
- equational logic
- first-order logic
- second-order logic

These are all hugely important. Note, though, that higher-order logic tends to drift off into set theory land: quantifying over sets and functions is a radical step that provides a huge boost in power, but also introduces a host of difficulties. Some would say that SOL is really set theory.

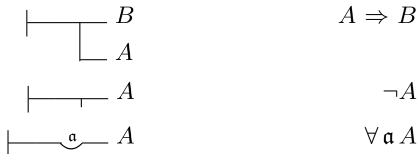
Nowadays, first-order logic is the general workhorse in math and TCS.



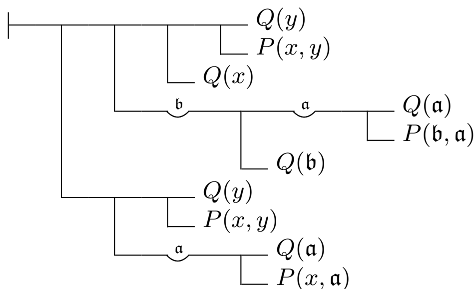
Both first- and higher-order logic were introduced more or less by Frege in 1879.

In 1879 Frege published his *Begriffsschrift*, which translates roughly as "concept script." Frege's system essentially invented modern quantified logic and he used it to try to formalize mathematics in his *Grundgesetze* (the "fundamental laws").

Unfortunately, Frege developed a two-dimensional notation system.



This may seem harmless, but if one constructs larger formulae from these primitives, things start to look quite ominous.



His notation system helped greatly to sink his whole enterprise.
As did the fact that his system was found to be inconsistent by Russell (whose response was to develop *type theory*).

$\bot$, $\top$	constants false, true
p, q, r, ...	propositional variables
$\neg$	not
$\wedge$	and, conjunction
$\vee$	or, disjunction
$\Rightarrow$	conditional (implies)

Negation is unary, all the others a binary.

A "structure" here is just an assignment of truth values to variables, an assignment or valuation

$$\sigma : \mathsf{Var} \to \mathbf{2}$$

We have seen that an accepting computation of a polynomial time Turing machine $\mathcal{M}$ can be translated into a question of whether a certain Boolean formula $\Phi_{\mathcal{M}}$ has a satisfying truth assignment.

The trick is to use lots and lots of Boolean variables to code up the whole computation.

One might wonder whether a more expressive logic would produce other interesting arguments along these lines: translate a machine into an "equivalent" formula.

We'll do this for finite state machines, and then again for Turing machines.

The main problem with propositional logic is that our translation from Turing machines is quite heavy-handed; in particular it has little to do with the way a computation of a TM would be defined ordinarily, using intuition.

More promising seems a system like first-order logic which serves as the standard workhorse in much of math and CS.

To wit, what is generally considered to be a "math proof" is an argument that *could be formalized* in FOL given some reasonable background theory (Dedekind-Peano, Zermelo-Fraenkel, von Neumann-Bernays-Gödel). Note the hedge: what is published nowadays in a standard math journal is just a proof sketch. And often just a sketch of a proof sketch. Or a sketch of a sketch of a proof sketch.

We add the following ingredients to propositional logic:

- quantifiers $\forall$ and $\exists$
- variables and constants for elements of some domain
- function symbols
- relation symbols

These all have their intuitive meaning.

Definition

A (first-order) structure is a set together with a collection of functions and relations on that set. The signature of a first-order structure is the list of arities of its functions and relations.

In order to interpret a formula we need something like

$$\mathcal{A} = \langle A; f_1, f_2, \ldots, R_1, R_2, \ldots \rangle$$

Here A is the carrier set of the structure. In addition, we have actual functions $f_i : A^{n_i} \to A$ and relations $R_i \subseteq A^{m_i}$ that interpret that symbols in the language (we'll fudge notation)†.

A, f_i and R_i all live in set theory la-la land, in general we have no computational access to these objects. In the case when they are all computable we have a computable FO structure.

†If you want to be really careful, write f for a function symbol, and $f^{\mathcal{A}}$ for its interpretation over the structure $\mathcal{A}$

$$\mathfrak{N} = \langle \mathbb{N}; +, *, S, 0, < \rangle$$

All of elementary arithmetic takes place in $\mathfrak{N}$, as does just about all of 15-151 and 15-251; not calculus, though—it requires substantially bigger guns (see below on higher-order logic).

This structure is completely natural and everyone has a good, intuitive understanding of what is going on in $\mathfrak{N}$—up to a point, that is.

The first serious attempts to axiomatize the natural numbers date back to the 1880s.

The standard description of $\mathfrak{N}$ in terms of first-order logic is Dedekind-Peano arithmetic. This involves in particular some sort of induction axioms, undoubtedly your favorite pastime in discrete math.

successor

$$S(x) \neq 0 \qquad S(x) = S(y) \Rightarrow x = y$$

addition

$$x + 0 = x \qquad x + S(y) = S(x + y)$$

multiplication

$$x \cdot 0 = 0 \qquad x \cdot S(y) = (x \cdot y) + x$$

order

$$\neg(x < 0) \qquad x < S(y) \Leftrightarrow x = y \vee x < y$$

Addition and multiplication are defined by recursion using successor as a more primitive concept.

Arithmetic operations alone are not enough, we are missing one essential feature of the natural numbers: induction. To capture induction we add the Induction Axiom:

$$\varphi(0) \land \forall\, x\, \big(\varphi(x) \Rightarrow \varphi(S(x))\big) \Rightarrow \forall\, x\, \varphi(x)$$

Strictly speaking, this is not an axiom but an axiom schema: we get one axiom for each choice of φ.

One might feel that these axioms completely pin down $\mathfrak{N}$, but that is far from true: there are weird models that look different from $\mathfrak{N}$, so-called non-standard models. On the plus side, $\mathfrak{N}$ is the only computable model.

As Alfred Tarski realized, one can give a fairly simple definition of what it means that a formula is true over a structure (using induction on the formula):

$$\mathcal{A} \models \varphi$$

We won't go through the details, it's all quite natural. For example, a universal quantifier is handled like so:

$$\mathcal{A} \models \forall x \, \varphi(x) \iff \text{for all } a \in \mathcal{A} : \mathcal{A} \models \varphi(a)$$

So when someone claims that some number theoretic assertion φ is true, that simply means that

$$\mathfrak{N} \models \varphi$$

In his famous 1936 paper, Turing showed that first-order logic is undecidable by constructing a computable function f such that

$$f : \text{Turing machines} \longrightarrow \text{FOL sentences}$$

with the property that

$$\mathcal{M} \text{ fails to halt} \iff f(\mathcal{M}) \text{ has a model}$$

To this end one can express an infinite chessboard in FOL, and then use the board to represent computations of the Turing machine much in the way we saw in the tiling problem.

The intended model here is the set of squares of the board; they are labeled by state/head information as usual. A little care is required to make sure no unintended model can ruin the construction.

1 Descriptive Complexity

2 **Words as Structures**

3 Existential SOL

Recall our basic method to express decision problems: we think of the Yes-instances as a language $L \subseteq \mathbf{2}^\star$.

This approach has the great advantage that we can directly apply Turing machines to solving decision problems, so that the time/space/whatever complexity of the machine can be used to measure the complexity of the decision problem.

It's also slightly weird since it produces some junk theorems[†]: $\mathbb{NP}$ is closed under Kleene star. Who really cares about the Kleene star of a language of Yes-instances?

At any rate, we want to use logic rather than Turing machines. The question is: what sort of logic is appropriate to describe a language?

[†]These are really irrelevant, but very suitable to torturing students.

Wild Idea: Can we think of a single word as a structure?

And then concoct some sort of logic to describe the properties of the word, viewed as a structure?

This may seem a bit weird, but bear with me. First, we need to fix an appropriate language for our logic.

As always, we want at least propositional logic: logical connectives “not,” “and,” “or,” and so forth.

We will have variables x, y, z, ... that range over positions in a word, integers in the range 1 through n where n is the length of the word.

We allow the following basic predicates between variables:

$$x < y \qquad\qquad x = y$$

Of course, we can get, say, $x \geq y$ by Boolean operations.

Most importantly, we write

$$\mathbf{a}(x)$$

for “there is a letter a in position x.”

We allow quantification for position variables.

$$\exists\, x\, \varphi \qquad \forall\, x\, \varphi$$

For example, the formula

$$\exists\, x, y\, (x < y \wedge \mathbf{a}(x) \wedge \mathbf{b}(y))$$

intuitively means “somewhere there is an a and somewhere, to the right of it, there is a b.”

The formula

$$\forall\, x, y\, (\mathbf{a}(x) \wedge \mathbf{b}(y) \Rightarrow x < y)$$

intuitively means “all the as come before all the bs.”

We need some notion of truth, "formula φ holds over word w" or "word w is a model for formula φ"

$$w \models \varphi$$

where w is a word and φ a sentence in $\mathrm{MSO}[<]$.

We won't give a formal definition, but the basic idea is simple: Let $|w| = n$:

- the variables range over $[n] = \{1, 2, \ldots, n\}$,
- $x < y$ means: position x is to the left of position y ,
- $x = y$: well . . . ,
- for the $\mathbf{a}(x)$ predicate we let

$$\mathbf{a}(x) \iff w_x = a$$

$$aaacbbb \models \forall x \left(\mathbf{a}(x) \vee \mathbf{b}(x) \vee \mathbf{c}(x)\right)$$

$$aaaabbb \models \exists x, y \left(x < y \wedge \mathbf{a}(x) \wedge \mathbf{b}(y)\right)$$

$$bbbaaaa \not\models \exists x, y \left(x < y \wedge \mathbf{a}(x) \wedge \mathbf{b}(y)\right)$$

$$aaaabbb \models \exists x, y \left(x < y \wedge \neg\exists z \, (x < z \wedge z < y) \wedge \mathbf{a}(x) \wedge \mathbf{b}(y)\right)$$

$$aaacbbb \not\models \exists x, y \left(x < y \wedge \neg\exists z \, (x < z \wedge z < y) \wedge \mathbf{a}(x) \wedge \mathbf{b}(y)\right)$$

$$aaacbbb \models \exists x \left(\mathbf{c}(x) \Rightarrow \forall y \, (x < y \Rightarrow \mathbf{b}(y))\right)$$

$$aaaaaa \models \exists x \left(\mathbf{c}(x) \Rightarrow \forall y \, (x < y \Rightarrow \mathbf{b}(y))\right)$$

Very good, but recall that we are not really interested in single words, we want languages, sets of words. No problem, for any sentence φ, we can consider the collection of all words that satisfy φ:

$$\mathcal{L}(\varphi) = \{\, w \in \Sigma^\star \mid w \models \varphi \,\}.$$

So our key idea is that the “complexity” of $\mathcal{L}(\varphi)$ is just the complexity of the formula φ.

The hope is that the right logic, and perhaps constraints of the type of formula used, will produce interesting collections of languages, i.e., complexity classes.

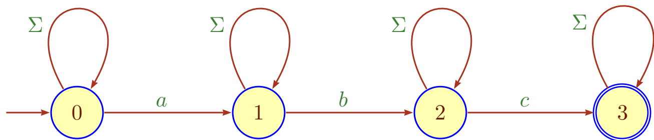
Example

We can obtain scattered subwords:

$$\varphi \equiv \exists\, x, y, z \left(x < y \wedge y < z \wedge \mathbf{a}(x) \wedge \mathbf{b}(y) \wedge \mathbf{c}(z) \right)$$

Then $w \models \varphi$ iff $w \in \Sigma^\star a \Sigma^\star b \Sigma^\star c \Sigma^\star$.

You might feel that this is a complicated formula for a fairly simple concept, but note that the analogous formula φ_u for an arbitrary scattered subword u has length $|u|$ and is trivial to construct.



The natural (nondeterministic) automaton is quite similar to the formula.

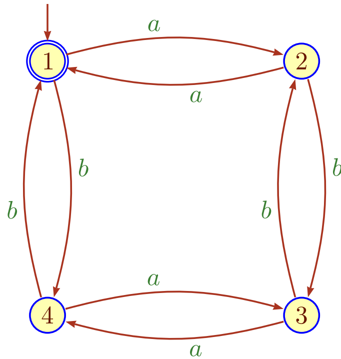
If one experiments a bit more, one cannot fail to notice that the languages $\mathcal{L}(\varphi)$ appear to be regular.

If you are the kind of person that jumps to conclusion you might suspect that we get exactly the regular languages from our little logic.

Here is the first serious challenge to this conjecture, the double-parity language

$$L_{e,e} = \{ x \in \{a, b\}^\star \mid \#_a x, \#_b x \text{ even} \}$$

This language has a trivial 4-state DFA, but building a corresponding formula φ seems rather difficult.



$\varphi = ??????$

This and other examples lead one to suspect that first-order logic is a bit too weak to produce all regular languages.

In logic, if FOL does not work, one turns to second-order logic. In our case, it turns out we need only a weak subsystem where second-order quantification is restricted to just sets of individuals. By contrast, we cannot quantify over, say, binary relations or functions.

Notation:

$$\exists X \qquad \forall X$$

$$z \in X \qquad X(z)$$

Example: Reachability

Lastly, consider a digraph, a single binary edge relation E.

We can express the assertion that there is a path from s to t as follows:

$$\forall X \left(X(s) \land \forall u, v \, (X(u) \land u \to v \Rightarrow X(v)) \Rightarrow X(t) \right)$$

Again, FOL is not strong enough to express path existence in general (and thus other concepts like connectivity).

Again assume a total order $\leq$. We can express the assertion that we have a well-order in terms of the least-element principle: every non-empty set has a least element.

$$\forall X \bigl(\exists z\, X(z) \Rightarrow \\ \exists u\, (X(u) \land \forall z\, (X(z) \Rightarrow u \leq z))\bigr)$$

This is the critical property of the natural numbers with the standard order, and cannot be expressed in FOL.

Let's ignore words for a moment, and just try to get an idea what kinds of concepts one can express in MSO.

Assuming a total order $\leq$, we can express the assertion that every bounded set has a least upper bound:

$$\forall X \big(\exists z\, X(z) \wedge \exists u \forall z\, (X(z) \Rightarrow z \leq u) \Rightarrow$$

$$\exists u\, (\forall z\, (X(z) \Rightarrow z \leq u) \wedge \forall y\, (\forall z\, (X(z) \Rightarrow z \leq y) \Rightarrow u \leq y))\big)$$

This is the critical property of the standard order on the reals, and cannot be expressed in FOL.

We allow second-order variables X, Y, Z, ... that range over sets of positions in a word.

$$\exists X \varphi \qquad \forall X \varphi$$

Again: sets of positions are all there is; we do not have variables in our language for, say, binary relations on positions (we do not use full SOL).

This system is called monadic second-order logic (with less-than), written $\mathrm{MSO}[<]$.

Often one would like to talk about the "next position" $x{+}1$ in a word.

We have seen how to express this using a FO quantifier:

$$y = x + 1 \iff x < y \land \forall\, z\, (x < z \Rightarrow y \leq z)$$

On the other hand, write closed(X) for the formula $\forall\, z\, (X(z) \Rightarrow X(z+1))$. Then

$$x < y \iff x \neq y \land \forall\, X\, (X(x) \land \mathsf{closed}(X) \Rightarrow X(y))$$

So $x{+}1$ and $x < y$ have the exact same expressiveness (though $x < y$ is slightly more useful than $x{+}1$). This is sometimes written as $\mathrm{MSO}[<] = \mathrm{MSO}[+1]$.

Example

Write $\mathsf{even}(X)$ to mean that X has even cardinality and consider

$$\varphi \equiv \exists X \left(\forall x \,(\mathbf{a}(x) \iff X(x)) \wedge \mathsf{even}(X)\right)$$

Then $w \models \varphi$ iff the number of as in w is even.

We're cheating, of course; we need to show that the predicate $\mathsf{even}(X)$ is definable in our setting. This is tedious but not really hard:

$$\mathsf{even}(X) \iff \exists Y, Z \,(X = Y \cup Z \wedge \emptyset = Y \cap Z \wedge \mathsf{alt}(Y, Z))$$

Here $\mathsf{alt}(Y, Z)$ is supposed to express that the elements of Y and Z strictly alternate as in

$$y_1 < z_1 < y_2 < z_2 < \ldots < y_k < z_k$$

$$\begin{aligned}
X = Y \cup Z &\iff \forall\, u\,(X(u) \Leftrightarrow Y(u) \vee Z(u)) \\
\emptyset = Y \cap Z &\iff \neg\exists\, u\,(Y(u) \wedge Z(u)) \\
\mathsf{alt}(Y, Z) &\iff \exists\, y \in Y\, \forall\, x < y\,(\neg Z(x)) \wedge \\
&\qquad \exists\, z \in Z\, \forall\, x > z\,(\neg Y(x)) \wedge \\
&\qquad \forall\, y \in Y\, \exists\, z \in Z\,(y < z \wedge \forall\, x\,(y < x < z \Rightarrow \neg Y(x) \wedge \neg Z(x))) \\
&\qquad \forall\, z \in Z\, \exists\, y \in Y\,(y < z \wedge \forall\, x\,(y < x < z \Rightarrow \neg Y(x) \wedge \neg Z(x)))
\end{aligned}$$

Exercise

The alt *formula does not handle the case where Y and Z are empty; fix this.*
Show that one can check if the number of as is a multiple of k, for any fixed k.

Definition

A language L is $\mathrm{MSO}[<]$ definable (or simply $\mathrm{MSO}[<]$) if there is some sentence φ such that

$$L = \mathcal{L}(\varphi) = \{\, w \in \Sigma^\star \mid w \models \varphi \,\}.$$

Our examples suggest the following theorem that connects complexity with definability:

Theorem (Buechi/Elgot/Trakhtenbrot 1960/1961/1961)

A language is regular if, and only if, it is $\mathrm{MSO}[<]$ *definable.*

Part (2) is by straightforward induction on φ, but there is the usual technical twist: we need to deal not just with sentences but also with free variables. Since we don't have a formal semantics we will not give details of this construction.

For the other direction, suppose we have a DFA $\mathcal{A}$ that recognizes L. Think of the states Q of $\mathcal{A}$ as colors, and use the formula to color the letters of some input word $w \in \Sigma^\star$ according to the transition function of the machine. For example, if $w = \ldots ab \ldots$ and a is colored p, then b must be colored green.

Colors come down to a partition of $[|w|]$ and can be handled by an existential second-order formula.

Then $\mathcal{A}$ accepts iff there is a coloring starting with the initial color and ending in a final color.

Inquisitive minds will want to know what happened to plain first-order logic? It must correspond to some subset of regular, but is there any meaningful characterization of the languages definable by FO formulae?

A language $L \subseteq \Sigma^\star$ is star-free iff it can be generated from $\emptyset$ and the singletons $\{a\}$, $a \in \Sigma$, using only operations union, concatenation and complement (but not Kleene star).

Note well: $a^\star b^\star a^\star$ is star-free.

Theorem

A language $L \subseteq \Sigma^\star$ is $\mathrm{FOL}[<]$ definable if, and only if, L is star-free.

1 Descriptive Complexity

2 Words as Structures

3 Existential SOL

Regular and star-free languages are nice, but nowhere near where we want to be in complexity theory. How do we get an alternative description of a complexity class like $\mathbb{NP}$?

We need a stronger logic to get up there. Our goal is to establish the following result.

Theorem (Fagin 1974)

The complexity class $\mathbb{NP}$ corresponds to existential second-order logic.

We will write existential SOL as ∃SO.

∃SO means we are considering formulae of the kind

$$\exists X_1, X_2, \ldots, X_k \, \Phi$$

where Φ is first-order: there are no second-order quantifiers other than the existential ones up front.

But now the X_i need not be monadic, in particular we will be allowed to quantify over k-ary relations: $\exists X \subseteq A^k \ldots$ for any $k \geq 1$.

We really should write something like $X^{(k)}$, but we won't bother.

So far, we have focused on word structures, we now need to handle more general structures. It is enough to deal with relational structures where there are no function symbols:

$$\mathcal{A} = \langle A; R_1, R_2, \ldots \rangle$$

This works since we can express functions as relations.

$$F(x, y) \iff f(x) = y$$

and actually makes things easier from a conceptual perspective: $f(g(x)) = y$ translates into the more transparent $\exists\, z \,\big(G(x, z) \land F(z, y)\big)$.

At any rate, validity for relational structures has the same complexity as for general ones with function symbols.

To model digraphs, we just need one binary predicate E for edges. We can then express 3-Colorability as a $\exists\mathrm{SO}$ formula as follows:

$$\exists\, X, Y, Z \Big(\forall\, u\, (X(u) \lor Y(u) \lor Z(u)) \land \forall\, u, v\, (E(u,v) \Rightarrow \\ \neg(X(u) \land X(v)) \land \neg(Y(u) \land Y(v)) \land \neg(Z(u) \land Z(v)))\Big)$$

So this is really just the standard definition of 3-colorability, spelled out in formal notation.

With minor effort, we can concoct similar descriptions for all our $\mathbb{NP}$ problems.

Hence it is $\mathbb{NP}$-hard to determine the validity of a $\exists\mathrm{SO}$ formula.

Theorem

Validity of an existential second-order formula is in $\mathbb{NP}$*.*

Suppose we have a formula

$$\Psi = \exists\, X_1, X_2, \ldots, X_k\, \Phi\big(X_1, \ldots, X_k\big)$$

where Φ is first-order, but may very well contain first-order quantifiers. So we should first worry about first-order formulae that have relational constants (which come from instantiating the existential quantifiers in Ψ).

Lemma

Validity of a first-order sentence can be determined in logarithmic space.

Proof.

Let us write

$$\Phi = \exists\, x_1\, \forall\, x_2\, \ldots Q_\ell\, x_\ell\, \phi(\boldsymbol{x})$$

Set $n = |A|$, so elements of the carrier set can be represented in $\log n$ bits.

If $\ell = 0$ we only have to deal with atomic formulae and logical connectives in ϕ. This can clearly be handled by a log-space machine by repeated table lookup of the relations involved.

By induction, we may assume we can handle $\ell - 1$ quantifiers. Assume the next quantifier is existential (it really makes no difference). Then we can add a loop that tries to find a suitable witness for x_1, using the machine that handles the remainder of the formula. This produces another log-space machine.

□

Let's return to the second-order formula Ψ. For the sake of simplicity, let us only consider one second-order variable X of arity 2, $X \subseteq A \times A$. We can represent X by a bitvector B of length n^2.

A nondeterministic TM can guess these bits in polynomial time. Once we have $B \in \mathbf{2}^{n \times n}$, we need to verify that $\Phi(B)$ actually holds.

As we have just seen, this can be handled in deterministic logarithmic space. Done.

□

So testing a $\exists\mathrm{SO}$ formula for validity over some structure is in $\mathbb{NP}$.

This already follows from 3-Colorability mentioned above. However, it is instructive to concoct a direct proof. Suppose $\mathcal{M}$ is some deterministic polynomial time verifier. We want to express the computation of $\mathcal{M}$ on some instance $\boldsymbol{x}$, given a witness $\boldsymbol{w}$.

For simplicity assume that the running time of $\mathcal{M}$ on an input of size n is $N = n^k - 1$: this allows us to think of both time and space as being written in k-digit base n numbers.

We can write a configuration in a computation as a word in

$$\Gamma^\star \, (Q \times \Gamma) \, \Gamma^\star$$

of length N where Γ is the tape alphabet of $\mathcal{M}$ and Q the state set. So the whole computation of $\mathcal{M}$ is a $N \times N$ table of letters in Γ, augmented in one place per row by a state (remember tiling?).

In an accepting computation, the first and the last row look like

$$\underset{\sqcup}{q_0}\, w_1 \ldots w_m \,\#\, x_1 \ldots x_n \sqcup \sqcup \ldots \sqcup$$

$$\underset{\sqcup}{q_Y} \sqcup \ldots \sqcup$$

Here $\boldsymbol{x}$ is the actual instance, and $\boldsymbol{w}$ the corresponding witness.

Since we are dealing with existential sentences, the witness part comes for free: given input x, we can always write something like

$$\exists\, W\, \Phi(W, \boldsymbol{x}, \ldots)$$

So we can safely pretend that $\boldsymbol{w}$ is part of the input.

$$\begin{array}{cccccccccccccc}
\stackrel{q_0}{\sqcup} & 0 & 1 & 0 & \# & a & b & c & d & \sqcup & \sqcup & \sqcup & \sqcup & \sqcup \\
\sqcup & \stackrel{p}{0} & 1 & 0 & \# & a & b & c & d & \sqcup & \sqcup & \sqcup & \sqcup & \sqcup \\
\sqcup & 0 & \stackrel{p}{1} & 0 & \# & a & b & c & d & \sqcup & \sqcup & \sqcup & \sqcup & \sqcup \\
\sqcup & 0 & 1 & \stackrel{p}{0} & \# & a & b & c & d & \sqcup & \sqcup & \sqcup & \sqcup & \sqcup \\
\sqcup & 0 & 1 & 0 & \stackrel{p}{\#} & a & b & c & d & \sqcup & \sqcup & \sqcup & \sqcup & \sqcup \\
\sqcup & 0 & 1 & 0 & \# & \stackrel{q}{a} & b & c & d & \sqcup & \sqcup & \sqcup & \sqcup & \sqcup \\
\sqcup & 0 & 1 & 0 & \stackrel{q'}{\#} & a & b & c & d & \sqcup & \sqcup & \sqcup & \sqcup & \sqcup
\end{array}$$

A typical initial segment of a computation of $\mathcal{M}$.
Again, this is very similar to the tiling problem.

Write C for the computation of $\mathcal{M}$, the tableaux we just constructed. Recall that time will be expressed as a k-tuple $\boldsymbol{t} = t_0, t_1, \ldots, t_{k-1}$ of elements in the carrier set $\{0, 1, \ldots, n-1\}$; ditto for space.

Let $\gamma = |Q \times \varGamma \cup \varGamma|$ the number of possible symbols in C.

We use $2k$-ary predicates X_a, $1 \leq a \leq \gamma$, with the intent that

$$X_a(\boldsymbol{s}, \boldsymbol{t}) \iff C(\boldsymbol{s}, \boldsymbol{t}) = a$$

We have for example

$$\forall\, \boldsymbol{s}, \boldsymbol{t}\, \exists\, a\, X_g(\boldsymbol{s}, \boldsymbol{t}) \wedge \forall\, \boldsymbol{s}, \boldsymbol{t}, a, b \left(X_a(\boldsymbol{s}, \boldsymbol{t}) \wedge X_b(\boldsymbol{s}, \boldsymbol{t}) \Rightarrow a = b \right)$$

We need to make sure that the entries in the table change only according to the rules of the Turing machine: for the most part, row k is copied to row $k+1$, but close to the position of the $\Gamma \times Q$ symbol there may be changes.

In essence, we need express the transition function of $\mathcal{M}$ as a formula. For example, let $g = p/a \in Q \times \Gamma$ and suppose $\delta(p, a) = (q, b, +1)$. Letting $g' = q/c$ we can pin down this transition by the formula

$$\forall\, \boldsymbol{s}, \boldsymbol{t} \left(X_g(\boldsymbol{s}, \boldsymbol{t}) \wedge X_c(\boldsymbol{s}{+}1, \boldsymbol{t}) \Rightarrow X_b(\boldsymbol{s}, \boldsymbol{t}{+}1) \wedge X_{g'}(\boldsymbol{s}{+}1, \boldsymbol{t}{+}1) \right)$$

Here $\boldsymbol{s}+1$ is supposed to be the string representing the successor of the value of $\boldsymbol{s}$.

Transitions $\delta(p, a) = (q, b, -1)$ are analogous.

In the end, the formula will look somewhat like

$$\exists\, W, X_1, \ldots, X_\gamma\, \exists\, \boldsymbol{u}\, \forall\, \boldsymbol{v}\ \ldots \varPhi(W, X_1, \ldots, X_\gamma, \boldsymbol{u}, \boldsymbol{v}, \ldots)$$

and this formula will be valid iff the verifier $\mathcal{M}$ accepts $\boldsymbol{x}$ together with some suitable witness $\boldsymbol{w}$.

The formula is messy, but it is easy to construct given $\mathcal{M}$ and $\boldsymbol{x}$ in polynomial time.

Hence we can translate any problem in $\mathbb{NP}$ into a corresponding $\exists\mathrm{SO}$ formula.

Ages ago, in the proof of the Cook-Levin theorem, we showed how to translate computations of a Turing machine into a formula of propositional logic. The formula is rather big and somewhat clumsy, simply because our target logic is very limited.

Here we translate into second-order logic, a much richer and more expressive language. As a consequence, the translation is much more natural, we are really just rewriting the definition of a computation in a slightly more formal way than the customary standard.

The Büchi/Elgot theorem establishes a connection between regular languages (aka constant space) and $\mathrm{MSO}[<]$.

As we have just seen, Fagin's theorem shows that $\mathbb{NP}$ corresponds exactly to existential SOL.

To establish Fagin's result, we used existential witnesses:

$$\boldsymbol{x} \in L \iff \exists\, \boldsymbol{X}\ \mathcal{M}(\boldsymbol{X}, \boldsymbol{x}) \downarrow$$

This is very similar to restricting quantified Boolean formulae to just Σ_1, which is just another way of describing satisfiability.

One might suspect that we can push much further ...

By allowing more quantifiers as in

$$x \in L \iff \exists\, X_1\, \forall\, X_2\, \exists\, X_3\, \ldots \mathcal{M}(\boldsymbol{X}, \boldsymbol{x}) \downarrow$$

we can naturally climb up the polynomial hierarchy. Recall, though, that we do not know whether PH is a proper hierarchy, it might collapse at some level (and, in fact, right at the bottom).

But this much we do know:

- PH corresponds to SOL.
- PSPACE corresponds to SOL plus a transitive closure operator.