# Introduction to Descriptive Complexity

**George Voutsadakis**[1]

[1]Mathematics and Computer Science
Lake Superior State University

LSSU Math 600

Subsection 1

## Second-Order Logic

# Second-Order Logic

- **Second-order logic** consists of first-order logic plus new relation variables over which we may quantify.

  Example: The formula

$$(\forall A^r)\varphi$$

  means that, for all choices of $r$-ary relation $A$, $\varphi$ holds.

- Let SO be the set of second-order expressible boolean queries.
- Any second-order formula may be transformed into an equivalent formula with all second-order quantifiers in front.
- If all these second-order quantifiers are existential, then we have a **second-order existential formula**.
- Let SO$\exists$ be the set of second-order existential boolean queries.

# Example

- Let $R$, $Y$ and $B$ be unary relation variables.
- To indicate their arity, we place exponents on relation variables where they are quantified.

$$\begin{aligned}\Phi_{\text{3-color}} \equiv (\exists R^1)(\exists Y^1)(\exists B^1)(\forall x)[&(R(x) \vee Y(x) \vee B(x)) \\ &\wedge (\forall y)(E(x,y) \rightarrow \neg(R(x) \wedge R(y)) \\ &\wedge \neg(Y(x) \wedge Y(y)) \wedge \neg(B(x) \wedge B(y)))].\end{aligned}$$

- Observe that a graph $G$ satisfies $\Phi_{\text{3-color}}$ iff $G$ is 3-colorable.
- Three colorability of graphs is an NP complete problem (3-COLOR).
- We will see that three colorability remains complete via first-order reductions.
- It follows that every query computable in NP is describable in SO∃.

# Expressivity

- Second-order logic is extremely expressive.
- Because of its expressivity:
  - It is very easy to write second-order specifications of queries.
  - Such specifications are not feasible to execute without further refinement.
- Recall that the first-order queries are those that can be computed on a CRAM in constant time, using polynomially many processors.
- We will see that the second-order queries are those that can be computed in constant parallel time, but using exponentially many processors.

# Example of a SO∃ Query: SAT

- SAT is the set of boolean formulas in conjunctive normal form (CNF) that admit a satisfying assignment.
- Recall that $\varphi$ is encoded as

$$\mathcal{A}_\varphi = \langle A, P, N \rangle,$$

  where
  - The universe $A$ is a set of clauses and variables;
  - The relation $P(c, v)$ means that variable $v$ occurs positively in clause $c$;
  - The relation $N(c, v)$ means that $v$ occurs negatively in $c$.
- The boolean query SAT is expressible in SO∃ as follows:

$$\Phi_{\mathrm{SAT}} \equiv (\exists S)(\forall x)(\exists y)((P(x, y) \wedge S(y)) \vee (N(x, y) \wedge \neg S(y))).$$

- $\Phi_{\mathrm{SAT}}$ asserts that there exists a set $S$ of variables, those to be assigned true, forming a satisfying assignment of the input formula.

# Example of a SO∃ Query: CLIQUE

- Boolean query CLIQUE is the set of pairs $\langle G, k \rangle$ such that graph $G$ has a complete subgraph of size $k$.
- The vocabulary for CLIQUE is $\tau_{gk} = \langle E^2, k \rangle$.
- The SO∃ sentence $\Phi_{\text{CLIQUE}}$ says that there is a numbering of the vertices such that those vertices numbered less than $k$ form a clique.
- In order to describe this numbering it is convenient to existentially quantify a function $f$.
- This can be replaced by a binary relation in the usual way.
- Let $\text{Inj}(f)$ mean that $f$ is an injective function,

$$\text{Inj}(f) \equiv (\forall xy)(f(x) = f(y) \rightarrow x = y).$$

- Then, we have

$$\Phi_{\text{CLIQUE}} \equiv (\exists f^1.\text{Inj}(f))(\forall xy)((x \neq y \wedge f(x) < k \wedge f(y) < k) \rightarrow E(x, y)).$$

# Easy Half of Fagin's Theorem

**Proposition**

All second-order existentially definable boolean queries are computable in NP. In symbols, $\text{SO}\exists \subseteq \text{NP}$.

- Consider a second-order existential sentence

$$\Phi \equiv (\exists R_1^{r_1}) \dots (\exists R_k^{r_k})\psi.$$

Let $\tau$ be the vocabulary of $\Phi$.

We build an NP machine $N$, such that, for all $\mathcal{A} \in \text{STRUC}[\tau]$,

$$(\mathcal{A} \vDash \Phi) \;\Leftrightarrow\; (N(\text{bin}(\mathcal{A}))\downarrow).$$

Let $\mathcal{A}$ be an input structure to $N$, with $\|\mathcal{A}\| = n$.

What $N$ does is to nondeterministically write down a binary string of length $n^{r_1}$ representing $R_1$, and, similarly, for $R_2$ through $R_k$.

# Easy Half of Fagin's Theorem (Cont'd)

- By nondeterministically writing down a binary string, we mean that at each step, $N$ nondeterministically chooses to write a 0 or a 1.
- After this polynomial number of steps, we have an expanded structure

$$\mathcal{A}' = (\mathcal{A}, R_1, R_2, \ldots, R_k).$$

$N$ should accept iff $\mathcal{A}' \vDash \psi$.

By a previous theorem, we can test whether $\mathcal{A}' \vDash \psi$ in logspace.

So we can certainly test whether $\mathcal{A}' \vDash \psi$ in NP.

Notice that $N$ accepts $\mathcal{A}$ iff there is some choice of relations $R_1$ through $R_k$ such that

$$(\mathcal{A}, R_1, R_2, \ldots, R_k) \vDash \psi.$$

Thus, the required equivalence holds.

Subsection 2

## Proof of Fagin's Theorem

# Fagin's Theorem

**Theorem (Fagin's Theorem)**

NP equals the set of existential, second-order boolean queries, NP = SO∃. Furthermore, this equality remains true when the first-order part of the second-order formulas is restricted to be universal.

- Let $N$ be a nondetenninistic Turing machine.

  Suppose $N$ uses time $n^k - 1$ for inputs $\text{bin}(\mathcal{A})$ with $\|\mathcal{A}\| = n$.

  We write a second-order sentence

  $$\Phi = (\exists C_1^{2k} \cdots C_g^{2k} \Delta^k)\varphi$$

  that says "there exists an accepting computation $\overline{C}, \Delta$ of $N$".

# Fagin's Theorem (Encoding Configurations)

- More precisely, first-order sentence $\varphi$ will have the property that

$$(\mathcal{A}, \overline{C}, \Delta) \vDash \varphi \quad \text{iff} \quad \overline{C}, \Delta \text{ is an accepting computation of } N \text{ on input } \mathcal{A}.$$

That is,

$$(A \vDash \Phi) \quad \Leftrightarrow \quad (N(\text{bin}(\mathcal{A})) \downarrow).$$

We describe how to code $N$'s computation.

$\overline{C}$ consists of a matrix $\overline{C}(\overline{s}, \overline{t})$ of $n^{2k}$ tape cells with space $\overline{s}$ and time $\overline{t}$ varying between 0 and $n^k - 1$.

We use $k$-tuples of variables $\overline{t} = t_1, \ldots, t_k$ and $\overline{s} = s_1, \ldots, s_k$ each ranging over the universe of $\mathcal{A}$, i.e., from 0 to $n-1$, to code these values.

# Fagin's Theorem (Encoding Configurations)

- For each $\overline{s}, \overline{t}$ pair, $\overline{C}(\overline{s}, \overline{t})$ codes the tape symbol $\sigma$ that appears in cell $\overline{s}$ at time $\overline{t}$, if $N$'s head is not on this cell.

  If the head is present, then $\overline{C}(\overline{s}, \overline{t})$ codes the pair $\langle q, \sigma \rangle$ consisting of $N$'s state $q$ at time $\overline{t}$ and tape symbol $\sigma$.

  Let a listing of the possible contents of a computation cell be

  $$\Gamma = \{\gamma_0, \ldots, \gamma_g\} = (Q \times \Sigma) \cup \Sigma.$$

  We will let $C_i$ be a $2k$-ary relation variable for $0 \leq i \leq g$.

  $C_i(\overline{s}, \overline{t})$ means "computation cell $\overline{s}$ at time $\overline{t}$ contains symbol $\gamma_i$".

# Fagin's Theorem (Encoding Computation)

- At each step, the nondeterministic Turing machine will make one of at most two possible choices.

  We encode these choices in $k$-ary relation $\Delta$.
  - $\Delta(\overline{t})$ is true, if step $\overline{t}+1$ of the computation makes choice "1";
  - $\Delta(\overline{t})$ is false, if step $\overline{t}+1$ of the computation makes choice "0".

  Note that these choices can be determined from $\overline{C}$.

  However, the formula is simplified when we explicitly quantify $\Delta$.

| | **Space** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | $0$ | $1$ | $p$ | $n-1$ | $n$ | | $n^k-1$ | $\Delta$ |
| **Time** $0$ | $\langle q_0, w_0\rangle$ | $w_1$ | $\cdots$ | $w_{n-1}$ | $\sqcup$ | $\cdots$ | $\sqcup$ | $\delta_0$ |
| $1$ | $w_0$ | $\langle q_1, w_1\rangle$ | $\cdots$ | $w_{n-1}$ | $\sqcup$ | $\cdots$ | $\sqcup$ | $\delta_1$ |
| | $\vdots$ | $\vdots$ | $\vdots$ | | | $\vdots$ | | $\vdots$ |
| $t$ | | | $a_{-1}a_0a_1$ | | | | | $\delta_t$ |
| $t+1$ | | | $b$ | | | | | $\delta_{t+1}$ |
| | $\vdots$ | $\vdots$ | $\vdots$ | | | $\vdots$ | | $\vdots$ |
| $n^k-1$ | $\langle q_f, 1\rangle$ | $\cdots$ | $\cdots$ | | | $\cdots$ | | |

# Fagin's Theorem (The First-Order Sentence)

- Now write the first-order sentence $\varphi(\overline{C}, \Delta)$ saying that $\overline{C}, \Delta$ codes a valid accepting computation of $N$.

  The sentence $\varphi$ consists of four parts,

$$\varphi \equiv \alpha \wedge \beta \wedge \eta \wedge \zeta,$$

  where:
  - $\alpha$ asserts that row 0 of the computation correctly codes input $\text{bin}(\mathcal{A})$;
  - $\beta$ says that it is never the case that, for $i \neq j$, $C_i(\overline{s}, \overline{t})$ and $C_j(\overline{s}, \overline{t})$ both hold;
  - $\eta$ says that, for all $\overline{t}$, row $\overline{t} + 1$ of $\overline{C}$ follows from row $\overline{t}$ via move $\Delta(\overline{t})$ of $N$;
  - $\zeta$ says that the last row of the computation includes the accept state.

# Fagin's Theorem (The First-Order Sentence $\zeta$)

- We can write sentence $\zeta$ explicitly.

  We may assume that, when $N$ accepts:
  - It clears its tape;
  - Moves all the way to the left;
  - Enters a unique accept state $q_f$.

  Let $\gamma_{17}$ be the member of $\Gamma$ corresponding to the pair $\langle q_f, 1\rangle$ of state $q_f$, looking at the symbol 1.

  Then we have

$$\zeta = C_{17}(\overline{0}, \overline{\max}).$$

# Fagin's Theorem (The First-Order Sentence $\alpha$)

- Sentence $\alpha$ must assert that the input is of length $I_\tau(n)$ for some $n$ and that $\mathcal{A}$ has been correctly coded as $\text{bin}(\mathcal{A})$.

  Example: Suppose that $\tau$ includes relation symbol $R_1$ of arity one.

  Assume that cell symbols $\gamma_0, \gamma_1$ are '0', '1', respectively.

  Then $\alpha$ includes the following clauses, meaning that:

  - Cell $0 \ldots 0 s_k$ contains 1, if $R_1(s_k)$ holds;
  - Cell $0 \ldots 0 s_k$ contains 0, if $R_1(s_k)$ does not hold.

$$\cdots \wedge (\overline{t} = 0 = s_1 = \cdots = s_{k-1} \wedge s_k \neq 0 \wedge R_1(s_k) \rightarrow C_1(\overline{s}, \overline{t}))$$
$$\wedge(\overline{t} = 0 = s_1 = \cdots = s_{k-1} \wedge s_k \neq 0 \wedge \neg R_1(s_k) \rightarrow C_0(\overline{s}, \overline{t})) \wedge \cdots$$

# Fagin's Theorem (The First-Order Sentence $\eta$)

- The following sentence $\eta$ asserts that the contents of tape cell $(\overline{s}, \overline{t}+1)$ follow from the contents of cells $(\overline{s}-1, \overline{t})$, $(\overline{s}, \overline{t})$, and $(\overline{s}+1, \overline{t})$ via the move $\Delta(\overline{t})$ of $N$.

  Let $\langle a_{-1}, a_0, a_1, \delta\rangle \overset{N}{\to} b$ mean that the triple of cell contents $a_{-1}, a_0, a_1$ lead to cell $b$ via move $\delta$ of $N$.

$$\begin{aligned}\eta_1 \quad \equiv \quad & (\forall \overline{t}.\overline{t} \neq \overline{\max})(\forall \overline{s}.\overline{0} < \overline{s} < \overline{\max}) \\ & \bigwedge_{\langle a_{-1}, a_0, a_1, \delta\rangle \overset{N}{\to} b} (\neg^{\delta}\Delta(\overline{t}) \vee \neg C_{a_{-1}}(\overline{s}-1, \overline{t}) \vee \neg C_{a_0}(\overline{s}, \overline{t}) \\ & \qquad \vee \neg C_{a_1}(\overline{s}+1, \overline{t}) \vee C_b(\overline{s}, \overline{t}+1)).\end{aligned}$$

  Here, $\neg^{\delta}$ is $\neg$, if $\delta = 1$, and is the empty symbol, if $\delta = 0$.

  Finally, let $\eta \equiv \eta_0 \wedge \eta_1 \wedge \eta_2$, where $\eta_0$ and $\eta_2$ encode the same information when $\overline{s} = \overline{0}$ and $\overline{\max}$, respectively.

# Polynomial Time and Existential Second Order

- Observe that the first-order part of formula $\Phi$ in the proof of the proposition is:
  - Universal;
  - In conjunctive normal form.
- Furthermore, if $N$ is a deterministic polynomial-time machine, then we do not need choice relation $\Delta$.
- So the first-order part of $\Phi$ is a Horn formula (a formula in conjunctive normal form with at most one positive literal per clause).
- Accordingly, we obtain the following corollary.

**Corollary**

Every polynomial-time query is expressible as a second-order, existential Horn formula, $\mathrm{P} \subseteq \mathrm{SO}\exists$-Horn.

# Introducing Lynch's Theorem

- The proof of the proposition shows that nondeterministic time $n^k$ is contained in SO∃(arity $2k$).
- Lynch improved this to arity $k$ using the numeric predicate PLUS.
- Fagin's Theorem holds even without numeric predicates, since we can existentially quantify binary relations and assert they are $\leq$ and BIT.
- However, without the numeric predicates, we need an existential first-order quantifier to specify time $\overline{t}+1$, given time $\overline{t}$.

# Lynch's Theorem

**Theorem (Lynch's Theorem)**

For $k \geq 1$,

$$\mathrm{NTIME}[n^k] \subseteq \mathrm{SO}\exists(\text{arity } k).$$

- We need to modify the proof of Fagin's Theorem.

  We only sketch the main ideas involved.

  In Fagin's Theorem, we guessed the entire tape at every step.

  Here, only a bounded number of bits per step is guessed.

  The following relations need to be guessed.

  1. $Q_i(\overline{t})$, meaning that the state at move $\overline{t}$ is $q_i$;
  2. $S_i(\overline{t})$, meaning that the symbol written at move $\overline{t}$ is $\sigma_i$;
  3. $D(\overline{t})$, meaning that the head moves one space to the right after move $\overline{t}$. Otherwise, it moves one space to the left.

# Lynch's Theorem (Cont'd)

- We must write a first-order formula asserting that $\overline{Q}$, $\overline{S}$, $D$ encode a correct accepting computation of $N$.

  The only difficulty in doing this is that, for each move $\overline{t}$, we must ascertain the symbol $\rho_{\overline{t}}$ that is read by $N$.

  $\rho_{\overline{t}}$ is equal to $\sigma_i$, where $S_i(\overline{t}')$ holds and $\overline{t}'$ is the last time before $\overline{t}$ that the head was in its present location (or it is the corresponding input symbol if this is the first time the head is at this cell).

# Lynch's Theorem (Cont'd)

- To express $\rho_{\overline{t}}$, we need to express the function

$$\overline{s} = p(\overline{t}),$$

meaning that at time $\overline{t}$, the head is at position $\overline{s}$.
However, we are restricted to relations of arity $k$.
So we cannot guess the $k \log n$ bits per time needed to specify $p$.
The solution rests on doing the next best thing.
We existentially quantify the current head position once every $\log n$ steps.
We do this by quantifying $k$ bits per step in relations

$$P_i(\overline{t}), \quad i = 1, 2, \ldots, k.$$

Suppose we string $\log n$ of these together, from time $r \log n$ through time $(r + 1) \log n - 1$.
Then we obtain a total of $k \log n$ bits which encode the head position at time $r \log n$.

# Lynch's Theorem (Cont'd)

- The idea is similar to the proof of Bit Sum Lemma.

  Numeric predicate BIT allows us to use each first-order variable to store $\log n$ bits.

  Furthermore, predicate $\mathrm{BSUM}(x, y)$, meaning that the number of one's in the binary expansion of $x$ is $y$, is first-order.

  This enables us to assert that relations $\overline{P}$ are consistent with the head movements given by $D$.

  So we can correctly code the head position at $\log n$ step intervals.

  Finally, using BSUM again, we can ascertain the head position at any time $\overline{t}$.

Subsection 3

## NP-Complete Problems

# NP-Completeness of SAT

- In 1971, Cook proved that SAT (satisfiable boolean formulas) is NP-complete via polynomial time Turing reductions.
- In fact, SAT is NP-complete via significantly weaker reductions.

**Theorem**

SAT is complete for NP via first-order reductions.

- This follows from Fagin's theorem.
  Let $B \in \mathrm{NP}$ be a boolean query.
  We know that $B = \mathrm{MOD}[\Phi]$, where
  $$\Phi = (\exists S_1^{a_1} \cdots S_g^{a_g} \Delta^k)(\forall x_1 \cdots x_t)\psi(\overline{x}),$$
  with $\psi$ quantifier-free.
  We may assume that $\psi$ is in conjunctive normal form,
  $$\psi(\overline{x}) = \bigwedge_{j=1}^{r} C_j(\overline{x}).$$

# NP-Completeness of SAT (Cont'd)

- Let $\mathcal{A}$ be an input structure, with $n = \|\mathcal{A}\|$.
  Define the boolean formula $\gamma(\mathcal{A})$ as follows.
  $\gamma(\mathcal{A})$ has boolean variables
  $$S_i(e_1, \ldots, e_{a_i}) \quad \text{and} \quad D(e_1, \ldots, e_k),$$
  with $i = 1, \ldots, g$, $e_1, \ldots, e_{a_i} \in |\mathcal{A}|$.
  The clauses of $\gamma(\mathcal{A})$ are
  $$C_j(\overline{e}), \quad j = 1, \ldots, r,$$
  as $\overline{e}$ ranges over all $t$-tuples from $|\mathcal{A}|$.

# NP-Completeness of SAT (Cont'd)

- In each $C_j(\overline{e})$, there may be some occurrences of numeric or input predicates, $\gamma(\overline{e})$.

  These should be replaced by true or false, according to whether they are true or false in $\mathcal{A}$.

  It is clear from the construction that

  $$\begin{aligned} \mathcal{A} \in B \quad & \text{iff} \quad \mathcal{A} \vDash \Phi \\ & \text{iff} \quad \gamma(\mathcal{A}) \in \text{SAT}. \end{aligned}$$

  The mapping from $\mathcal{A}$ to $\gamma(\mathcal{A})$ is a $(t+1)$-ary first-order query.

# NP-Completeness of 3-SAT

- We know that SAT is NP-complete via first-order reductions.
- Suppose an SO∃ boolean query is given.
- Then, we can reduce SAT to the given query iff the query is also NP-complete via first-order reductions.

**Proposition**

Let 3-SAT be the subset of SAT in which each clause has at most three literals. Then 3-SAT is NP-complete via first-order reductions.

- We show that SAT $\leq_{\text{fo}}$ 3-SAT.

  First, we give an example of the idea behind the reduction.

# NP-Completeness of 3-SAT (Cont'd)

- Let
$$C = (\ell_1 \vee \ell_2 \vee \cdots \vee \ell_7)$$
be a clause with more than three literals.
Introduce fresh variables $d_1, \ldots, d_4$.
Form the clause
$$\begin{array}{rcl} C' & \equiv & (\ell_1 \vee \ell_2 \vee d_1) \wedge (\overline{d_1} \vee \ell_3 \vee d_2) \wedge (\overline{d_2} \vee \ell_4 \vee d_3) \wedge \\ & & (\overline{d_3} \vee \ell_5 \vee d_4) \wedge (\overline{d_4} \vee \ell_6 \vee \ell_7). \end{array}$$
Observe that $C \in \text{SAT}$ iff $C' \in \text{3-SAT}$.

# NP-Completeness of 3-SAT (Cont'd)

- The first-order reduction from SAT to 3-SAT proceeds as follows.
  Let $\mathcal{A} \in \text{STRUC}[\langle P^2, N^2 \rangle]$ be an instance of SAT with $n = \|\mathcal{A}\|$.
  Each clause $c$ of $\mathcal{A}$ is replaced by $2n$ clauses,

$$\begin{aligned} c' \quad \equiv \quad & ([x_1]^c \vee d_1) \wedge (\overline{d_1} \vee [x_2]^c \vee d_2) \wedge (\overline{d_2} \vee [x_3]^c \vee d_3) \wedge \cdots \wedge \\ & (\overline{d_n} \vee [\overline{x_1}]^c \vee d_{n+1}) \wedge (\overline{d_{n+1}} \vee [\overline{x_2}]^c \vee d_{n+2}) \wedge \cdots \wedge (\overline{d_{2n-1}} \vee [\overline{x_n}]^c). \end{aligned}$$

  Here

$$[\ell]^c = \begin{cases} \ell, & \text{if } \ell \text{ occurs in } c, \\ \textbf{false}, & \text{otherwise.} \end{cases}$$

  We can show that $c'$ is satisfiable iff $c$ is satisfiable.
  Moreover, $c'$ is definable in a first-order way from $c$.

# NP-Completeness of 3-COLOR

## Proposition

3-COLOR is NP-complete via first-order reductions.

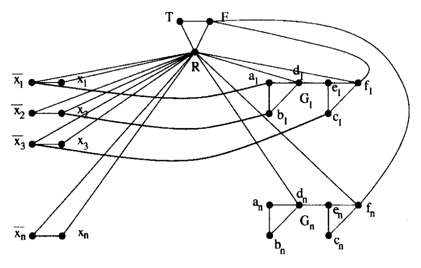
- We will show that 3-SAT $\leq_{\text{fo}}$ 3-COLOR.

  We are given an instance $\mathcal{A}$ of 3-SAT.

  We must produce a graph $f(\mathcal{A})$ that is three colorable iff $\mathcal{A} \in$ 3-SAT.

  Let $n = \|\mathcal{A}\|$, so $\mathcal{A}$ is a boolean formula with at most $n$ variables and $n$ clauses.

  In the triangle, with vertices labeled $T, F, R$, any three-coloring of the graph must color these three vertices distinct colors.

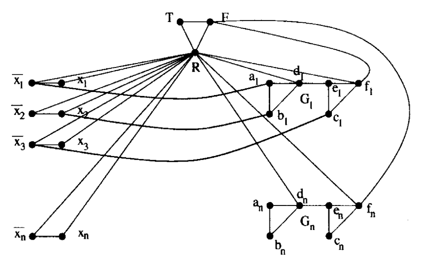



  We may assume without loss of generality that the colors used to color $T$, $F$ and $R$ are true, false and red, respectively.

# NP-Completeness of 3-COLOR (Cont'd)

- Graph $f(\mathcal{A})$ also contains a ladder each rung of which is a variable $x_i$ and its negation $\overline{x}_i$.

  Each of these is connected to $R$, meaning that any valid three-coloring colors one of $x_i$, $\overline{x}_i$ true and the other false.




  Finally, for each clause $C_i = \ell_1 \vee \ell_2 \vee \ell_3$, $f(\mathcal{A})$ contains the gadget $G_i$ consisting of six vertices.

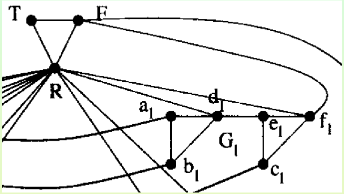
  $G_i$ has:
  - Three inputs $a_i, b_i, c_i$, connected to literals $\ell_1$, $\ell_2$, $\ell_3$, respectively;
  - One output, $f_i$.

  In the figure the gadget $G_1$ corresponds to clause $C_1 = \overline{x}_1 \vee x_2 \vee \overline{x}_3$.

# NP-Completeness of 3-COLOR (Cont'd)

- The triangle $a_1, b_1, d_1$ serves as an "or"-gate in that $d_1$ may be colored true iff at least one of its inputs $\overline{x}_1, x_2$ is colored true.

  Similarly, output $f_1$ may be colored true iff at least one of $d_1$ and the third input, $\overline{x}_3$ is colored true.




  Since $f_i$ is connected to both $F$ and $R$, $f_i$ can only be colored true.

  It follows that a three coloring of the literals can be extended to color $G_i$ iff the corresponding truth assignment makes $C_i$ true.

  Thus, $f(\mathcal{A}) \in$ 3-COLOR iff $\mathcal{A} \in$ 3-SAT.

# NP-Completeness of 3-COLOR (Cont'd)

- The details of first-order reduction $f$ are easy to fill in.

  $f(\mathcal{A})$ consists of:
  - One triangle;
  - A ladder with $n$ rungs;
  - $n$ copies of the gadget.

  The only dependency on the input $\mathcal{A}$, as opposed to its size, is that there is an edge from literal $\ell$ to input $j$ of gadget $G_i$ iff $\ell$ is the $j$-th literal occurring in $C_i$.

Subsection 4

# The Polynomial Time Hierarchy

# The Polynomial Time Hierarchy Revisited

- We defined the polynomial-time hierarchy (PH) to be the set of boolean queries accepted in polynomial time by alternating Turing machines making a bounded number of alternations between existential and universal states.
- The original definition of the polynomial-time hierarchy was via nondeterministic polynomial-time Turing reductions.

# The Polynomial Time Hierarchy via Oracles

## Definition (Polynomial-Time Hierarchy via Oracles)

Let $\Sigma_0^p = \mathrm{P}$ be level 0 of the polynomial-time hierarchy. Inductively, let

$$\Sigma_{i+1}^p = \{L(M^A) : M \text{ is an NP oracle TM}, \ A \in \Sigma_i^p\}.$$

Equivalently, $\Sigma_{i+1}^p$ is the set of boolean queries that are nondeterministic polynomial-time Turing reducible to a set from $\Sigma_i^p$,

$$\Sigma_{i+1}^p = \{B : B \leq_{np}^t A, \text{ for some } A \in \Sigma_i^p\}.$$

Define $\Pi_i^p$ to be co-$\Sigma_i^p$,

$$\Pi_i^p = \{\overline{A} : A \in \Sigma_i^p\}.$$

Finally, define

$$\mathrm{PH} = \bigcup_{k=1}^{\infty} \Sigma_k^p.$$

# Second Order Queries and the Polynomial Hierarchy

## Theorem

Let $S \subseteq \text{STRUC}[\tau]$ be a boolean query, and let $k \geq 1$.
The following are equivalent:

1. $S = \text{MOD}[\Phi]$, for some $\Phi \in \Sigma_k^{\text{SO}}$, where $\Sigma_k^{\text{SO}}$ is the set of all second order sentences with second order quantifier prefix
$$(\exists \overline{R}_1)(\forall \overline{R}_2)\cdots(Q_k \overline{R}_d);$$
2. $S = \{x : (\exists y_1.|y_1| \leq |x|^c)(\forall y_2.|y_2| \leq |x|^c)\cdots(Q_k y_k.|y_k| \leq |x|^c)R(x,\overline{y})\}$, where $R$ is a deterministic polynomial-time predicate on $k+1$ tuples of binary strings and $c$ is a constant;
3. $S \in \text{ATIME-ALT}[n^{O(1)}, k]$;
4. $S \in \Sigma_k^p$.

# Proof of the Theorem

- By induction on $k$.

  The subtle part is relating $\Sigma_k^p$ to the other conditions.

  For this, note that an NP machine with an oracle $A \in \Sigma_{k-1}^p$ can guess all the answers to its oracle queries.

  Then, at the end of its computation, it can check that these answers were all correct.

  This involves a polynomial number of $\Sigma_{k-1}^p$ and $\Pi_{k-1}^p$ questions.

**Corollary**

A boolean query is in the polynomial-time hierarchy iff it is second-order expressible, PH = SO.

# P, NP and Inductive Definitions

- We have shown that P = FO(LFP).
- Thus, by the preceding corollary, we obtain the following descriptive characterization of the $P \stackrel{?}{=} NP$ question.

  P is equal to NP iff every second-order query - over finite, ordered structures - is expressible as a first-order inductive definition.

## Corollary

The following conditions are equivalent:

1. P = NP;
2. Over finite, ordered structures, FO(LFP) = SO.

- Suppose, first, that FO(LFP) = SO.

  Then $P \subseteq NP \subseteq PH = P$.

  Conversely, suppose P = NP. Then PH = NP.

  So FO(LFP) = SO.

# PH and Parallelism

- Up to this point, we had been assuming for notational simplicity that a CRAM has at most polynomially many processors.
- However, the class CRAM-PROC$[t(n), p(n)]$ still makes sense for numbers of processors $p(n)$ that are not polynomially bounded.

**Corollary**

PH is equal to the set of boolean queries recognizable by a CRAM using exponentially many processors and constant time,

$$\text{PH} = \bigcup_{k=1}^{\infty} \text{CRAM-PROC}[1, 2^{n^k}].$$

- The inclusion $\text{SO} \subseteq \text{CRAM-PROC}[1, 2^{n^{O(1)}}]$ follows along the lines of the proof of $\text{FO}[t(n)] \subseteq \text{CRAM}[t(n)]$, presented previously.

# PH and Parallelism (Cont'd)

- A processor number is now large enough to give values to all the relational variables as well as to all the first-order variables.

  Thus, as in that proof, the CRAM can evaluate each first or second-order quantifier in three steps.

  The inclusion CRAM-PROC$[1, 2^{n^{O(1)}}] \subseteq$ SO follows along the lines of the proof of CRAM$[t(n)] \subseteq$ IND$[t(n)]$, also presented previously.

  The only difference is that we use second order variables to specify the processor number.

# SO and Parallelism

- The preceding corollary can be extended to

**Corollary**

For all constructible $t(n)$,

$$\mathrm{SO}[t(n)] = \mathrm{CRAM\text{-}PROC}[t(n), 2^{n^{O(1)}}].$$

- Observe that the previous corollary suggests that PH is a rather strange complexity class.
- No one would ever buy exponentially many processors and then use them only for constant time.
- In contrast, as we will see, the much more robust complexity class PSPACE is encapsulated by exponentially many processors running in polynomial time.