Savitch's Theorem: For $s(n) \geq \log n$,

$$\textbf{NSPACE}[s(n)] \quad \subseteq \quad \textbf{DSPACE}[(s(n))^2]$$

Immerman-Szelepcsényi Theorem: For $s(n) \geq \log n$,

$$\textbf{NSPACE}[s(n)] \quad = \quad \text{co-}\textbf{NSPACE}[s(n)]$$

Closure Theorem: Virtually all the classes we've considered are closed downward under logspace reductions.

Important Technical Fact: Logspace reductions are transitive, i.e., if $A \leq B$ and $B \leq C$ then $A \leq C$.

Consider the input (the object we are working on) to be a finite logical structure, e.g., a binary string, a graph, a relational database, or whatever. Remember that a structure includes a list of the objects and lookup tables for all the variables, constants, relations and functions.

Definition 19.1 FO is the set of first-order definable decision problems on finite structures. Let $S \subseteq \mathrm{STRUC}_{\mathrm{fin}}[\Sigma]$.

$S \in \mathrm{FO}$ iff

$$S = \{\mathcal{A} \in \mathrm{STRUC}_{\mathrm{fin}}[\Sigma] \mid \mathcal{A} \models \varphi\}, \quad \text{some } \varphi \in \mathcal{L}(\Sigma) \spadesuit$$

Defining Addition in First-Order Logic:

Addition is a function from pairs of binary numbers to binary numbers, that is, a map from structures of one vocabulary to structures of another:

$$Q_+ : \text{STRUC}[\Sigma_{AB}] \;\to\; \text{STRUC}[\Sigma_s]$$

$$\begin{array}{llccccc}
A & & a_1 & a_2 & \ldots & a_{n-1} & a_n \\
B & + & b_1 & b_2 & \ldots & b_{n-1} & b_n \\
\hline
S & & s_1 & s_2 & \ldots & s_{n-1} & s_n
\end{array}$$

$$\begin{aligned}
C(i) \;\equiv\; & (\exists j > i)(A(j) \wedge B(j) \;\wedge \\
& \qquad (\forall k . j > k > i)(A(k) \vee B(k)))
\end{aligned}$$

$$Q_+(i) \;\equiv\; A(i) \;\oplus\; B(i) \;\oplus\; C(i)$$

$$Q_+(c) \;\in\; \text{FO}$$

Each bit of the sum is definable by a first-order formula.

Structures and Strings:

Our computational models act on various different data structures. Turing machines take strings, usually binary strings, as input. Propositional formulas take unstructured sets of bits. First-order formulas take first-order structures.

As long as type-casting among these different representations is "easy" in the context of the complexity problem we are considering, we can deal with any of these representations. If pressed, we can say that we are always dealing with strings.

With first-order structures, we pick a natural way to encode each structure $\mathcal{A} \in \mathrm{STRUC}_{\mathrm{fin}}[\Sigma]$ as a binary strings $\mathrm{bin}(\mathcal{A})$.

Example:

- binary strings: $\mathrm{bin}(\mathcal{A}_w) = w$
- graphs: $G = (\{1, \ldots, n\}, E, s, t)$
  $$\mathrm{bin}(G) = a_{11}a_{12} \ldots a_{nn}s_1s_2 \ldots s_{\log n}t_1 \ldots t_{\log n}$$
- pair of numbers: $a_{11} \ldots a_{1n}b_{11} \ldots b_{1n}$

Theorem 19.2 FO $\subseteq$ **L** $=$ **DSPACE**$[\log n]$

Proof:

Given: $\varphi \quad \equiv \quad (\exists x_1)(\forall x_2)\cdots(\forall x_{2k})\psi$,

we build a **DSPACE**$[\log n]$ TM M such that

$$\mathcal{A} \models \varphi \qquad \Leftrightarrow \qquad M(\text{bin}(\mathcal{A})) = 1$$

We use induction on k, the number of quantifier pairs.

Base case: $k = 0$.

$\varphi \equiv E(s,t)$ or another atomic predicate: Look up the right bit.

$\varphi \equiv s \leq t$ or another *numerical* predicate: Do the calculation with the given values.

Inductive step:

$$\varphi' \quad \equiv \quad (\exists x_3)(\forall x_4)\cdots(\forall x_{2k})\psi$$

Note that if other variables like x_1 or x_2 appear in ψ they must be replaced by particular values.

By the inductive hypothesis, there is a logspace TM M',

$$\mathcal{A} \models \varphi' \qquad \Leftrightarrow \qquad M'(\text{bin}(\mathcal{A})) = 1$$

We modify M' by adding $2\lceil \log n \rceil$ worktape cells.

Worktape of M: $\underbrace{\boxed{\quad x_1 \quad}}_{\lceil \log n \rceil}\underbrace{\boxed{\quad x_2 \quad}}_{\lceil \log n \rceil}\boxed{\quad \text{Worktape of } M' \quad}$

M cycles through all values of x_1 until it finds one such that for all x_2, M' accepts. $\spadesuit$

A Java program can easily be written to test whether $\mathcal{A} \models \varphi$. It has nested `for` loops, one for each quantifier. Since it uses only a constant number of variables of $\log n$ bits each, it represents a deterministic logspace algorithm.

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

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

For all choices of the r-ary relation A, φ holds.

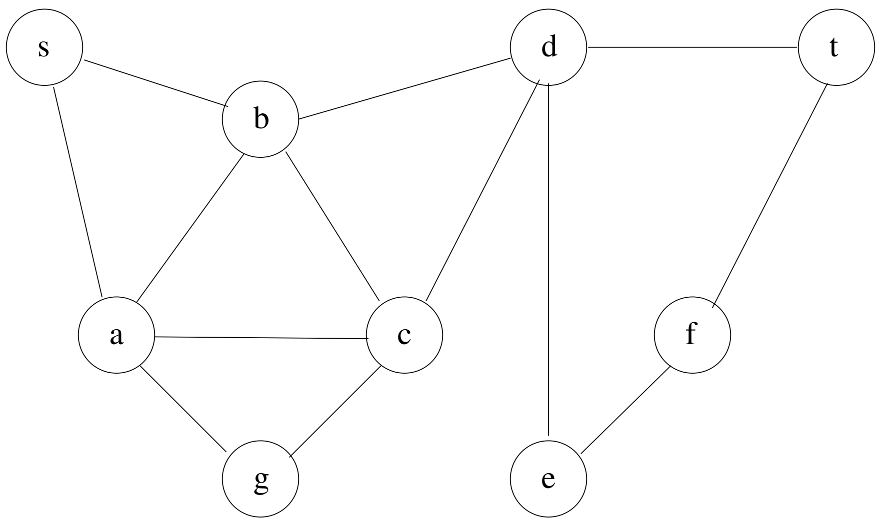
In a finite model with a universe of size n, a first-order variable represents the name of an element of the universe, which is $\lceil \log n \rceil$ bits. A *unary* second-order variable $A(x)$ represents a *property* which each element of the domain has or doesn't have, which is n bits. And a k-ary relation $B(x_1, \ldots, x_k)$ requires n^k bits to specify, because it is true or false for each k-tuple of domain elements.

SO is the set of second-order expressible boolean queries.

SO$\exists$ is the set of second-order *existential* boolean queries, where all the second-order quantifiers are existential.

Graph 3-Colorability is in SO∃:

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

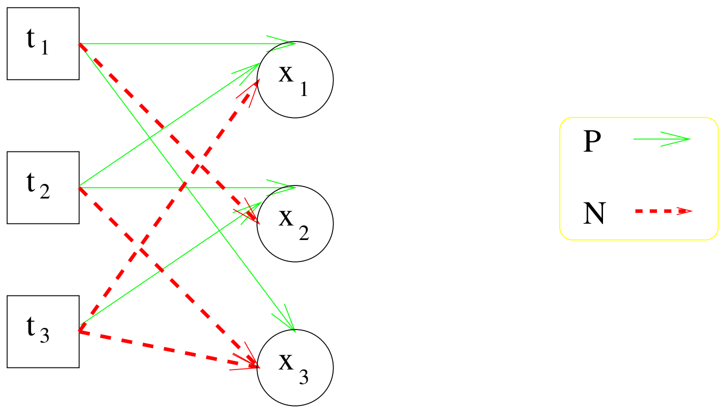


SAT is the set of boolean formulas in conjunctive normal form (CNF) that admit a satisfying assignment.

$$\Phi_{\text{SAT}} \equiv (\exists S^1)(\forall t)(\exists x)(C(t) \rightarrow$$
$$(P(t,x) \wedge S(x)) \vee (N(t,x) \wedge \neg S(x)))$$

$$\begin{aligned} C(t) &\equiv \text{“}t \text{ is a clause; otherwise } t \text{ is a variable.”} \\ P(t,x) &\equiv \text{“Variable } x \text{ occurs positively in clause } t\text{.”} \\ N(t,x) &\equiv \text{“Variable } x \text{ occurs negatively in clause } t\text{.”} \end{aligned}$$

$$\varphi \quad \equiv \quad (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3})$$



CLIQUE is the set of pairs $\langle G, k\rangle$ such that G is a graph that has a complete subgraph of size k.

Let $\mathrm{Inj}(f)$ mean that f is an injective (one-to-one) function. Being an injective function is a *first-order definable* property of a binary relation:

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

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

We could name a particular set of vertices with a unary relation, but the injective function lets us easily say that the set has size exactly k.

Theorem 19.3 (Fagin's Theorem) NP *is equal to the set of existential, second-order boolean queries,* **NP** $=$ SO$\exists$.

Proof: NP $\supseteq$ SO$\exists$: We are given a second-order existential sentence

$$\Phi \equiv (\exists R_1^{r_1}) \ldots (\exists R_k^{r_k})\psi \ \in \ \mathcal{L}(\Sigma)$$

We build an NP machine N such that for every $\mathcal{A} \in$ STRUC$_{\text{fin}}[\Sigma]$,

$$\mathcal{A} \models \Phi \quad \Leftrightarrow \quad N(\text{bin}(\mathcal{A})) = 1 \tag{19.4}$$

$\mathcal{A} \in$ STRUC$_{\text{fin}}[\Sigma]$, $\qquad n = \|\mathcal{A}\|$.

N nondeterministically writes down a binary string of length n^{r_1} representing R_1, and similarly for R_2 through R_k.

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

N accepts iff $\mathcal{A}' \models \psi$.

Since FO $\subseteq$ **L**, as we showed earlier this lecture, we can test whether $\mathcal{A}' \models \psi$ in logspace and so certainly in NP. Thus Equivalence 19.4 holds.

NP $\subseteq$ SO$\exists$: Let N be an **NTIME**$[n^k]$ one-tape TM.

We will define an SO$\exists$ sentence,

$$\Phi \quad = \quad (\exists C_0^{2k} \ldots C_{g-1}^{2k} \Delta^k)\varphi \qquad (19.5)$$

meaning, "There exists an accepting computation $\overline{C}, \Delta$ of N." The main work will be defining the first-order part φ.

We will show that:

$$\mathcal{A} \models \Phi \quad \Leftrightarrow \quad N(\text{bin}(\mathcal{A})) = 1$$

Remark 19.6 *Assume that language has numeric relations:* $\leq$, SUC *and constants* 0, *max refering to total ordering on the universe, its successor relation, the minimum and maximum elements in this ordering, respectively.*

Then φ *in Equation 19.5 can be made universal,*

$$\varphi \quad \equiv \quad (\forall x_1 \cdots x_t)\psi,$$

with ψ *quantifier free.*

$$\text{Fix } \mathcal{A}, \quad n = \|\mathcal{A}\|$$

Possible contents of a computation cell for N:

$$\Gamma = \{\gamma_0, \ldots, \gamma_{g-1}\} = (Q \times \Sigma) \cup \Sigma$$

$C_i(s_1, \ldots, s_k, t_1, \ldots, t_k)$ means that cell $\bar{s}$ at time $\bar{t}$ is symbol γ_i

$\Delta(\bar{t})$ means that the $\bar{t} + 1^{\text{st}}$ step of the computation makes nondeterministic choice "1"; otherwise it makes choice "0". (We have normalized N so that it chooses one bit per step.)

	Space							
	0	1	$\bar{s}$	$n-1$	n		n^k-1	Δ
Time 0	$\langle q_0, w_0 \rangle$	w_1	$\cdots$	w_{n-1}	$\sqcup$	$\cdots$	$\sqcup$	δ_0
1	w_0	$\langle q_1, w_1 \rangle$	$\cdots$	w_{n-1}	$\sqcup$	$\cdots$	$\sqcup$	δ_1
	$\vdots$	$\vdots$	$\vdots$			$\vdots$		$\vdots$
$\bar{t}$			$\boxed{a_{-1}}\boxed{a_0}\boxed{a_1}$					δ_t
$\bar{t}+1$			$\boxed{b}$					δ_{t+1}
	$\vdots$	$\vdots$	$\vdots$			$\vdots$		$\vdots$
n^k-1	$\langle q_f, 1 \rangle$	$\sqcup$	$\cdots$	$\sqcup$	$\sqcup$	$\cdots$	$\sqcup$	

Accepting computation of N on input $w_0 w_1 \cdots w_{n-1}$

Note that we can tell whether the symbol b occurs legally in this computation by looking at the symbols a_{-1}, a_0, and a_1, and consulting the state table of N.

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

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

$\alpha \ \equiv$ row 0 codes input bin$(\mathcal{A})$

$\beta \ \equiv \ (\forall \bar{s}, \bar{t}, i \neq j)(\neg(C_i(\bar{s}, \bar{t}) \wedge C_j(\bar{s}, \bar{t})))$

$\eta \ \equiv \ (\forall \bar{t})$(row $\bar{t}+1$ follows from row $\bar{t}$ via move $\Delta(\bar{t})$ of N)

$\zeta \ \equiv$ last row of computation is accept ID

$$\mathcal{A} \models \Phi \quad \Leftrightarrow \quad N(\mathrm{bin}(\mathcal{A})) = 1$$

$$\Phi \ \equiv \ \exists C_0^{2k} C_1^{2k} \cdots C_{g-1}^{2k} \Delta^k (\varphi)$$

$\equiv$ "$\exists$ an accepting compution: N(me) = 1"

Checking the start configuration:

$$\alpha \quad \equiv \quad \text{row 0 codes input bin}(\mathcal{A})$$

For simplicity, we look at what happens when Σ has only a single unary relation symbol, R, so the input is just a binary string.

$$\begin{array}{|cccccccc|}
0 & 1 & & n-1 & n & & n^k-1 \\
\hline
\langle q_0, w_0\rangle & w_1 & \cdots & w_{n-1} & \sqcup & \cdots & \sqcup
\end{array}$$

$$\gamma_0 = 0;\ \gamma_1 = 1;\ \gamma_2 = \sqcup;\ \gamma_3 = \langle q_0, 0\rangle;\ \gamma_4 = \langle q_0, 1\rangle$$

$$\begin{array}{rrl}
\alpha \equiv & & R(0) \to C_4(\bar{0}, \bar{0}) \\
& \wedge & \neg R(0) \to C_3(\bar{0}, \bar{0}) \\
& \wedge & (\forall i > 0)(R(i) \to C_1(\bar{0}i, \bar{0}) \\
& & \qquad\qquad \wedge \neg R(i) \to C_0(\bar{0}i, \bar{0})) \\
& \wedge & (\forall \bar{s} \geq n) C_2(\bar{s}, \bar{0})
\end{array}$$

The most interesting case: η

We view the state table of N as a finite function, so that

$$\langle a_{-1}, a_0, a_1, \delta \rangle \stackrel{N}{\to} b$$

means that the triple a_{-1}, a_0, a_1 leads to b via move δ of N.

$$\eta_1 \quad \equiv$$

$$(\forall \bar{t}.\bar{t} < \overline{max})(\forall \bar{s}.\bar{0} < \bar{s} < \overline{max}) \bigwedge_{\langle a_{-1},a_0,a_1,\delta\rangle \stackrel{N}{\to} b} (\neg^\delta \Delta(\bar{t}) \vee$$

$$\neg C_{a_{-1}}(\bar{s}{-}1, \bar{t}) \vee \neg C_{a_0}(\bar{s}, \bar{t}) \vee \neg C_{a_1}(\bar{s}{+}1, \bar{t}) \vee C_b(\bar{s}, \bar{t}{+}1))$$

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

$$\eta \quad \equiv \quad \eta_0 \ \wedge \ \eta_1 \ \wedge \ \eta_2$$

where η_0 and η_2 encode the same information when $\bar{s} = \overline{0}$ and $\overline{max}$ respectively. ♠

Theorem 19.7 (Cook-Levin Theorem)

SAT *is* **NP**-*complete.*

(This theorem was proved roughly simultaneously by Steve Cook in the USA and Leonid Levin in the USSR, *before* Fagin proved his theorem. We'll prove Cook-Levin as a corollary of Fagin's Theorem, somewhat contrary to history. But note that the proof of Cook-Levin in Sipser, for example, is almost the same as our proof of Fagin.)

Proof: Let $B \in$ **NP**. By Fagin's theorem,

$$\begin{aligned} B &= \{\mathcal{A} \mid \mathcal{A} \models \Phi\} \\ \Phi &= (\exists C_0^{2k} \cdots C_{g-1}^{2k} \Delta^k)(\forall x_1 \cdots x_t)\psi(\bar{x}) \end{aligned}$$

with ψ quantifier-free and CNF,

$$\psi(\bar{x}) \quad = \quad \bigwedge_{j=1}^{r} T_j(\bar{x})$$

with each T_j a disjunction of literals.

Let $\mathcal{A}$ be arbitrary, with $n = \|\mathcal{A}\|$.

Define a boolean formula $\varphi(\mathcal{A})$ as follows:

boolean variables:

$C_i(e_1, \ldots, e_{2k}), \Delta(e_1, \ldots, e_k), \qquad i = 1, \ldots, g, e_1, \ldots, e_{2k} \in |\mathcal{A}|$

clauses:

$$T_j(\bar{e}), \quad j = 1, \ldots, r, \bar{e} \in |\mathcal{A}|^t$$

$T'_j(\bar{e})$ is $T_j(\bar{e})$ with atomic numeric or input predicates, $R(\bar{e})$, replaced by **true** or **false** according as they are true or false in $\mathcal{A}$. Occurrences of $C_i(\bar{e})$, and $\Delta(\bar{e})$ are considered boolean variables.

$$\begin{aligned} \Phi \; &\equiv \; (\exists C_0^{2k} \cdots C_{g-1}^{2k} \Delta^k)(\forall x_1 \cdots x_t) \bigwedge_{j=1}^{r} T_j(\bar{x}) \\ \varphi(\mathcal{A}) \; &\equiv \; \bigwedge_{e_1, \ldots, e_t \in |\mathcal{A}|} \bigwedge_{j=1}^{r} T'_j(\bar{e}) \end{aligned}$$

$$\mathcal{A} \in B \qquad \Leftrightarrow \qquad \mathcal{A} \models \Phi \qquad \Leftrightarrow \qquad \varphi(\mathcal{A}) \in \mathrm{SAT} \spadesuit$$

Proposition 19.8

$$\text{3-SAT} \quad = \quad \{\varphi \in \text{CNF-SAT} \mid \varphi \textit{ has} \leq 3 \textit{ literals per clause}\}$$

3-SAT *is* **NP**-*complete.*

Proof: Show SAT $\leq$ 3-SAT.

Example:

$$C \quad = \quad (\ell_1 \vee \ell_2 \vee \cdots \vee \ell_7)$$

$$C' \equiv (\ell_1 \vee \ell_2 \vee d_1) \wedge (\overline{d_1} \vee \ell_3 \vee d_2) \wedge (\overline{d_2} \vee \ell_4 \vee d_3) \wedge (\overline{d_3} \vee \ell_5 \vee d_4) \wedge (\overline{d_4} \vee \ell_6 \vee \ell_7)$$

Claim: $\quad C \in \text{SAT} \quad \Leftrightarrow \quad C' \in \text{3-SAT}$

In general, just do this construction for each clause independently, introducing separate dummy variables for each cluase. The AND of all the new 3-variable clauses is satisfiable iff the AND of all the old clauses is. ♠