Descriptive Complexity provides an alternative perspective on Computational Complexity.

**Descriptive Complexity**

• Complexity of a class of structures—e.g. a collection of graphs.

**Computational Complexity**

• Measure use of resources (space, time, etc.) on a machine model of computation;

• Complexity of a language—i.e. a set of strings.

**Descriptive Complexity**

• Measure the complexity of describing the collection in a formal logic, using resources such as variables, quantifiers, higher-order operators, etc.

There is a fascinating interplay between the views.
Signature and Structure

In general a signature (or vocabulary) $\sigma$ is a finite sequence of relation, function and constant symbols:

$$\sigma = (R_1, \ldots, R_m, f_1, \ldots, f_n, c_1, \ldots, c_p)$$

where, associated with each relation and function symbol is an arity.
A structure $A$ over the signature $\sigma$ is a tuple:

$$A = (A, R^A_1, \ldots, R^A_m, f^A_1, \ldots, f^A_n, c^A_1, \ldots, c^A_l),$$

where,

- $A$ is a non-empty set, the universe of the structure $A$,
- each $R^A_i$ is a relation over $A$ of the appropriate arity.
- each $f^A_i$ is a function over $A$ of the appropriate arity.
- each $c^A_i$ is an element of $A$. 
First-order Logic

Formulas of first-order logic are formed from the signature $\sigma$ and an infinite collection $X$ of variables as follows.

*terms* – $c$, $x$, $f(t_1, \ldots, t_a)$

Formulas are defined by induction:

- atomic formulas – $R(t_1, \ldots, t_a)$, $t_1 = t_2$
- Boolean operations – $\phi \land \psi$, $\phi \lor \psi$, $\neg \phi$
- first-order quantifiers – $\exists x \phi$, $\forall x \phi$
Queries

A formula $\phi$ with free variables among $x_1, \ldots, x_n$ defines a map $Q$ from structures to relations:

$$Q(\mathcal{A}) = \{a \mid \mathcal{A} \models \phi[a]\}.$$  

Any such map $Q$ which associates to every structure $\mathcal{A}$ a $(n$-ary) relation on $A$, and is isomorphism invariant, is called a $(n$-ary) query.

$Q$ is isomorphism invariant if, whenever $f : \mathcal{A} \to \mathcal{B}$ is an isomorphism between $\mathcal{A}$ and $\mathcal{B}$, it is also an isomorphism between $(\mathcal{A}, Q(\mathcal{A}))$ and $(\mathcal{B}, Q(\mathcal{B}))$.

If $n = 0$, we can regard the query as a map from structures to $\{0, 1\}$—a Boolean query.
For example, take the signature \((E)\), where \(E\) is a binary relation symbol. Finite structures \((V, E)\) of this signature are directed graphs.

Moreover, the class of such finite structures satisfying the sentence

\[
\forall x \neg Exx \land \forall x \forall y (Exy \rightarrow Eyx)
\]

can be identified with the class of \((loop-free, undirected)\) graphs.
For a first-order sentence $\phi$, we ask what is the *computational complexity* of the problem:

- **Input**: a structure $A$
- **Decide**: if $A \models \phi$

In other words, how complex can the collection of finite models of $\phi$ be?

In order to talk of the complexity of a class of finite structures, we need to fix some way of representing finite structures as strings.
Representing Structures as Strings

We use an alphabet $\Sigma = \{0, 1, #, −\}$.

For a structure $A = (A, R_1, \ldots, R_m, f_1, \ldots, f_l)$, fix a linear order $<$ on $A = \{a_1, \ldots, a_n\}$.

$R_i$ (of arity $k$) is encoded by a string $[R_i]_<$ of 0s and 1s of length $n^k$.

$f_i$ is encoded by a string $[f_i]_<$ of 0s, 1s and $-$s of length $n^k \log n$.

$$[A]_< = 1 \cdots 1 \# [R_1]_< \# \cdots \# [R_m]_< \# [f_1]_< \# \cdots \# [f_l]_<$$

The exact string obtained depends on the choice of order.
Naïve Algorithm

The straightforward algorithm proceeds recursively on the structure of $\phi$:

- Atomic formulas by direct lookup.
- Boolean connectives are easy.
- If $\phi \equiv \exists x \psi$ then for each $a \in A$ check whether
  \[
  (A, c \mapsto a) \models \psi[c/x],
  \]
  where $c$ is a new constant symbol.

This runs in time $O(\ln m)$ and $O(m \log n)$ space, where $m$ is the nesting depth of quantifiers in $\phi$.

\[
\text{Mod}(\phi) = \{A | A \models \phi\}
\]

is in logarithmic space and polynomial time.
The following problem:

**FO satisfaction**

*Input*: a structure $A$ and a first-order sentence $\phi$

*Decide*: if $A \models \phi$

is $\text{PSPACE}$-complete.

It follows from the $O(\ln^m)$ and $O(m \log n)$ space algorithm that the problem is in $\text{PSPACE}$.

How do we prove completeness?
We define *quantified Boolean formulas* inductively as follows, from a set $\mathcal{X}$ of *propositional variables*.

- A propositional constant $T$ or $F$ is a formula
- A propositional variable $X \in \mathcal{X}$ is a formula
- If $\phi$ and $\psi$ are formulas then so are: $\neg \phi$, $\phi \land \psi$ and $\phi \lor \psi$
- If $\phi$ is a formula and $X$ is a variable then $\exists X \phi$ and $\forall X \phi$ are formulas.

Say that an occurrence of a variable $X$ is *free* in a formula $\phi$ if it is not within the scope of a quantifier of the form $\exists X$ or $\forall X$. 
Given a quantified Boolean formula $\phi$ and an assignment of \textit{truth values} to its free variables, we can ask whether $\phi$ evaluates to \textit{true} or \textit{false}. In particular, if $\phi$ has no free variables, then it is equivalent to either \textit{true} or \textit{false}.

\textbf{QBF} is the following decision problem:

\textit{Input: a quantified Boolean formula $\phi$ with no free variables.}

\textit{Decide: whether $\phi$ evaluates to \textit{true}.}
Complexity of QBF

Note that a Boolean formula $\phi$ without quantifiers and with variables $X_1, \ldots, X_n$ is *satisfiable* if, and only if, the formula

$$\exists X_1 \cdots \exists X_n \phi \text{ is true.}$$

Similarly, $\phi$ is *valid* if, and only if, the formula

$$\forall X_1 \cdots \forall X_n \phi \text{ is true.}$$

Thus, $\text{SAT} \leq_L \text{QBF}$ and $\text{VAL} \leq_L \text{QBF}$ and so QBF is NP-hard and co-NP-hard.
In fact, QBF is PSPACE-complete.
PSPACE-hardness

To prove that QBF is PSPACE-hard, we want to show:

Given a machine $M$ with a polynomial space bound and an input $x$, we can define a quantified Boolean formula $\phi^M_x$ which evaluates to true if, and only if, $M$ accepts $x$.

Moreover, $\phi^M_x$ can be computed from $x$ in polynomial time (or even logarithmic space).

The number of distinct configurations of $M$ on input $x$ is bounded by $2^{nk}$ for some $k$ ($n = |x|$).
Each configuration can be represented by $nk$ bits.
Constructing $\phi^M_x$

We use tuples $A, B$ of $n^k$ Boolean variables each to encode *configurations* of $M$.

Inductively, we define a formula $\psi_i(A, B)$ which is *true* if the configuration coded by $B$ is reachable from that coded by $A$ in at most $2^i$ steps.

$$\psi_0(A, B) \equiv \text{“A = B” } \lor \text{“A } \rightarrow^*_M \text{ B”}$$
$$\psi_{i+1}(A, B) \equiv \exists Z \forall X \forall Y \ [(X = A \land Y = Z) \lor (X = Z \land Y = B)]$$
$$\Rightarrow \psi_i(X, Y)$$

$$\phi \equiv \psi_{n^k}(A, B) \land \text{“A = start” } \land \text{“B = accept”}$$
Reducing QBF to FO satisfaction

We have seen that *FO satisfaction* is in *PSPACE*. To show that it is *PSPACE*-complete, it suffices to show that \( \text{QBF} \leq_L \text{FO sat} \).

The reduction maps a quantified Boolean formula \( \phi \) to a pair \((\mathbb{A}, \phi^*)\) where \( \mathbb{A} \) is a structure with two elements: 0 and 1 and one unary relation \( T \) with \( T^\mathbb{A} = \{1\} \).

\( \phi^* \) is obtained from \( \phi \) by a simple inductive definition.
Expressive Power of FO

For any \emph{fixed} sentence $\phi$ of first-order logic, the class of structures $\text{Mod}(\phi)$ is in $\mathbf{L}$.

There are computationally easy properties that are not definable in first-order logic.

- There is no sentence $\phi$ of first-order logic such that $A \models \phi$ if, and only if, $|A|$ is even.
- There is no formula $\phi(E, x, y)$ that defines the transitive closure of a binary relation $E$.

We will see proofs of these facts later on.
Second-Order Logic

We extend first-order logic by a set of *relational variables*. For each \( m \in \mathbb{N} \) there is an infinite collection of variables \( \mathcal{V}^m = \{ V_1^m, V_2^m, \ldots \} \) of *arity* \( m \).

Second-order logic extends first-order logic by allowing *second-order quantifiers*

\[
\exists X \phi \quad \text{for } X \in \mathcal{V}^m
\]

A structure \( \mathcal{A} \) satisfies \( \exists X \phi \) if there is an \( m \)-ary relation \( R \) on the universe of \( \mathcal{A} \) such that \( (\mathcal{A}, X \rightarrow R) \) satisfies \( \phi \).
Existential Second-Order Logic

ESO—*existential second-order logic* consists of those formulas of second-order logic of the form:

$$\exists X_1 \cdots \exists X_k \phi$$

where $\phi$ is a first-order formula.
Examples

**Evenness**

This formula is true in a structure if, and only if, the size of the domain is even.

\[ \exists B \exists S \forall x \exists y B(x, y) \land \forall x \forall y \forall z B(x, y) \land B(x, z) \rightarrow y = z \]

\[ \forall x \forall y \forall z B(x, z) \land B(y, z) \rightarrow x = y \]

\[ \forall x \forall y S(x) \land B(x, y) \rightarrow \neg S(y) \]

\[ \forall x \forall y \neg S(x) \land B(x, y) \rightarrow S(y) \]
**Examples**

*Transitive Closure*

This formula is true of a pair of elements $a, b$ in a structure if, and only if, there is an $E$-path from $a$ to $b$.

\[
\begin{align*}
\exists P & \quad \forall x \forall y \ P(x, y) \rightarrow E(x, y) \\
& \quad \exists x P(a, x) \land \exists x P(x, b) \land \neg \exists x P(x, a) \land \neg \exists x P(b, x) \\
& \quad \forall x \forall y (P(x, y) \rightarrow \forall z (P(x, z) \rightarrow y = z)) \\
& \quad \forall x \forall y (P(x, y) \rightarrow \forall z (P(z, y) \rightarrow x = z)) \\
& \quad \forall x ((x \neq a \land \exists y P(x, y)) \rightarrow \exists z P(z, x)) \\
& \quad \forall x ((x \neq b \land \exists y P(y, x)) \rightarrow \exists z P(x, z))
\end{align*}
\]
3-Colourability
The following formula is true in a graph \((V, E)\) if, and only if, it is 3-colourable.

\[
\begin{align*}
\exists R \exists B \exists G & \quad \forall x (Rx \lor Bx \lor Gx) \land \\
& \quad \forall x (\neg (Rx \land Bx) \land \neg (Bx \land Gx) \land \neg (Rx \land Gx)) \land \\
& \quad \forall x \forall y (Exy \rightarrow (\neg (Rx \land Ry) \land \\
& \quad \neg (Bx \land By) \land \\
& \quad \neg (Gx \land Gy)))
\end{align*}
\]
Fagin’s Theorem

**Theorem (Fagin)**
A class $C$ of finite structures is definable by a sentence of *existential second-order logic* if, and only if, it is decidable by a *nondeterminisitic machine* running in polynomial time.

$$\text{ESO} = \text{NP}$$

One direction is easy: Given $A$ and $\exists P_1 \ldots \exists P_m \phi$.

*a nondeterministic machine can guess an interpretation for $P_1, \ldots, P_m$ and then verify $\phi$.***
Fagin’s Theorem

Given a machine $M$ and an integer $k$, there is an ESO sentence $\phi$ such that $\mathbb{A} \models \phi$ if, and only if, $M$ accepts $[\mathbb{A}]_<$, for some order $<$ in $n^k$ steps.

We construct a first-order formula $\phi_{M,k}$ such that

$$(\mathbb{A}, <, X) \models \phi_{M,k} \iff X \text{ codes an accepting computation of } M \text{ of length at most } n^k \text{ on input } [\mathbb{A}]_<$$

So, $\mathbb{A} \models \exists < \exists X \phi_{M,k}$ if, and only if, there is some order $<$ on $\mathbb{A}$ so that $M$ accepts $[\mathbb{A}]_<$ in time $n^k$. 

The formula $\phi_{M,k}$ is built up as the conjunction of a number of formulas. The first of these simply says that $<$ is a linear order:

$$\forall x(\neg x < x) \land$$  
$$\forall x \forall y (x < y \rightarrow \neg y < x) \land$$  
$$\forall x \forall y (x < y \lor y < x \lor x = y)$$  
$$\forall x \forall y \forall z (x < y \land y < z \rightarrow x < z)$$

We can use a linear order on the elements of $A$ to define a lexicographic order on $k$-tuples.
If $x = x_1, \ldots, x_k$ and $y = y_1, \ldots, y_k$ are $k$-tuples of variables, we use $x = y$ as shorthand for the formula $\bigwedge_{i \leq k} x_i = y_i$ and $x < y$ as shorthand for the formula

$$\bigvee_{i \leq k} ((\bigwedge_{j < i} x_j = y_j) \land x_i < y_i)$$

We also write $y = x + 1$ for the following formula:

$$x < y \land \forall z (x < z \to (y = z \lor y < z))$$
Constructing the Formula

Let $M = (K, \Sigma, s, \delta)$.

The tuple $X$ of second-order variables appearing in $\phi_{M,k}$ contains the following:

- $S_q$ a $k$-ary relation symbol for each $q \in K$
- $T_\sigma$ a $2k$-ary relation symbol for each $\sigma \in \Sigma$
- $H$ a $2k$-ary relation symbol
Intuitively, these relations are intended to capture the following:

- \( S_q(x) \) – the state of the machine at time \( x \) is \( q \).
- \( T_\sigma(x, y) \) – at time \( x \), the symbol at position \( y \) of the tape is \( \sigma \).
- \( H(x, y) \) – at time \( x \), the tape head is pointing at tape cell \( y \).

We now have to see how to write the formula \( \phi_{M,k} \), so that it enforces these meanings.
Initial state is $s$ and the head is initially at the beginning of the tape.

$$\forall x ( (\forall y \ x \leq y) \rightarrow S_s(x) \land H(x, x))$$

The head is never in two places at once

$$\forall x \forall y (H(x, y) \rightarrow (\forall z (y \neq z) \rightarrow (\neg H(x, z))))$$

The machine is never in two states at once

$$\forall x \bigwedge_{q} (S_q(x) \rightarrow \bigwedge_{q' \neq q} (\neg S_{q'}(x)))$$

Each tape cell contains only one symbol

$$\forall x \forall y \bigwedge_{\sigma} (T_{\sigma}(x, y) \rightarrow \bigwedge_{\sigma' \neq \sigma} (\neg T_{\sigma'}(x, y)))$$
The initial contents of the tape are $[\mathbb{A}]_<$. 

\[
\forall x \quad x \leq n \rightarrow T_1(1, x) \land \\
\quad x \leq n^a \rightarrow (T_1(1, x + n + 1) \leftrightarrow R_1(x|_a))
\]

\[
\ldots
\]

where,

\[
x < n^a : \bigwedge_{i \leq (k-a)} x_i = 0
\]

*Note:* This formula does *not* depend on the structure $\mathbb{A}$ in any way.
The tape does not change except under the head

$$\forall x \forall y \forall z (y \neq z \rightarrow (\bigwedge_{\sigma} (H(x, y) \land T_{\sigma}(x, z) \rightarrow T_{\sigma}(x + 1, z))))$$

Each step is according to $\delta$.

$$\forall x \forall y \bigwedge_{\sigma} \bigwedge_{q} (H(x, y) \land S_{q}(x) \land T_{\sigma}(x, y))$$

$$\rightarrow \bigvee_{\Delta} (H(x + 1, y') \land S_{q'}(x + 1) \land T_{\sigma'}(x + 1, y))$$
where $\Delta$ is the set of all triples $(q', \sigma', D)$ such that $((q, \sigma), (q', \sigma', D)) \in \delta$ and

$$y' = \begin{cases} y & \text{if } D = S \\ y - 1 & \text{if } D = L \\ y + 1 & \text{if } D = R \end{cases}$$

Finally, some accepting state is reached

$$\exists x \ S_{\text{acc}}(x)$$
Recall that a language $L$ is in $\text{NP}$ if, and only if,

$$L = \{x \mid \exists y R(x, y)\}$$

where $R$ is *polynomial-time decidable* and *polynomially-balanced*.

Fagin’s theorem tells us that polynomial-time decidability can, in some sense, be replaced by *first-order definability*.
**USO**—*universal second-order logic* consists of those formulas of second-order logic of the form:

\[ \forall X_1 \cdots \forall X_k \phi \]

where \( \phi \) is a first-order formula.

A corollary of Fagin’s theorem is that a class \( C \) of finite structures is definable by a sentence of *universal second-order logic* if, and only if, its complement is decidable by a *nondeterministic machine* running in polynomial time.

\[ \text{USO} = \text{co-NP} \]
Second-Order Alternation Hierarchy

We can define further classes by allowing other second-order quantifier prefixes.  
\[ \Sigma^1_1 = \text{ESO} \]
\[ \Pi^1_1 = \text{USO} \]
\[ \Sigma^1_{n+1} \] is the collection of properties definable by a sentence of the form:  
\[ \exists X_1 \cdots \exists X_k \phi \] where \( \phi \) is a \( \Pi^1_n \) formula.
\[ \Pi^1_{n+1} \] is the collection of properties definable by a sentence of the form:  
\[ \forall X_1 \cdots \forall X_k \phi \] where \( \phi \) is a \( \Sigma^1_n \) formula.

*Note:* every formula of second-order logic is \( \Sigma^1_n \) and \( \Pi^1_n \) for some \( n \).
Polynomial Hierarchy

We have, for each $n$:

$$\Sigma^1_n \cup \Pi^1_n \subseteq \Sigma^1_{n+1} \cap \Pi^1_{n+1}$$

The classes together form the *polynomial hierarchy* or $\text{PH}$.

$\text{NP} \subseteq \text{PH} \subseteq \text{PSPACE}$

$P = \text{NP}$ if, and only if, $P = \text{PH}$