

UCT

Descriptive Complexity

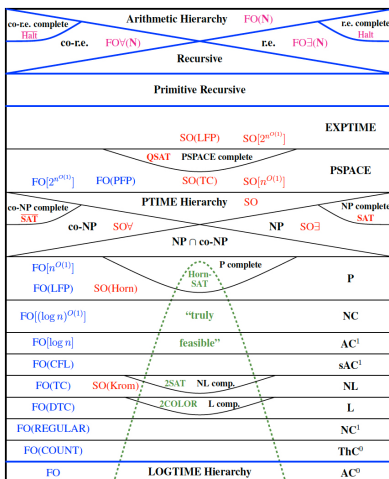
K. SUTNER
CARNEGIE MELLON UNIVERSITY
SPRING 2022



- 1 Descriptive Complexity
- 2 Words as Structures
- 3 Existential SOL

Immerman's Zoo

2



abstract computation

civilized computation

feasible computation

Where Are We?

3

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.

Logic to the Rescue

4

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

Main 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 Peano arithmetic or Zermelo-Fraenkel set theory (typically using first-order logic): these are both sledge hammers, very convenient and powerful, but not subtle.

What's Logic?

5

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

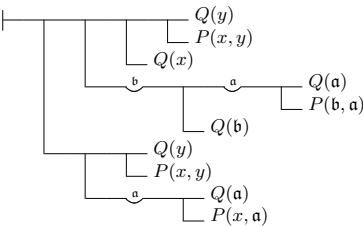
The effectiveness requirements depend a bit on the system in question, minimally we would want that it is decidable whether a string is a formula. Also, it should be decidable whether an object is a valid proof (this says nothing about proof search).

At any rate, we are here mostly interested in syntax and semantics.

- propositional logic
- equational logic
- first-order logic
- higher-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 HOL is really set theory.

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



It may seem that syntax questions are trivial, but just take a look at Frege's system in his Begriffsschrift, where he essentially invented modern quantified logic.

\perp, \top	constants false, true
p, q, r, \dots	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**

$$\sigma : \text{Var} \rightarrow 2$$

We have seen that an accepting computation of a polynomial time Turing machine \mathfrak{M} can be translated into a question of whether a certain Boolean formula $\Phi_{\mathfrak{M}}$ 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 they way a computation of a TM would be defined ordinarily.

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 in a journal is just 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

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

The set A is the carrier set of the structure.

We have actual functions $f_i : A^{n_i} \rightarrow A$ and relations $R_i \subseteq A^{m_i}$ that interpret that symbols in the language (we'll fudge notation).

$$\mathfrak{N} = \langle \mathbb{N}; +, *, 0, 1, < \rangle$$

All elementary arithmetic takes place in \mathfrak{N} (and most of 15-151 and 15-251).

The standard description of \mathfrak{N} as a FO structure is Dedekind-Peano arithmetic (and in particular some sort of induction axiom).

Truth of FO logic over \mathfrak{N} is undecidable, even for Σ_1 formulae.

On the upside, the structure is obviously computable (actually, it's the only computable model of the axioms).

Note that a first-order structure is not all that different from a data type. To wit, we are dealing with a

- collection of objects,
- operations on these objects, and
- relations on these objects.

In the case where the carrier set is finite (actually, finite and small) we can in fact represent the whole FO structure by a suitable data structure (for example, explicit lookup tables). For infinite carrier sets, things are a bit more complicated.

Data types (or rather, their values) are manipulated in programs, we are here interested in describing properties of structures using the machinery of FOL.

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

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

with the property that

$$\mathcal{M} \text{ fails to halt} \iff \phi(\mathcal{M}) \text{ is satisfiable}$$

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

We are used to coding finite objects as words over some alphabet.

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

And, of course, use logic to describe the properties of the word/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, \dots 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:

$$x < y \quad x = y$$

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

Most importantly, we write

$$Q_a(x)$$

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

We allow quantification for position variables.

$$\exists x \varphi \quad \forall x \varphi$$

For example, the formula

$$\exists x, y (x < y \wedge Q_a(x) \wedge Q_b(y))$$

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

The formula

$$\forall x, y (Q_a(x) \wedge Q_b(y) \Rightarrow x < y)$$

intuitively means “all the a s come before all the b s.”

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 $\text{MSO}[\prec]$.

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

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

$$Q_a(x) \iff w_x = a$$

$$aaacbbb \models \forall x (Q_a(x) \vee Q_b(x) \vee Q_c(x))$$

$$aaabbb \models \exists x, y (x < y \wedge Q_a(x) \wedge Q_b(y))$$

$$bbbaaa \not\models \exists x, y (x < y \wedge Q_a(x) \wedge Q_b(y))$$

$$aaabbb \models \exists x, y (x < y \wedge \neg \exists z (x < z \wedge z < y) \wedge Q_a(x) \wedge Q_b(y))$$

$$aaacbbb \not\models \exists x, y (x < y \wedge \neg \exists z (x < z \wedge z < y) \wedge Q_a(x) \wedge Q_b(y))$$

$$aaacbbb \models \exists x (Q_c(x) \Rightarrow \forall y (x < y \Rightarrow Q_b(y)))$$

$$aaaaaa \models \exists x (Q_c(x) \Rightarrow \forall y (x < y \Rightarrow Q_b(y)))$$

Very good, but recall that we 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 :

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

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

In FOL, we can hardwire factors. For example, to obtain a factor abc let

$$\varphi \equiv \exists x, y, z (y = x+1 \wedge z = y+1 \wedge Q_a(x) \wedge Q_b(y) \wedge Q_c(z))$$

Then $w \models \varphi$ iff $w \in \Sigma^* abc \Sigma^*$.

You might object to the use of “ $y = x+1$ ” which is not part of our language. No worries, it's just an abbreviation:

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

This is quite typical: one defines a small language that is easy to handle, and then boosts usability by adding abbreviations.

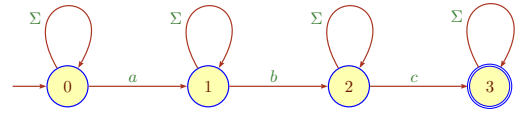
Example

Instead of factors we can similarly get (scattered) subwords by dropping the adjacency condition for the positions:

$$\varphi \equiv \exists x, y, z (x < y \wedge y < z \wedge Q_a(x) \wedge Q_b(y) \wedge Q_c(z))$$

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

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



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

Example

We can split a word into two parts as in

$$\varphi \equiv \exists x \forall y ((y \leq x \Rightarrow Q_a(y)) \wedge (y > x \Rightarrow Q_b(y))) \vee \forall x (Q_b(x))$$

Then $w \models \varphi$ iff $w \in a^* b^*$.

Example

Let $\text{first}(x)$ be shorthand for $\forall z (x \leq z)$, and $\text{last}(x)$ shorthand for $\forall z (x \geq z)$. Then

$$\varphi \equiv \exists x, y (\text{first}(x) \wedge Q_a(x) \wedge \text{last}(y) \wedge Q_b(y))$$

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

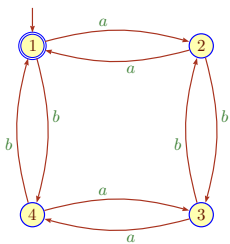
One cannot fail to notice that all the languages $\mathcal{L}(\varphi)$ we have seen so far are in fact regular.

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.

Alas, consider the double-parity language

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

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



$$\varphi = \text{?????}$$

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 (relations of type R^1). By contrast, we cannot quantify over, say, binary relations or functions.

Notation:

$$\exists X \quad \forall X$$

$$x \in X \quad X(x)$$

Example: Least Upper Bounds

30

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:

$$\forall X \left(\exists z X(z) \wedge \exists x \forall z (X(z) \Rightarrow z \leq x) \Rightarrow \right. \\ \left. \exists x (\forall z (X(z) \Rightarrow z \leq x) \wedge \forall y (\forall z (X(z) \Rightarrow z \leq y) \Rightarrow x \leq y)) \right)$$

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

Example: Well-Order

31

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.

$$\forall X \left(\exists z X(z) \Rightarrow \right. \\ \left. \exists x (X(x) \wedge \forall z (X(z) \Rightarrow x \leq z)) \right)$$

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

Example: Reachability

32

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:

$$\forall X \left(X(s) \wedge \forall x, y (X(x) \wedge E(x, y) \Rightarrow X(y)) \Rightarrow X(t) \right)$$

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

MSO for Words

33

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

$$\exists X \varphi \quad \forall X \varphi \quad X(x)$$

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 **MSO[<]**.

Less-Than or Successor

34

In applications, the atomic relation $x < y$ is slightly more useful than $y = x + 1$, but either one would have the same expressiveness.

We have already seen

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

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

$$x < y \iff x \neq y \wedge \forall X (X(x) \wedge \text{closed}(X) \Rightarrow X(y))$$

This is sometimes written as $\text{MSO}[<] = \text{MSO}[+1]$.

Counting

35

Suppose we want at least three a s.

$$\exists x, y, z (x < y < z \wedge Q_a(x) \wedge Q_a(y) \wedge Q_a(z))$$

And at most three a s.

$$\exists x, y, z \forall u (Q_a(u) \Rightarrow u = x \vee u = y \vee u = z)$$

Exactly three is now obtained by conjunction, much easier than a product operation on finite state machines.

Example

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

$$\varphi \equiv \exists X \left(\forall x (Q_a(x) \iff X(x)) \wedge \text{even}(X) \right)$$

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

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

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

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

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

$$X = Y \cup Z \iff \forall u (X(u) \iff Y(u) \vee Z(u))$$

$$\emptyset = Y \cap Z \iff \neg \exists u (Y(u) \wedge Z(u))$$

$$\text{alt}(Y, Z) \iff \exists y \in Y \forall x < y (\neg Z(x)) \wedge$$

$$\exists z \in Z \forall x > z (\neg Y(x)) \wedge$$

$$\forall y \in Y \exists z \in Z (y < z \wedge \forall x (y < x < z \Rightarrow \neg Y(x) \wedge \neg Z(x)))$$

$$\forall z \in Z \exists y \in Y (y < z \wedge \forall x (y < x < z \Rightarrow \neg Y(x) \wedge \neg Z(x)))$$

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 a s is a multiple of k , for any fixed k .

Definition

A language L is **MSO[<] definable** (or simply MSO[<]) if there is some sentence φ such that

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

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

Theorem (Büchi 1960, Elgot 1961)

A language is regular if, and only if, it is MSO[<] definable.

Obviously, the proof comes in two parts:

- For every regular language L , construct a sentence φ such that $L = \mathcal{L}(\varphi)$.
- For every sentence φ , show that the language $\mathcal{L}(\varphi)$ is regular.

We should expect part (1) to be harder since there is no good inductive structure to exploit.

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.

We may safely assume that the regular language L is given by a DFA $M = \langle Q, \Sigma, \delta, q_0, F \rangle$.

For simplicity assume $Q = [n]$ and $q_0 = 1$.

We have to construct a formula φ such that $w \models \varphi$ iff M accepts w .

Consider a trace of M on input w

$$q_0 \ w_1 \ q_1 \ w_2 \ q_2 \ \dots \ q_{m-1} \ w_m \ q_m.$$

Here m can be arbitrarily large.

We can think of states as being associated with the letters of the word as in

$$\begin{array}{cccccc} w_1 & w_2 & w_3 & \dots & w_m \\ q_0 & q_1 & q_2 & q_3 & \dots & q_m \end{array}$$

Thus, position $x = 1, \dots, m$ in the word is associated with state $\delta(q_0, w_1 \dots w_x)$.

In order to express this in a MSO[<] formula, we partition the set of positions $[m]$ into $n = |Q|$ blocks X_1, X_2, \dots, X_n such that

$$X_p(x) \iff \delta(q_0, w_1 \dots w_x) = p$$

Some of these blocks may be empty, but note that the number of blocks is always exactly n (which we can express as a formula).

But given state p in position x we can determine the state in position $x+1$ given w_{x+1} by a table lookup – which table lookup can be hardwired in a formula.

Technically, this is done by a formula

$$\Phi_{p,a} \equiv \forall x \left(X_p(x) \wedge Q_a(x+1) \Rightarrow X_{\delta(p,a)}(x+1) \right)$$

meaning “if at position x we are in state p and the next letter is an a , then the state in position $x+1$ is $\delta(p,a)$.”

Note that this is not quite right, we really need a non-existing position 0 corresponding to state q_0 .

Exercise

Figure out how to fix this little glitch. Also figure out how to express “the last state is final.”

Now consider the big conjunction of $\Phi_{p,a}$ where $p \in Q$ and $a \in \Sigma$. Add formulae that pin down the first and last state to arrive at a formula of the form

$$\varphi \equiv \exists X_1, \dots, X_n \Psi$$

where Ψ is first-order as indicated above. □

Note that in conjunction with the opposite direction of Büchi’s theorem, this result has the surprising consequence that every $\text{MSO}[<]$ formula is equivalent to a $\text{MSO}[<]$ formula containing only one block of existential second-order quantifiers.

Exercise

Fill in all the details in the last proof.

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 $L \subseteq \Sigma^*$ 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: $a^*b^*a^*$ is star-free.

Theorem

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

While we’re at it: star-free languages are quite interesting since they admit a purely algebraic characterization in terms of their **syntactic semigroups**: these are the semigroups generated by the functions $\delta_a : Q \rightarrow Q$ (under composition).

A semigroup is **aperiodic** if it contains only trivial subgroups (the idempotents of the semigroup).

Theorem (Schützenberger 1965)

A regular language is star-free if, and only if, its syntactic semigroup is aperiodic.

- 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 NP ?

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

Theorem (Fagin 1974)

The complexity class NP corresponds to existential second-order logic.

We will write existential SOL as $\exists\text{SO}$.

$\exists\text{SO}$ means we are considering formulae of the kind

$$\exists X_1, X_2, \dots, X_k \varphi$$

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: $\exists X^k \dots$ for any $k \geq 1$.

So far, we have focused on word structures, but it is not hard to generalize to other combinatorial objects such as ugraphs: we need a binary predicate E for edges.

3-Colorability of a ugraph is easily expressed as a $\exists\text{SO}$ formula:

$$\exists X, Y, Z \left(\forall u (X(u) \vee Y(u) \vee Z(u)) \wedge \forall u, v (E(u, v) \Rightarrow \neg(X(u) \wedge X(v)) \wedge \neg(Y(u) \wedge Y(v)) \wedge \neg(Z(u) \wedge Z(v))) \right)$$

Note that this is just the ordinary definition of 3-colorability, written in a formal notation. There is nothing mysterious going on.

Similar descriptions exist for all our NP problems.

Suppose we have a formula

$$\psi = \exists X_1, X_2, \dots, X_k \varphi$$

where φ is first-order.

To check whether φ holds we can guess the subsets X_i of the carrier set A , and then verify in polynomial time that φ holds. More precisely, if X_i has arity k we need to guess a subset of A^k , so we are still within polynomial time. The first-order quantifiers in φ are not a problem either: each just corresponds to a loop over some finite set.

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

Suppose \mathfrak{M} is some verifier. We want to express the computation of \mathfrak{M} on some input x , given some witness w .

For simplicity assume that the running time of \mathfrak{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

$$\Gamma^* (\Gamma \times Q) \Gamma^*$$

of length N where Γ is the tape alphabet of \mathfrak{M} and Q the state set. So the whole computation C of \mathfrak{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 C , the first and the last row look like

$$\begin{array}{c} q_0 \\ \vdots \end{array} x_1 \dots x_n \sqcup w_1 \dots w_m \sqcup \dots \sqcup \begin{array}{c} q_Y \\ \vdots \end{array}$$

Here x is the input, and 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

$$\exists W \Phi(x, W, \dots)$$

We'll just pretend w is part of the input.

$$\begin{array}{cccccccccccc} q_0 & a & b & c & \vdots & 0 & 1 & 0 & 1 & \vdots & \vdots & \vdots & \vdots \\ \vdots & p & a & b & c & \vdots & 0 & 1 & 0 & 1 & \vdots & \vdots & \vdots & \vdots \\ \vdots & a & p & b & c & \vdots & 0 & 1 & 0 & 1 & \vdots & \vdots & \vdots & \vdots \\ \vdots & a & b & p & c & \vdots & 0 & 1 & 0 & 1 & \vdots & \vdots & \vdots & \vdots \\ \vdots & a & b & c & p & \vdots & 0 & 1 & 0 & 1 & \vdots & \vdots & \vdots & \vdots \\ \vdots & a & b & c & \vdots & q & 0 & 1 & 0 & 1 & \vdots & \vdots & \vdots & \vdots \\ \vdots & a & b & c & q' & \vdots & 0 & 1 & 0 & 1 & \vdots & \vdots & \vdots & \vdots \end{array}$$

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

Recall that time will be expressed as a k -tuple $t = t_0, t_1, \dots, t_{n-1}$ of elements in the carrier set $\{0, 1, \dots, n-1\}$; ditto for space.

Let $\gamma = |I \times Q \cup Q|$.

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

$$X_g(s, t) \iff C(s, t) = g$$

We have for example

$$\forall s, t \exists g X_g(s, t) \wedge \forall s, t, g, g' (X_g(s, t) \wedge X_{g'}(s, t) \Rightarrow g = g')$$

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 $I \times Q$ symbol there may be changes.

In essence, we need express the transition function of \mathfrak{M} as a formula using assertions such as

$$X_g(s, t) \wedge X_{g'}(s+1, t) \Rightarrow X_h(s, t+1) \wedge X_{h'}(s+1, t+1)$$

In English, this is something like

If at time t and in position s there is a symbol a and the head is positioned at s and the state is p , then, at time $t+1$, in position s there is symbol a' and the head has moved one to the right, and is looking at the same symbol that was in position $s+1$ at time t .

In the end, the formula will look somewhat like

$$\exists W, X_1, \dots, X_\gamma \exists u \forall v \dots \Phi(W, X_1, \dots, X_\gamma, u, v, \dots)$$

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

The formula is messy, but it is easy to construct given \mathfrak{M} and x .

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

The Büchi/Elgot theorem establishes a connection between regular languages (aka constant space) and $\text{MSO}[<1]$.

As we have just seen, Fagin's theorem shows that NP corresponds exactly to existential SOL.

And one can push further:

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

$$x \in L \iff \exists u \mathfrak{M}(x, u) \downarrow$$

By allowing more quantifiers as in

$$x \in L \iff \exists u \forall v \exists w \dots \mathfrak{M}(x, u, v, w, \dots) \downarrow$$

we obtain a classification $\mathbb{P}, \Sigma_1^P = \text{NP}, \Pi_1^P = \text{co-NP}, \Sigma_2^P, \Pi_2^P, \dots$

The collection of all these problems is the **polynomial hierarchy** and seems to contain problem much harder than just NP .

For example, the question of whether a circuit has an equivalent circuit using at most k gates is Σ_2^P .

Alas, we do not know whether PH is a proper hierarchy.

But this much we do know:

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