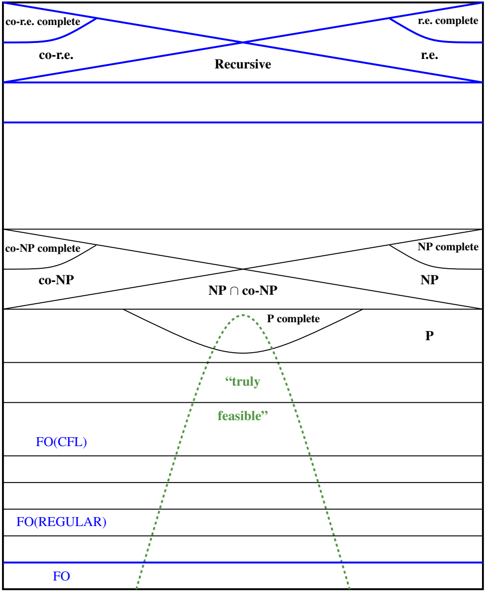
Descriptive Complexity

Neil Immerman

College of Computer and Information Sciences
University of Massachusetts, Amherst
Amherst, MA, USA

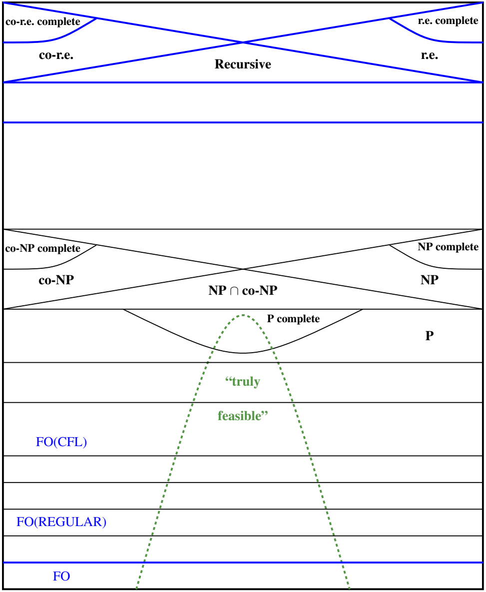
people.cs.umass.edu/~immerman

“truly feasible” is the informal set of problems we can solve exactly on all reasonably sized instances.



$$P = \bigcup_{k=1}^{\infty} \mathrm{DTIME}[n^k]$$

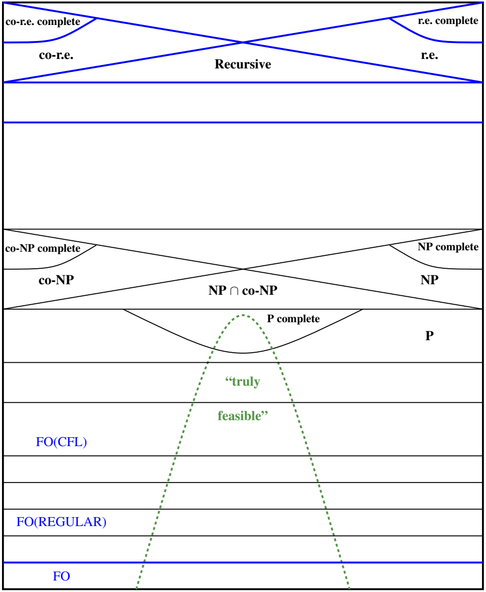
“truly feasible” is the informal set of problems we can solve exactly on all reasonably sized instances.



$$P = \bigcup_{k=1}^{\infty} \mathrm{DTIME}[n^k]$$

P is a good mathematical wrapper for “truly feasible”.

“truly feasible” is the informal set of problems we can solve exactly on all reasonably sized instances.



NTIME[$t(n)$]: a mathematical fiction

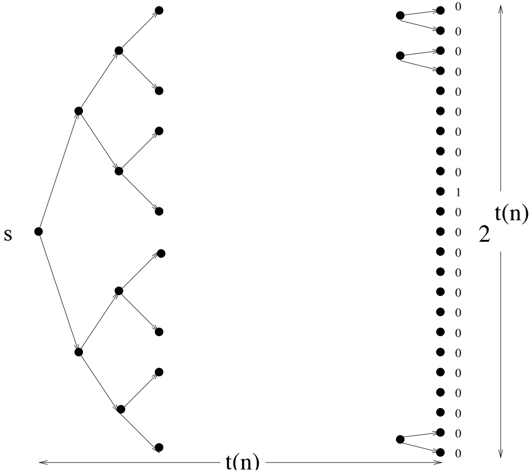
input w, $|w| = n$

N accepts w

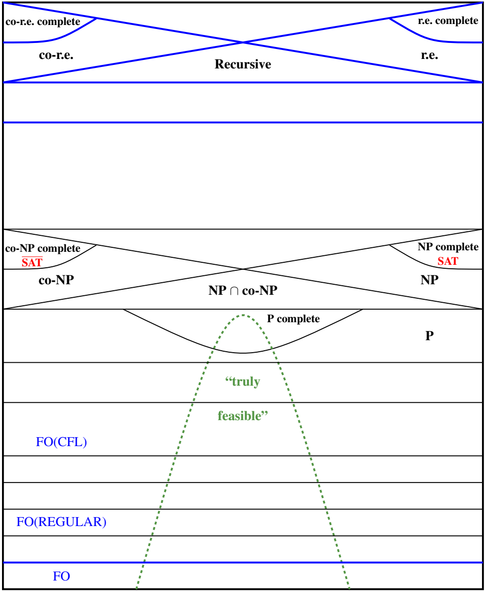
if at least

one of the $2^{t(n)}$

paths accepts.



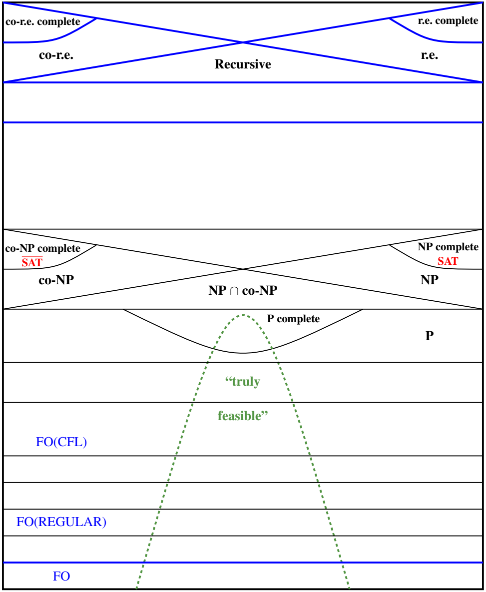
$$\mathrm{NP} = \bigcup_{k=1}^{\infty} \mathrm{NTIME}[n^k]$$



$$\text{NP} = \bigcup_{k=1}^{\infty} \text{NTIME}[n^k]$$

Many optimization problems we want to solve are NP complete.

SAT, TSP, 3-COLOR, CLIQUE, ...

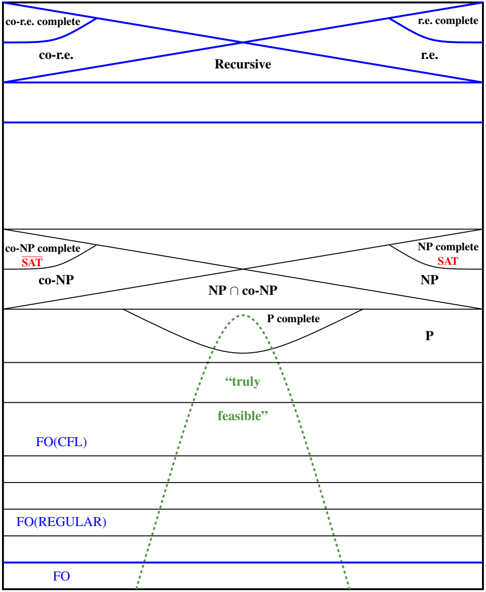


$$\text{NP} = \bigcup_{k=1}^{\infty} \text{NTIME}[n^k]$$

Many optimization problems we want to solve are NP complete.

SAT, TSP, 3-COLOR, CLIQUE, ...

As descison problems, all NP complete problems are isomorphic.

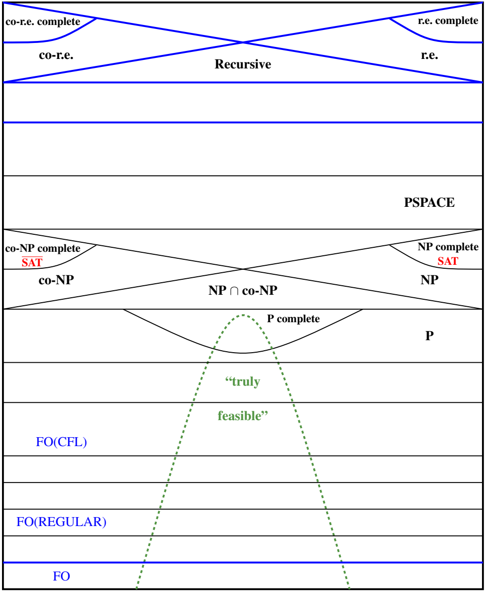


$$\text{NP} = \bigcup_{k=1}^{\infty} \text{NTIME}[n^k]$$

Many optimization problems we want to solve are NP complete.

SAT, TSP, 3-COLOR, CLIQUE, . . .

As descison problems, all NP complete problems are isomorphic.

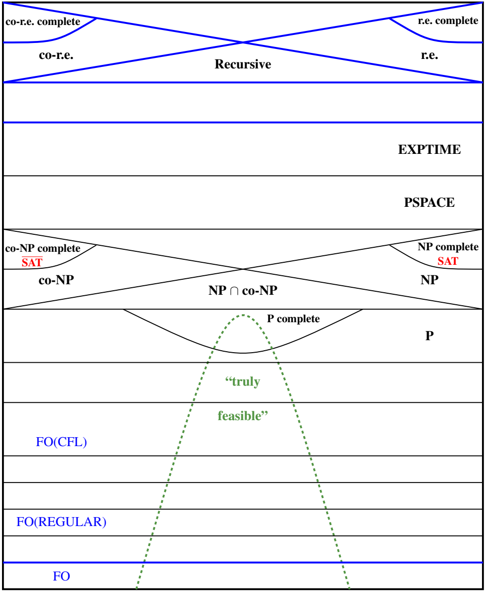


$$\text{NP} = \bigcup_{k=1}^{\infty} \text{NTIME}[n^k]$$

Many optimization problems we want to solve are NP complete.

SAT, TSP, 3-COLOR, CLIQUE, . . .

As descison problems, all NP complete problems are isomorphic.



Query $\mapsto$ **Computation** $\mapsto$ **Answer**

$q_1\ q_2\ \cdots\ q_n$ $\mapsto$ Computation $\mapsto$ $a_1\ a_2\ \cdots\ a_i\ \cdots\ a_m$

Query $\mapsto$ **Computation** $\mapsto$ **Answer**

$q_1\ q_2\ \cdots\ q_n \quad \mapsto \quad \boxed{\text{Computation}} \quad \mapsto \quad a_1\ a_2\ \cdots\ a_i\ \cdots\ a_m$

Restrict attention to the complexity of computing individual bits of the output, i.e., **decision problems**.

Query				**Answer**
$q_1\ q_2\ \cdots\ q_n$	$\mapsto$	**Computation**	$\mapsto$	$a_1\ a_2\ \cdots\ a_i\ \cdots\ a_m$
				$\cdots\ S\ \cdots$

Restrict attention to the complexity of computing individual bits of the output, i.e., **decision problems.**

How hard is it to **check** if input has property S ?

Descriptive Complexity

Query				**Answer**
$q_1 \; q_2 \; \cdots \; q_n$	$\mapsto$	**Computation**	$\mapsto$	$a_1 \; a_2 \; \cdots \; a_i \; \cdots \; a_m$
				$\cdots \; S \; \cdots$

Restrict attention to the complexity of computing individual bits of the output, i.e., **decision problems**.

How hard is it to **check** if input has property S ?

How rich a language do we need to **express** property S?

Descriptive Complexity

Query $q_1\ q_2\ \cdots\ q_n$ $\mapsto$ **Computation** $\mapsto$ **Answer** $a_1\ a_2\ \cdots\ a_i\ \cdots\ a_m$

$\cdots\ S\ \cdots$

Restrict attention to the complexity of computing individual bits of the output, i.e., **decision problems**.

How hard is it to **check** if input has property S ?

How rich a language do we need to **express** property S?

There is a constructive isomorphism between these two approaches.

Think of the Input as a Finite Logical Structure

Graph $\quad G = (\{v_1, \ldots, v_n\}, \leq, E, s, t)$

$\Sigma_g = (E^2, s, t)$

Binary String $\quad \mathcal{A}_w = (\{p_1, \ldots, p_8\}, \leq, S)$

$S = \{p_2, p_5, p_7, p_8\}$

$\Sigma_s = (S^1) \quad w = 01001011$

First-Order Logic

input symbols:	from Σ
variables:	$x, y, z, \ldots$
boolean connectives:	$\wedge, \vee, \neg$
quantifiers:	$\forall, \exists$
numeric symbols:	$=, \leq, +, \times, \min, max$

$$\alpha \equiv \forall x \exists y (E(x,y)) \in \mathcal{L}(\Sigma_g)$$

$$\beta \equiv \exists x \forall y (x \leq y \wedge S(x)) \in \mathcal{L}(\Sigma_s)$$

$$\beta \equiv S(\min) \in \mathcal{L}(\Sigma_s)$$

First-Order Logic

input symbols:	from Σ
variables:	$x, y, z, \ldots$
boolean connectives:	$\wedge, \vee, \neg$
quantifiers:	$\forall, \exists$
numeric symbols:	$=, \leq, +, \times, \text{min}, max$

$$\alpha \equiv \forall x \exists y (E(x,y)) \in \mathcal{L}(\Sigma_g)$$

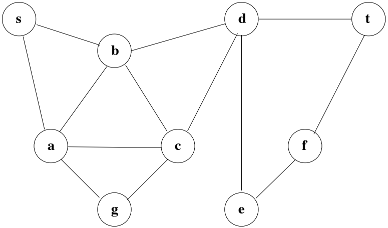
$$\beta \equiv \exists x \forall y (x \leq y \wedge S(x)) \in \mathcal{L}(\Sigma_s)$$

$$\beta \equiv S(\text{min}) \in \mathcal{L}(\Sigma_s)$$

In this setting, with the structure of interest being the **finite input**, FO is a weak, low-level complexity class.

Second-Order Logic: FO plus Relation Variables

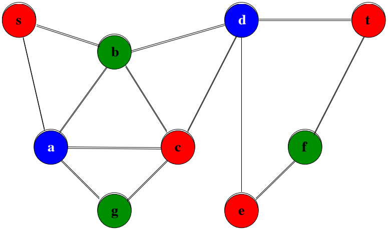
$$\Phi_{\text{3color}} \equiv \exists R^1\, G^1\, B^1\, \forall x\, y\, ((R(x) \vee G(x) \vee B(x)) \wedge (E(x,y) \rightarrow (\neg(R(x) \wedge R(y)) \wedge \neg(G(x) \wedge G(y)) \wedge \neg(B(x) \wedge B(y)))))$$

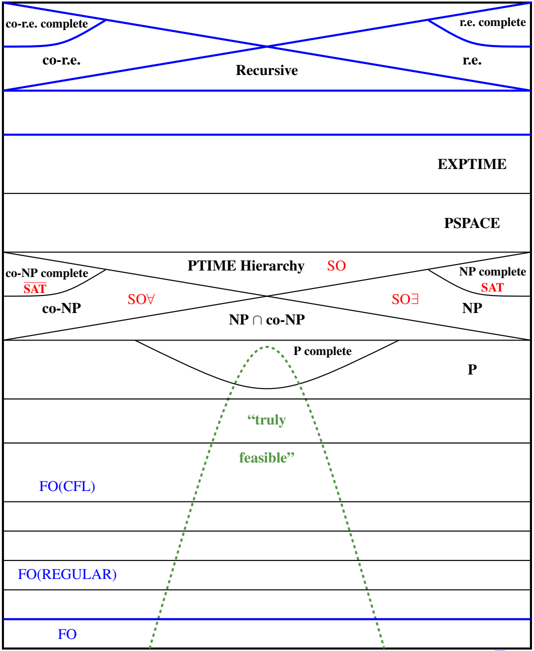


Second-Order Logic: FO plus Relation Variables

Fagin's Theorem: NP $=$ SO$\exists$

$$\Phi_{3\text{color}} \equiv \exists R^1\, G^1\, B^1\, \forall x\, y\, ((R(x) \vee G(x) \vee B(x)) \;\wedge\; (E(x,y) \rightarrow (\neg(R(x) \wedge R(y)) \wedge \neg(G(x) \wedge G(y)) \wedge \neg(B(x) \wedge B(y)))))$$




co-r.e. complete
r.e. complete
co-r.e.
Recursive
r.e.
EXPTIME
PSPACE
PTIME Hierarchy
SO
co-NP complete
$\overline{\text{SAT}}$
NP complete
SAT
SO∀
SO∃
co-NP
NP
NP ∩ co-NP
P complete
P
“truly
feasible”
FO(CFL)
FO(REGULAR)
FO

Addition is First-Order

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

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

Addition is First-Order

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

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

$$C(i) \quad \equiv \quad (\exists j > i)\Big(A(j) \wedge B(j) \ \wedge \\ (\forall k.j > k > i)(A(k) \vee B(k))\Big)$$

Addition is First-Order

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

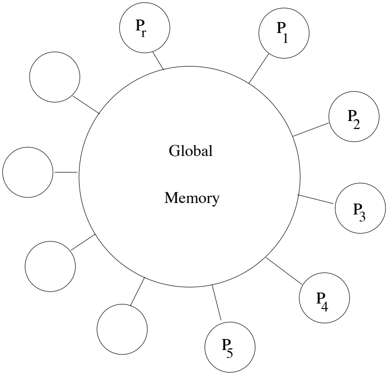
$$\begin{array}{llccccc} A & & a_1 & a_2 & \ldots & a_{n-1} & a_n \\ B & + & b_1 & b_2 & \ldots & b_{n-1} & b_n \\ \hline S & & s_1 & s_2 & \ldots & s_{n-1} & s_n \end{array}$$

$$C(i) \equiv (\exists j > i)\Big(A(j) \land B(j) \land (\forall k.j > k > i)(A(k) \lor B(k))\Big)$$

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

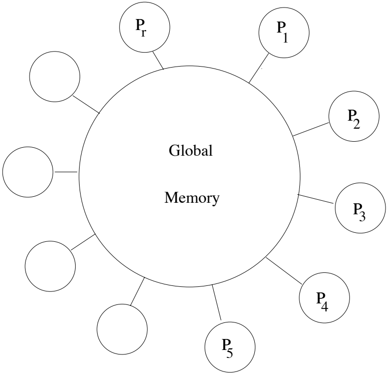
Parallel Machines:

$$\text{CRAM}[t(n)] \;=\; \text{CRCW-PRAM-TIME}[t(n)]\text{-HARD}[n^{O(1)}]$$



CRAM[$t(n)$] $=$ CRCW-PRAM-TIME[$t(n)$]-HARD[$n^{O(1)}$]

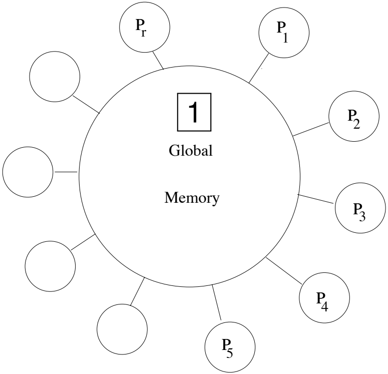
Assume array $A[x] : x = 1, \ldots, r$ in memory.



$\text{CRAM}[t(n)] = \text{CRCW-PRAM-TIME}[t(n)]\text{-HARD}[n^{O(1)}]$

Assume array $A[x] : x = 1, \ldots, r$ in memory.

$\forall x(A(x)) \equiv$ **write**(1);

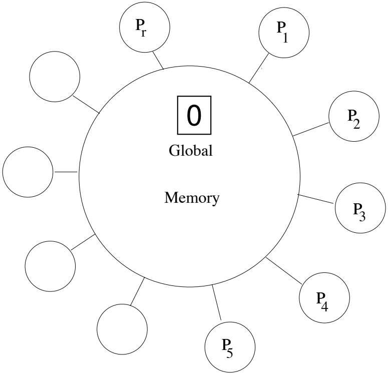


Parallel Machines: Quantifiers are Parallel

$\text{CRAM}[t(n)] = \text{CRCW-PRAM-TIME}[t(n)]\text{-HARD}[n^{O(1)}]$

Assume array $A[x] : x = 1, \ldots, r$ in memory.

$\forall x (A(x)) \equiv$ **write**(1); proc p_i : **if** $(A[i] = 0)$ **then** **write**(0)



FO

=

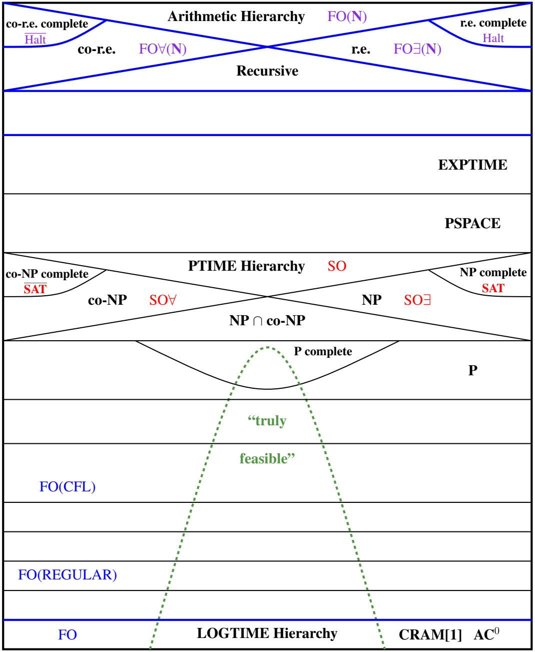
CRAM[1]

=

AC^0

=

Logarithmic-Time Hierarchy

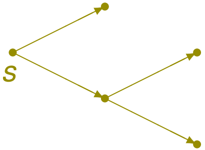


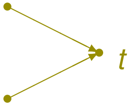
Inductive Definitions and Least Fixed Point

$$\text{REACH} \;=\; \{G, s, t \mid s \stackrel{\star}{\to} t\}$$

Inductive Definitions and Least Fixed Point

$$\text{REACH} = \{G, s, t \mid s \xrightarrow{\star} t\} \qquad \text{REACH} \notin \text{FO}$$

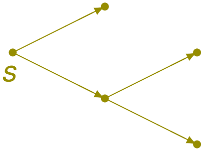


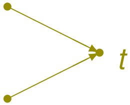


Inductive Definitions and Least Fixed Point

$$E^\star(x,y) \stackrel{\text{def}}{=} x=y \lor E(x,y) \lor \exists z(E^\star(x,z) \land E^\star(z,y))$$

$\text{REACH} = \{G, s, t \mid s \stackrel{\star}{\to} t\}$ $\qquad$ $\text{REACH} \notin \text{FO}$





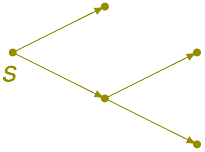
Inductive Definitions and Least Fixed Point

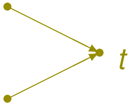
$$E^\star(x,y) \;\stackrel{\text{def}}{=}\; x=y \;\lor\; E(x,y) \;\lor\; \exists z(E^\star(x,z) \land E^\star(z,y))$$

$$\varphi_{tc}(R,x,y) \;\equiv\; x=y \;\lor\; E(x,y) \;\lor\; \exists z(R(x,z) \land R(z,y))$$

$$\text{REACH} \;=\; \left\{G,s,t \;\middle|\; s \stackrel{\star}{\to} t\right\}$$

$$\text{REACH} \notin \mathrm{FO}$$





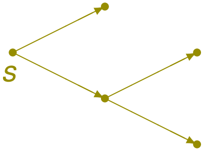
Inductive Definitions and Least Fixed Point

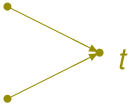
$$E^\star(x,y) \stackrel{\text{def}}{=} x = y \lor E(x,y) \lor \exists z(E^\star(x,z) \land E^\star(z,y))$$

$$\varphi_{tc}(R,x,y) \equiv x = y \lor E(x,y) \lor \exists z(R(x,z) \land R(z,y))$$

$$\varphi_{tc}^G : \text{binRel}(G) \to \text{binRel}(G) \quad \text{is a monotone operator}$$

$$\text{REACH} = \{G,s,t \mid s \stackrel{\star}{\to} t\} \qquad \text{REACH} \notin \text{FO}$$





Inductive Definitions and Least Fixed Point

$$E^\star(x,y) \stackrel{\text{def}}{=} x = y \ \lor\ E(x,y) \ \lor\ \exists z(E^\star(x,z) \land E^\star(z,y))$$

$$\varphi_{tc}(R,x,y) \equiv x = y \ \lor\ E(x,y) \ \lor\ \exists z(R(x,z) \land R(z,y))$$

$\varphi_{tc}^G : \text{binRel}(G) \to \text{binRel}(G)$ is a monotone operator

$$E^\star = (\text{LFP}\varphi_{tc})$$

$$\text{REACH} = \{G, s, t \mid s \stackrel{\star}{\to} t\} \qquad \text{REACH} \notin \text{FO}$$

Inductive Definitions and Least Fixed Point

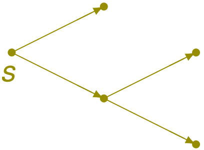
$$E^\star(x,y) \;\stackrel{\text{def}}{=}\; x = y \;\lor\; E(x,y) \;\lor\; \exists z(E^\star(x,z) \land E^\star(z,y))$$

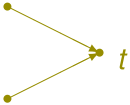
$$\varphi_{tc}(R,x,y) \;\equiv\; x = y \;\lor\; E(x,y) \;\lor\; \exists z(R(x,z) \land R(z,y))$$

$\varphi_{tc}^G : \mathrm{binRel}(G) \;\rightarrow\; \mathrm{binRel}(G)$ is a monotone operator

$$G \in \mathrm{REACH} \;\Leftrightarrow\; G \models (\mathrm{LFP}\varphi_{tc})(s,t) \qquad E^\star \;=\; (\mathrm{LFP}\varphi_{tc})$$

$$\mathrm{REACH} \;=\; \{G, s, t \mid s \stackrel{\star}{\rightarrow} t\} \qquad \mathrm{REACH} \not\in \mathrm{FO}$$





Tarski-Knaster Theorem

Thm. If $\varphi : \mathrm{Rel}^k(G) \to \mathrm{Rel}^k(G)$ is monotone, then $\mathrm{LFP}(\varphi)$ exists and can be computed in P.

Tarski-Knaster Theorem

Thm. If $\varphi : \mathrm{Rel}^k(G) \to \mathrm{Rel}^k(G)$ is monotone, then $\mathrm{LFP}(\varphi)$ exists and can be computed in P.

proof: Monotone means, for all $R \subseteq S$, $\quad \varphi(R) \subseteq \varphi(S)$.

Tarski-Knaster Theorem

Thm. If $\varphi : \mathrm{Rel}^k(G) \to \mathrm{Rel}^k(G)$ is monotone, then $\mathrm{LFP}(\varphi)$ exists and can be computed in P.

proof: Monotone means, for all $R \subseteq S, \quad \varphi(R) \subseteq \varphi(S)$.

Let $I^0 \stackrel{\text{def}}{=} \emptyset; \quad I^{r+1} \stackrel{\text{def}}{=} \varphi(I^r)$

Tarski-Knaster Theorem

Thm. If $\varphi : \mathrm{Rel}^k(G) \to \mathrm{Rel}^k(G)$ is monotone, then $\mathrm{LFP}(\varphi)$ exists and can be computed in P.

proof: Monotone means, for all $R \subseteq S$, $\quad \varphi(R) \subseteq \varphi(S)$.

Let $I^0 \stackrel{\text{def}}{=} \emptyset; \quad I^{r+1} \stackrel{\text{def}}{=} \varphi(I^r)$ $\quad$ Thus, $\emptyset = I^0 \subseteq I^1 \subseteq \cdots \subseteq I^t$.

Tarski-Knaster Theorem

Thm. If $\varphi : \mathrm{Rel}^k(G) \to \mathrm{Rel}^k(G)$ is monotone, then $\mathrm{LFP}(\varphi)$ exists and can be computed in P.

proof: Monotone means, for all $R \subseteq S$, $\quad \varphi(R) \subseteq \varphi(S)$.

Let $I^0 \stackrel{\text{def}}{=} \emptyset; \quad I^{r+1} \stackrel{\text{def}}{=} \varphi(I^r)$ Thus, $\emptyset = I^0 \subseteq I^1 \subseteq \cdots \subseteq I^t$.

Let t be min such that $I^t = I^{t+1}$. Note that $t \leq n^k$ where $n = |V^G|$.

Tarski-Knaster Theorem

Thm. If $\varphi : \mathrm{Rel}^k(G) \to \mathrm{Rel}^k(G)$ is monotone, then $\mathrm{LFP}(\varphi)$ exists and can be computed in P.

proof: Monotone means, for all $R \subseteq S$, $\quad \varphi(R) \subseteq \varphi(S)$.

Let $I^0 \stackrel{\text{def}}{=} \emptyset$; $\quad I^{r+1} \stackrel{\text{def}}{=} \varphi(I^r)$ $\quad$ Thus, $\emptyset = I^0 \subseteq I^1 \subseteq \cdots \subseteq I^t$.

Let t be min such that $I^t = I^{t+1}$. Note that $t \leq n^k$ where $n = |V^G|$. $\quad \varphi(I^t) = I^t$, $\quad$ so I^t is a fixed point of φ.

Tarski-Knaster Theorem

Thm. If $\varphi : \text{Rel}^k(G) \to \text{Rel}^k(G)$ is monotone, then $\text{LFP}(\varphi)$ exists and can be computed in P.

proof: Monotone means, for all $R \subseteq S$, $\quad \varphi(R) \subseteq \varphi(S)$.

Let $I^0 \stackrel{\text{def}}{=} \emptyset$; $\quad I^{r+1} \stackrel{\text{def}}{=} \varphi(I^r) \quad$ Thus, $\emptyset = I^0 \subseteq I^1 \subseteq \cdots \subseteq I^t$.

Let t be min such that $I^t = I^{t+1}$. Note that $t \leq n^k$ where $n = |V^G|$. $\quad \varphi(I^t) = I^t$, $\quad$ so I^t is a fixed point of φ.

Suppose $\varphi(F) = F$.

Tarski-Knaster Theorem

Thm. If $\varphi : \mathrm{Rel}^k(G) \to \mathrm{Rel}^k(G)$ is monotone, then $\mathrm{LFP}(\varphi)$ exists and can be computed in P.

proof: Monotone means, for all $R \subseteq S$, $\quad \varphi(R) \subseteq \varphi(S)$.

Let $I^0 \stackrel{\text{def}}{=} \emptyset$; $\quad I^{r+1} \stackrel{\text{def}}{=} \varphi(I^r)$ $\quad$ Thus, $\emptyset = I^0 \subseteq I^1 \subseteq \cdots \subseteq I^t$.

Let t be min such that $I^t = I^{t+1}$. Note that $t \leq n^k$ where $n = |V^G|$. $\quad \varphi(I^t) = I^t$, $\quad$ so I^t is a fixed point of φ.

Suppose $\varphi(F) = F$. $\quad$ By induction on r, for all r, $I^r \subseteq F$.

Tarski-Knaster Theorem

Thm. If $\varphi : \mathrm{Rel}^k(G) \to \mathrm{Rel}^k(G)$ is monotone, then $\mathrm{LFP}(\varphi)$ exists and can be computed in P.

proof: Monotone means, for all $R \subseteq S$, $\quad \varphi(R) \subseteq \varphi(S)$.

Let $I^0 \stackrel{\text{def}}{=} \emptyset$; $\quad I^{r+1} \stackrel{\text{def}}{=} \varphi(I^r)$ $\quad$ Thus, $\emptyset = I^0 \subseteq I^1 \subseteq \cdots \subseteq I^t$.

Let t be min such that $I^t = I^{t+1}$. Note that $t \leq n^k$ where $n = |V^G|$. $\quad \varphi(I^t) = I^t$, $\quad$ so I^t is a fixed point of φ.

Suppose $\varphi(F) = F$. $\quad$ By induction on r, for all r, $I^r \subseteq F$.

base case: $\quad I^0 = \emptyset \subseteq F$.

Tarski-Knaster Theorem

Thm. If $\varphi : \mathrm{Rel}^k(G) \to \mathrm{Rel}^k(G)$ is monotone, then $\mathrm{LFP}(\varphi)$ exists and can be computed in P.

proof: Monotone means, for all $R \subseteq S$, $\quad \varphi(R) \subseteq \varphi(S)$.

Let $I^0 \stackrel{\text{def}}{=} \emptyset; \quad I^{r+1} \stackrel{\text{def}}{=} \varphi(I^r)$ $\quad$ Thus, $\emptyset = I^0 \subseteq I^1 \subseteq \cdots \subseteq I^t$.

Let t be min such that $I^t = I^{t+1}$. Note that $t \leq n^k$ where $n = |V^G|$. $\quad \varphi(I^t) = I^t$, $\quad$ so I^t is a fixed point of φ.

Suppose $\varphi(F) = F$. $\quad$ By induction on r, for all r, $I^r \subseteq F$.

base case: $\quad I^0 = \emptyset \subseteq F$.

inductive case: $\quad$ Assume $I^j \subseteq F$

Tarski-Knaster Theorem

Thm. If $\varphi : \mathrm{Rel}^k(G) \to \mathrm{Rel}^k(G)$ is monotone, then $\mathrm{LFP}(\varphi)$ exists and can be computed in P.

proof: Monotone means, for all $R \subseteq S$, $\quad \varphi(R) \subseteq \varphi(S)$.

Let $I^0 \stackrel{\text{def}}{=} \emptyset$; $\quad I^{r+1} \stackrel{\text{def}}{=} \varphi(I^r)$ $\quad$ Thus, $\emptyset = I^0 \subseteq I^1 \subseteq \cdots \subseteq I^t$.

Let t be min such that $I^t = I^{t+1}$. Note that $t \leq n^k$ where $n = |V^G|$. $\quad \varphi(I^t) = I^t$, $\quad$ so I^t is a fixed point of φ.

Suppose $\varphi(F) = F$. $\quad$ By induction on r, for all r, $I^r \subseteq F$.

base case: $\quad I^0 = \emptyset \subseteq F$.

inductive case: $\quad$ Assume $I^j \subseteq F$

By monotonicity, $\quad \varphi(I^j) \subseteq \varphi(F)$, i.e., $\quad I^{j+1} \subseteq F$.

Tarski-Knaster Theorem

Thm. If $\varphi : \mathrm{Rel}^k(G) \to \mathrm{Rel}^k(G)$ is monotone, then $\mathrm{LFP}(\varphi)$ exists and can be computed in P.

proof: Monotone means, for all $R \subseteq S$, $\quad \varphi(R) \subseteq \varphi(S)$.

Let $I^0 \stackrel{\text{def}}{=} \emptyset$; $\quad I^{r+1} \stackrel{\text{def}}{=} \varphi(I^r)$ $\quad$ Thus, $\emptyset = I^0 \subseteq I^1 \subseteq \cdots \subseteq I^t$.

Let t be min such that $I^t = I^{t+1}$. Note that $t \leq n^k$ where $n = |V^G|$. $\quad \varphi(I^t) = I^t$, $\quad$ so I^t is a fixed point of φ.

Suppose $\varphi(F) = F$. $\quad$ By induction on r, for all r, $I^r \subseteq F$.

base case: $\quad I^0 = \emptyset \subseteq F$.

inductive case: $\quad$ Assume $I^j \subseteq F$

By monotonicity, $\quad \varphi(I^j) \subseteq \varphi(F)$, i.e., $\quad I^{j+1} \subseteq F$.

Thus $I^t \subseteq F$ $\quad$ and $\quad I^t = \mathrm{LFP}(\varphi)$. $\square$

Inductive Definition of Transitive Closure

$$\varphi_{tc}(R,x,y) \quad \equiv \quad x = y \ \lor \ E(x,y) \ \lor \ \exists z(R(x,z) \land R(z,y))$$

Inductive Definition of Transitive Closure

$$\varphi_{tc}(R, x, y) \equiv x = y \vee E(x, y) \vee \exists z(R(x, z) \wedge R(z, y))$$

$$I^1 = \varphi_{tc}^G(\emptyset) = \left\{(a, b) \in V^G \times V^G \;\middle|\; \mathrm{dist}(a, b) \leq 1\right\}$$

Inductive Definition of Transitive Closure

$$\begin{aligned}
\varphi_{tc}(R, x, y) &\equiv x = y \ \vee\ E(x, y)\ \vee\ \exists z (R(x, z) \wedge R(z, y)) \\
I^1 = \varphi_{tc}^G(\emptyset) &= \left\{(a, b) \in V^G \times V^G \;\middle|\; \mathrm{dist}(a, b) \le 1\right\} \\
I^2 = (\varphi_{tc}^G)^2(\emptyset) &= \left\{(a, b) \in V^G \times V^G \;\middle|\; \mathrm{dist}(a, b) \le 2\right\}
\end{aligned}$$

Inductive Definition of Transitive Closure

$$\varphi_{tc}(R, x, y) \equiv x = y \lor E(x, y) \lor \exists z(R(x, z) \land R(z, y))$$

$$I^1 = \varphi_{tc}^G(\emptyset) = \{(a, b) \in V^G \times V^G \mid \mathrm{dist}(a, b) \leq 1\}$$

$$I^2 = (\varphi_{tc}^G)^2(\emptyset) = \{(a, b) \in V^G \times V^G \mid \mathrm{dist}(a, b) \leq 2\}$$

$$I^3 = (\varphi_{tc}^G)^3(\emptyset) = \{(a, b) \in V^G \times V^G \mid \mathrm{dist}(a, b) \leq 4\}$$

Inductive Definition of Transitive Closure

$$
\begin{array}{rcl}
\varphi_{tc}(R,x,y) & \equiv & x = y \ \vee \ E(x,y) \ \vee \ \exists z(R(x,z) \wedge R(z,y)) \\
I^1 = \varphi_{tc}^G(\emptyset) & = & \{(a,b) \in V^G \times V^G \mid \mathrm{dist}(a,b) \leq 1\} \\
I^2 = (\varphi_{tc}^G)^2(\emptyset) & = & \{(a,b) \in V^G \times V^G \mid \mathrm{dist}(a,b) \leq 2\} \\
I^3 = (\varphi_{tc}^G)^3(\emptyset) & = & \{(a,b) \in V^G \times V^G \mid \mathrm{dist}(a,b) \leq 4\} \\
\vdots & = & \vdots \qquad \vdots \\
I^r = (\varphi_{tc}^G)^r(\emptyset) & = & \{(a,b) \in V^G \times V^G \mid \mathrm{dist}(a,b) \leq 2^{r-1}\} \\
\vdots & = & \vdots \qquad \vdots
\end{array}
$$

Inductive Definition of Transitive Closure

$$\begin{array}{rcl}
\varphi_{tc}(R,x,y) & \equiv & x = y \;\vee\; E(x,y) \;\vee\; \exists z(R(x,z) \wedge R(z,y)) \\
I^1 = \varphi_{tc}^G(\emptyset) & = & \{(a,b) \in V^G \times V^G \mid \mathrm{dist}(a,b) \le 1\} \\
I^2 = (\varphi_{tc}^G)^2(\emptyset) & = & \{(a,b) \in V^G \times V^G \mid \mathrm{dist}(a,b) \le 2\} \\
I^3 = (\varphi_{tc}^G)^3(\emptyset) & = & \{(a,b) \in V^G \times V^G \mid \mathrm{dist}(a,b) \le 4\} \\
\vdots & = & \vdots \qquad \vdots \\
I^r = (\varphi_{tc}^G)^r(\emptyset) & = & \{(a,b) \in V^G \times V^G \mid \mathrm{dist}(a,b) \le 2^{r-1}\} \\
\vdots & = & \vdots \qquad \vdots \\
(\varphi_{tc}^G)^{\lceil 1+\log n \rceil}(\emptyset) & = & \{(a,b) \in V^G \times V^G \mid \mathrm{dist}(a,b) \le n\}
\end{array}$$

Inductive Definition of Transitive Closure

$$
\begin{array}{rcl}
\varphi_{tc}(R, x, y) & \equiv & x = y \ \vee \ E(x, y) \ \vee \ \exists z(R(x, z) \wedge R(z, y)) \\
I^1 = \varphi_{tc}^G(\emptyset) & = & \{(a, b) \in V^G \times V^G \mid \mathrm{dist}(a, b) \le 1\} \\
I^2 = (\varphi_{tc}^G)^2(\emptyset) & = & \{(a, b) \in V^G \times V^G \mid \mathrm{dist}(a, b) \le 2\} \\
I^3 = (\varphi_{tc}^G)^3(\emptyset) & = & \{(a, b) \in V^G \times V^G \mid \mathrm{dist}(a, b) \le 4\} \\
\vdots & = & \vdots \qquad\qquad \vdots \\
I^r = (\varphi_{tc}^G)^r(\emptyset) & = & \{(a, b) \in V^G \times V^G \mid \mathrm{dist}(a, b) \le 2^{r-1}\} \\
\vdots & = & \vdots \qquad\qquad \vdots \\
(\varphi_{tc}^G)^{\lceil 1+\log n \rceil}(\emptyset) & = & \{(a, b) \in V^G \times V^G \mid \mathrm{dist}(a, b) \le n\} \\
\mathrm{LFP}(\varphi_{tc}) & = & \varphi_{tc}^{\lceil 1+\log n \rceil}(\emptyset); \qquad \mathrm{REACH} \in \mathrm{IND}[\log n]
\end{array}
$$

Inductive Definition of Transitive Closure

$$\begin{array}{rcl}
\varphi_{tc}(R,x,y) & \equiv & x = y \ \vee \ E(x,y) \ \vee \ \exists z(R(x,z) \wedge R(z,y)) \\
I^1 = \varphi_{tc}^G(\emptyset) & = & \{(a,b) \in V^G \times V^G \mid \mathrm{dist}(a,b) \le 1\} \\
I^2 = (\varphi_{tc}^G)^2(\emptyset) & = & \{(a,b) \in V^G \times V^G \mid \mathrm{dist}(a,b) \le 2\} \\
I^3 = (\varphi_{tc}^G)^3(\emptyset) & = & \{(a,b) \in V^G \times V^G \mid \mathrm{dist}(a,b) \le 4\} \\
\vdots & = & \vdots \qquad \vdots \\
I^r = (\varphi_{tc}^G)^r(\emptyset) & = & \{(a,b) \in V^G \times V^G \mid \mathrm{dist}(a,b) \le 2^{r-1}\} \\
\vdots & = & \vdots \qquad \vdots \\
(\varphi_{tc}^G)^{\lceil 1+\log n \rceil}(\emptyset) & = & \{(a,b) \in V^G \times V^G \mid \mathrm{dist}(a,b) \le n\} \\
\mathrm{LFP}(\varphi_{tc}) & = & \varphi_{tc}^{\lceil 1+\log n \rceil}(\emptyset); \qquad \mathrm{REACH} \in \mathrm{IND}[\log n]
\end{array}$$

Next we will show that $\mathrm{IND}[t(n)] = \mathrm{FO}[t(n)]$.

$\varphi_{tc}(R, x, y) \equiv x = y \vee E(x, y) \vee \exists z\,(R(x, z) \wedge R(z, y))$

1. Dummy universal quantification for base case:

$$
\begin{aligned}
\varphi_{tc}(R, x, y) &\equiv (\forall z. M_1)(\exists z)(R(x, z) \wedge R(z, y)) \\
M_1 &\equiv \neg(x = y \vee E(x, y))
\end{aligned}
$$

$\varphi_{tc}(R, x, y) \equiv x = y \vee E(x, y) \vee \exists z\,(R(x, z) \wedge R(z, y))$

1. Dummy universal quantification for base case:

$$\begin{aligned} \varphi_{tc}(R, x, y) &\equiv (\forall z.M_1)(\exists z)(R(x, z) \wedge R(z, y)) \\ M_1 &\equiv \neg(x = y \vee E(x, y)) \end{aligned}$$

2. Using $\forall$, replace two occurrences of R with one:

$$\begin{aligned} \varphi_{tc}(R, x, y) &\equiv (\forall z.M_1)(\exists z)(\forall uv.M_2)R(u, v) \\ M_2 &\equiv (u = x \wedge v = z) \ \vee \ (u = z \wedge v = y) \end{aligned}$$

$\varphi_{tc}(R, x, y) \equiv x = y \vee E(x, y) \vee \exists z\,(R(x, z) \wedge R(z, y))$

1. Dummy universal quantification for base case:

$$\begin{aligned}\varphi_{tc}(R, x, y) &\equiv (\forall z.M_1)(\exists z)(R(x, z) \wedge R(z, y)) \\ M_1 &\equiv \neg(x = y \vee E(x, y))\end{aligned}$$

2. Using $\forall$, replace two occurrences of R with one:

$$\begin{aligned}\varphi_{tc}(R, x, y) &\equiv (\forall z.M_1)(\exists z)(\forall uv.M_2)R(u, v) \\ M_2 &\equiv (u = x \wedge v = z) \vee (u = z \wedge v = y)\end{aligned}$$

3. Requantify x and y.

$$M_3 \equiv (x = u \wedge y = v)$$

$$\varphi_{tc}(R, x, y) \equiv [\,(\forall z.M_1)(\exists z)(\forall uv.M_2)(\exists xy.M_3)\,]\,R(x, y)$$

Every FO inductive definition is equivalent to a quantifier block.

$\mathrm{QB}_{tc} \equiv [(\forall z.M_1)(\exists z)(\forall uv.M_2)(\forall xy.M_3)]$

$$\varphi_{tc}(R, x, y) \equiv [(\forall z.M_1)(\exists z)(\forall uv.M_2)(\exists xy.M_3)]R(x, y)$$

$\mathrm{QB}_{tc} \equiv [(\forall z.M_1)(\exists z)(\forall uv.M_2)(\forall xy.M_3)]$

$$\varphi_{tc}(R,x,y) \equiv [(\forall z.M_1)(\exists z)(\forall uv.M_2)(\exists xy.M_3)]R(x,y)$$

$$\varphi_{tc}(R,x,y) \equiv [\mathrm{QB}_{tc}]R(x,y)$$

$\mathrm{QB}_{tc} \equiv [(\forall z.M_1)(\exists z)(\forall uv.M_2)(\forall xy.M_3)]$

$$\varphi_{tc}(R,x,y) \equiv [(\forall z.M_1)(\exists z)(\forall uv.M_2)(\exists xy.M_3)]R(x,y)$$

$$\varphi_{tc}(R,x,y) \equiv [\mathrm{QB}_{tc}]R(x,y)$$

$$\varphi_{tc}^{r}(\emptyset) \equiv [\mathrm{QB}_{tc}]^{r}(\textbf{false})$$

$\mathrm{QB}_{tc} \equiv [(\forall z.M_1)(\exists z)(\forall uv.M_2)(\forall xy.M_3)]$

$$\varphi_{tc}(R,x,y) \equiv [(\forall z.M_1)(\exists z)(\forall uv.M_2)(\exists xy.M_3)]R(x,y)$$

$$\varphi_{tc}(R,x,y) \equiv [\mathrm{QB}_{tc}]R(x,y)$$

$$\varphi_{tc}^r(\emptyset) \equiv [\mathrm{QB}_{tc}]^r(\textbf{false})$$

Thus, for any structure $\mathcal{A} \in \mathrm{STRUC}[\Sigma_g]$,

$$\mathcal{A} \in \text{REACH} \Leftrightarrow \mathcal{A} \models (\mathrm{LFP}\varphi_{tc})(s,t)$$

$$\Leftrightarrow \mathcal{A} \models ([\mathrm{QB}_{tc}]^{\lceil 1+\log \|\mathcal{A}\| \rceil}\, \textbf{false})(s,t)$$

$$\mathrm{CRAM}[t(n)] = \text{concurrent parallel random access machine; polynomial hardware, parallel time } O(t(n))$$

$$\mathrm{IND}[t(n)] = \text{first-order, depth } t(n) \text{ inductive definitions}$$

$$\mathrm{FO}[t(n)] = t(n) \text{ repetitions of a block of restricted quantifiers:}$$

$$\mathrm{QB} = [(Q_1 x_1 . M_1) \cdots (Q_k x_k . M_k)]; \quad M_i \text{ quantifier-free}$$

$$\varphi_n = \underbrace{[\mathrm{QB}][\mathrm{QB}] \cdots [\mathrm{QB}]}_{t(n)} M_0$$

parallel time = inductive depth = QB iteration

Thm. For all constructible, polynomially bounded $t(n)$,

$$\mathrm{CRAM}[t(n)] \;=\; \mathrm{IND}[t(n)] \;=\; \mathrm{FO}[t(n)]$$

parallel time = inductive depth = QB iteration

Thm. For all constructible, polynomially bounded $t(n)$,

$$\mathrm{CRAM}[t(n)] \;=\; \mathrm{IND}[t(n)] \;=\; \mathrm{FO}[t(n)]$$

proof idea: $\mathrm{CRAM}[t(n)] \supseteq \mathrm{FO}[t(n)]$: For QB with k variables, keep in memory current value of formula on all possible assignments, using n^k bits of global memory.

parallel time = inductive depth = QB iteration

Thm. For all constructible, polynomially bounded $t(n)$,

$$\mathrm{CRAM}[t(n)] \;=\; \mathrm{IND}[t(n)] \;=\; \mathrm{FO}[t(n)]$$

proof idea: $\mathrm{CRAM}[t(n)] \supseteq \mathrm{FO}[t(n)]$: For QB with k variables, keep in memory current value of formula on all possible assignments, using n^k bits of global memory.
Simulate each next quantifier in constant parallel time.

parallel time = inductive depth = QB iteration

Thm. For all constructible, polynomially bounded $t(n)$,

$$\mathrm{CRAM}[t(n)] \;=\; \mathrm{IND}[t(n)] \;=\; \mathrm{FO}[t(n)]$$

proof idea: $\mathrm{CRAM}[t(n)] \supseteq \mathrm{FO}[t(n)]$: For QB with k variables, keep in memory current value of formula on all possible assignments, using n^k bits of global memory.
Simulate each next quantifier in constant parallel time.

$\mathrm{CRAM}[t(n)] \subseteq \mathrm{FO}[t(n)]$: Inductively define new state of every bit of every register of every processor in terms of this global state at the previous time step. □

parallel time = inductive depth = QB iteration

Thm. For all constructible, polynomially bounded $t(n)$,

$$\mathrm{CRAM}[t(n)] \;=\; \mathrm{IND}[t(n)] \;=\; \mathrm{FO}[t(n)]$$

proof idea: $\mathrm{CRAM}[t(n)] \supseteq \mathrm{FO}[t(n)]$: For QB with k variables, keep in memory current value of formula on all possible assignments, using n^k bits of global memory.
Simulate each next quantifier in constant parallel time.

$\mathrm{CRAM}[t(n)] \subseteq \mathrm{FO}[t(n)]$: Inductively define new state of every bit of every register of every processor in terms of this global state at the previous time step. □

Thm. For all $t(n)$, even beyond polynomial,

$$\mathrm{CRAM}[t(n)] \quad = \quad \mathrm{FO}[t(n)]$$

For $t(n)$ poly bdd,

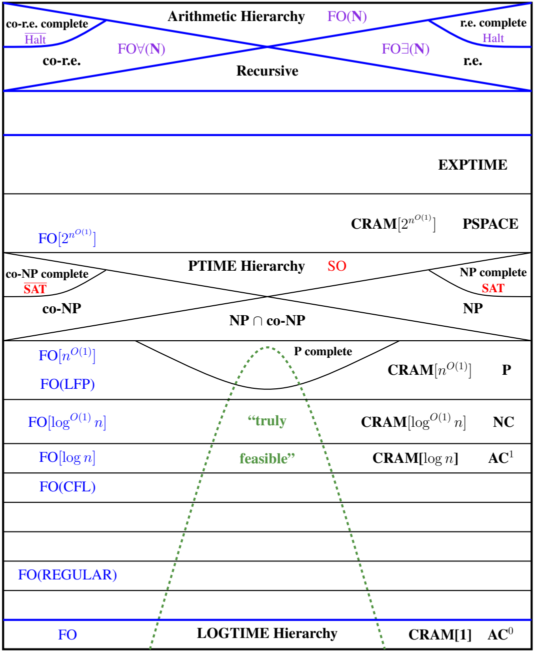
$$\text{CRAM}[t(n)]$$

$$=$$

$$\text{IND}[t(n)]$$

$$=$$

$$\text{FO}[t(n)]$$



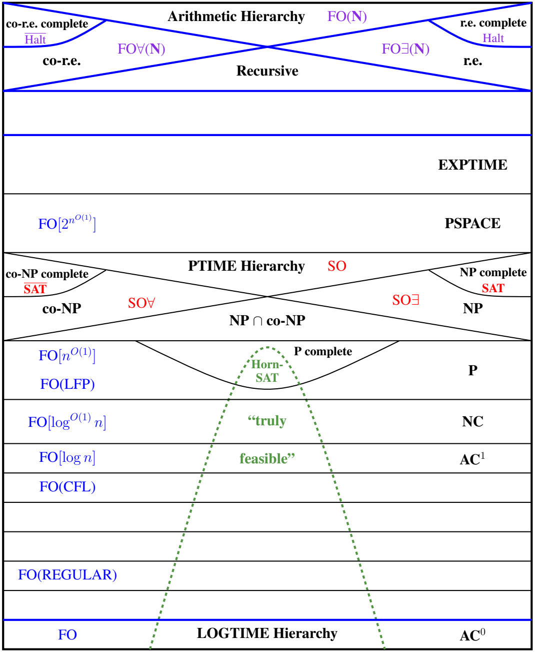
Remember that

for all $t(n)$,

$\mathrm{CRAM}[t(n)]$

$=$

$\mathrm{FO}[t(n)]$



Number of Variables Determines Amount of Hardware

Thm. For $k = 1, 2, \ldots,\quad \mathrm{DSPACE}[n^k] \;=\; \mathrm{VAR}[k+1]$

Number of Variables Determines Amount of Hardware

Thm. For $k = 1, 2, \ldots,\quad \mathrm{DSPACE}[n^k] \ = \ \mathrm{VAR}[k+1]$

Since variables range over a universe of size n, a constant number of variables can specify a polynomial number of gates.

Number of Variables Determines Amount of Hardware

Thm. For $k = 1, 2, \ldots,\quad \mathrm{DSPACE}[n^k] \;=\; \mathrm{VAR}[k+1]$

Since variables range over a universe of size n, a constant number of variables can specify a polynomial number of gates.

The proof is just a more detailed look at $\mathrm{CRAM}[t(n)] = \mathrm{FO}[t(n)]$.

Number of Variables Determines Amount of Hardware

Thm. For $k = 1, 2, \ldots, \quad \mathrm{DSPACE}[n^k] \;=\; \mathrm{VAR}[k+1]$

Since variables range over a universe of size n, a constant number of variables can specify a polynomial number of gates.

The proof is just a more detailed look at $\mathrm{CRAM}[t(n)] = \mathrm{FO}[t(n)]$.

A bounded number, k, of variables, is $k \log n$ bits and corresponds to n^k gates, i.e., polynomially much hardware.

Number of Variables Determines Amount of Hardware

Thm. For $k = 1, 2, \ldots, \quad \mathrm{DSPACE}[n^k] \ = \ \mathrm{VAR}[k+1]$

Since variables range over a universe of size n, a constant number of variables can specify a polynomial number of gates.

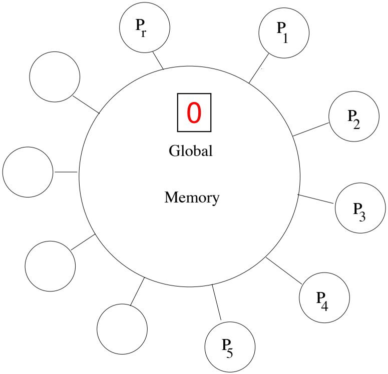
The proof is just a more detailed look at $\mathrm{CRAM}[t(n)] = \mathrm{FO}[t(n)]$.

A bounded number, k, of variables, is $k \log n$ bits and corresponds to n^k gates, i.e., polynomially much hardware.

A second-order variable of arity r is n^r bits, corresponding to 2^{n^r} gates.

SO: Parallel Machines with Exponential Hardware

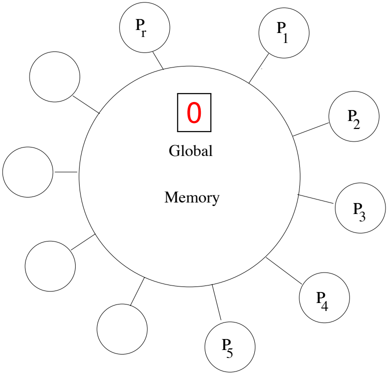
Given φ with n variables and m clauses, is $\varphi \in$ 3-SAT?



SO: Parallel Machines with Exponential Hardware

Given φ with n variables and m clauses, is $\varphi \in$ 3-SAT?

With $r = m2^n$ processors, recognize 3-SAT in constant time!

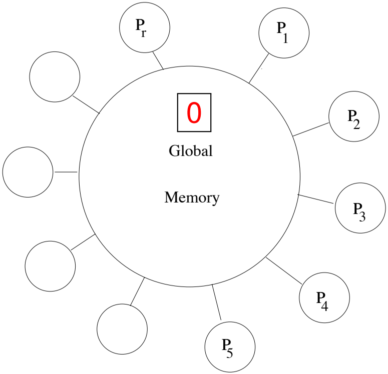


SO: Parallel Machines with Exponential Hardware

Given φ with n variables and m clauses, is $\varphi \in$ 3-SAT?

With $r = m2^n$ processors, recognize 3-SAT in constant time!

Let S be the first n bits of our processor number.

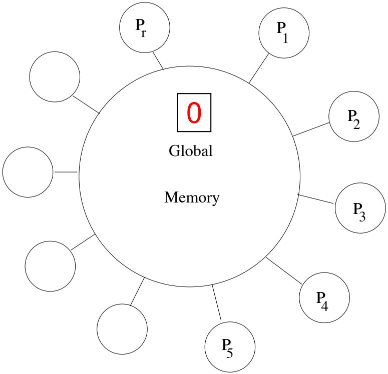


SO: Parallel Machines with Exponential Hardware

Given φ with n variables and m clauses, is $\varphi \in$ 3-SAT?

With $r = m2^n$ processors, recognize 3-SAT in constant time!

Let S be the first n bits of our processor number.
If processors $S1, \ldots Sm$ notice that truth assignment S makes all m clauses of φ true, then $\varphi \in$ 3-SAT,

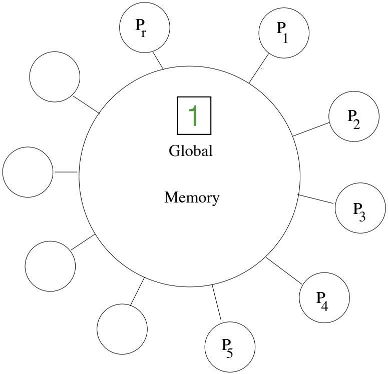


SO: Parallel Machines with Exponential Hardware

Given φ with n variables and m clauses, is $\varphi \in$ 3-SAT?

With $r = m2^n$ processors, recognize 3-SAT in constant time!

Let S be the first n bits of our processor number.
If processors $S1, \ldots Sm$ notice that truth assignment S makes all m clauses of φ true, then $\varphi \in$ 3-SAT, so $S1$ writes a 1.



SO: Parallel Machines with Exponential Hardware

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

SO: Parallel Machines with Exponential Hardware

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

proof: $\mathrm{SO}[t(n)]$ is like $\mathrm{FO}[t(n)]$ but using a quantifier block containing both first-order and second-order quantifiers. The proof is similar to $\mathrm{FO}[t(n)] = \mathrm{CRAM}[t(n)]$. □

SO: Parallel Machines with Exponential Hardware

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

proof: $\mathrm{SO}[t(n)]$ is like $\mathrm{FO}[t(n)]$ but using a quantifier block containing both first-order and second-order quantifiers. The proof is similar to $\mathrm{FO}[t(n)] = \mathrm{CRAM}[t(n)]$. □

Cor.

$$\mathrm{SO} \quad = \quad \text{PTIME Hierarchy} \quad = \quad \mathrm{CRAM}[1]\text{-}\mathrm{HARD}[2^{n^{O(1)}}]$$

SO: Parallel Machines with Exponential Hardware

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

proof: $\mathrm{SO}[t(n)]$ is like $\mathrm{FO}[t(n)]$ but using a quantifier block containing both first-order and second-order quantifiers. The proof is similar to $\mathrm{FO}[t(n)] = \mathrm{CRAM}[t(n)]$. □

Cor.

$$\mathrm{SO} = \text{PTIME Hierarchy} = \mathrm{CRAM}[1]\text{-}\mathrm{HARD}[2^{n^{O(1)}}]$$

$$\mathrm{SO}[n^{O(1)}] = \mathrm{PSPACE} = \mathrm{CRAM}[n^{O(1)}]\text{-}\mathrm{HARD}[2^{n^{O(1)}}]$$

SO: Parallel Machines with Exponential Hardware

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

proof: $\mathrm{SO}[t(n)]$ is like $\mathrm{FO}[t(n)]$ but using a quantifier block containing both first-order and second-order quantifiers. The proof is similar to $\mathrm{FO}[t(n)] = \mathrm{CRAM}[t(n)]$. □

Cor.

SO	=	PTIME Hierarchy	=	$\mathrm{CRAM}[1]\text{-}\mathrm{HARD}[2^{n^{O(1)}}]$
$\mathrm{SO}[n^{O(1)}]$	=	PSPACE	=	$\mathrm{CRAM}[n^{O(1)}]\text{-}\mathrm{HARD}[2^{n^{O(1)}}]$
$\mathrm{SO}[2^{n^{O(1)}}]$	=	EXPTIME	=	$\mathrm{CRAM}[2^{n^{O(1)}}]\text{-}\mathrm{HARD}[2^{n^{O(1)}}]$

Parallel Time versus Amount of Hardware

$$\begin{aligned} \mathrm{PSPACE} &= \mathrm{FO}[2^{n^{O(1)}}] &= \mathrm{CRAM}[2^{n^{O(1)}}]\text{-}\mathrm{HARD}[\mathrm{n}^{\mathrm{O}(1)}] \\ &= \mathrm{SO}[n^{O(1)}] &= \mathrm{CRAM}[n^{O(1)}]\text{-}\mathrm{HARD}[2^{\mathrm{n}^{\mathrm{O}(1)}}] \end{aligned}$$

Parallel Time versus Amount of Hardware

$$\begin{aligned} \text{PSPACE} &= \text{FO}[2^{n^{O(1)}}] &= \text{CRAM}[2^{n^{O(1)}}]\text{-HARD}[n^{O(1)}] \\ &= \text{SO}[n^{O(1)}] &= \text{CRAM}[n^{O(1)}]\text{-HARD}[2^{n^{O(1)}}] \end{aligned}$$

- We would love to understand this tradeoff.

Parallel Time versus Amount of Hardware

$$\begin{aligned} \mathrm{PSPACE} &= \mathrm{FO}[2^{n^{O(1)}}] &= \mathrm{CRAM}[2^{n^{O(1)}}]\text{-}\mathrm{HARD}[\mathrm{n}^{\mathrm{O}(1)}] \\ &= \mathrm{SO}[n^{O(1)}] &= \mathrm{CRAM}[n^{O(1)}]\text{-}\mathrm{HARD}[2^{\mathrm{n}^{\mathrm{O}(1)}}] \end{aligned}$$

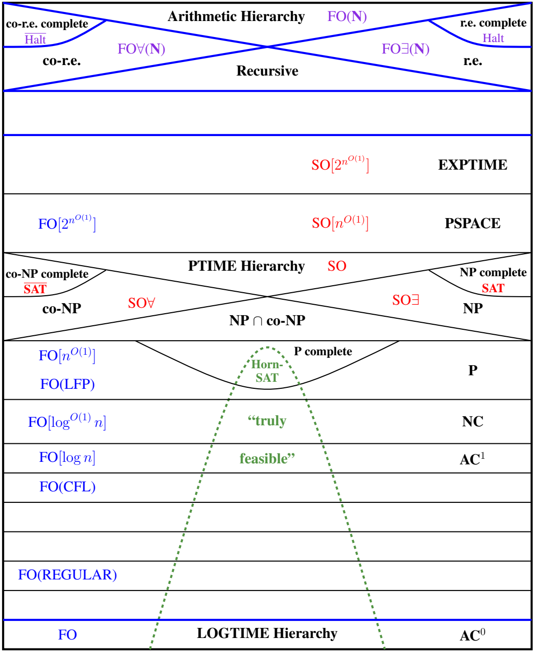
- We would love to understand this tradeoff.
- Is there such a thing as an inherently sequential problem?, i.e., is $\mathrm{NC} \neq \mathrm{P}$?

Parallel Time versus Amount of Hardware

$$\begin{aligned} \mathrm{PSPACE} &= \mathrm{FO}[2^{n^{O(1)}}] = \mathrm{CRAM}[2^{n^{O(1)}}]\text{-}\mathrm{HARD}[n^{O(1)}] \\ &= \mathrm{SO}[n^{O(1)}] = \mathrm{CRAM}[n^{O(1)}]\text{-}\mathrm{HARD}[2^{n^{O(1)}}] \end{aligned}$$

- We would love to understand this tradeoff.
- Is there such a thing as an inherently sequential problem?, i.e., is $\mathrm{NC} \neq \mathrm{P}$?
- Same tradeoff as number of variables vs. number of iterations of a quantifier block.

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



Recent Breakthroughs in Descriptive Complexity

Theorem [Ben Rossman] Any first-order formula with any numeric relations $(\leq, +, \times, \ldots)$ that means "I have a clique of size k" must have at least $k/4$ variables.

Creative new proof idea using Håstad's Switching Lemma gives the essentially optimal bound.

This lower bound is for a fixed formula, if it were for a sequence of polynomially-sized formulas, i.e., a fixed-point formula, it would follow that CLIQUE $\notin \mathrm{P}$ and thus $\mathrm{P} \neq \mathrm{NP}$.

Best previous bounds:

- k variables necessary and sufficient without ordering or other numeric relations [I 1980].
- Nothing was known with ordering except for the trivial fact that 2 variables are not enough.

Theorem [Martin Grohe] Fixed-Point Logic with Counting captures Polynomial Time on all classes of graphs with excluded minors.

Grohe proves that for every class of graphs with excluded minors, there is a constant k such that two graphs of the class are isomorphic iff they agree on all k-variable formulas in fixed-point logic with counting.

Using Ehrenfeucht-Fraïssé games, this can be checked in polynomial time, $(O(n^k(\log n)))$. In the same time we can give a canonical description of the isomorphism type of any graph in the class. Thus every class of graphs with excluded minors admits the same general polynomial time canonization algorithm: we're isomorphic iff we agree on all formulas in C_k and in particular, you are isomorphic to me iff your C_k canonical description is equal to mine.

Thm. REACH is complete for $\mathrm{NL} = \mathrm{NSPACE}[\log n]$.

Proof: Let $A \in \mathrm{NL}$, $A = \mathcal{L}(N)$, uses $c \log n$ bits of worktape.
Input w, $\quad n = |w|$

$$w \mapsto \mathrm{CompGraph}(N, w) \quad = \quad (V, E, s, t)$$

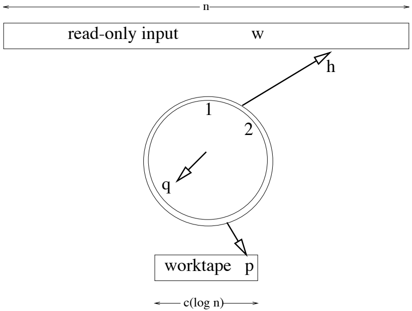
$$V = \{\mathrm{ID} = \langle q, h, p\rangle \mid q \in \text{States}(N), h \leq n, |p| \leq c\lceil \log n\rceil\}$$

$$E = \{(\mathrm{ID}_1, \mathrm{ID}_2) \mid \mathrm{ID}_1(w) \underset{N}{\longrightarrow} \mathrm{ID}_2(w)\}$$

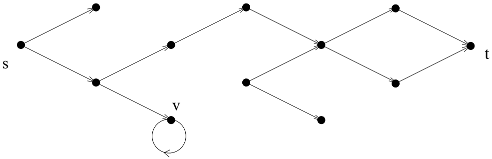
$$s = \text{initial ID}$$

$$t = \text{accepting ID}$$

NSPACE[log n] Turing Machine



Claim. $w \in \mathcal{L}(N) \Leftrightarrow \mathrm{CompGraph}(N, w) \in \mathsf{REACH}$



□

Cor: $\mathrm{NL} \subseteq \mathrm{P}$

Proof: REACH $\in \mathrm{P}$

P is closed under (logspace) reductions.

i.e., $(B \in \mathrm{P} \quad \wedge \quad A \leq B) \quad \Rightarrow \quad A \in \mathrm{P}$ $\square$

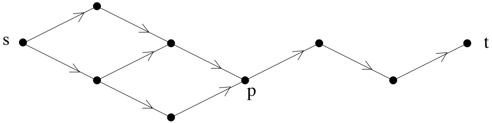
Prop.

$$\mathrm{NSPACE}[s(n)] \subseteq \mathrm{NTIME}[2^{O(s(n))}] \subseteq \mathrm{DSPACE}[2^{O(s(n))}]$$

We can do much better!

Savitch's Theorem

$$\text{REACH} \in \text{DSPACE}(\log n)^2$$



proof:

$$\begin{aligned} G \in \text{REACH} &\Leftrightarrow G \models \text{PATH}_n(s,t) \\ \text{PATH}_1(x,y) &\equiv x = y \lor E(x,y) \\ \text{PATH}_{2d}(x,y) &\equiv \exists z\,(\text{PATH}_d(x,z) \land \text{PATH}_d(z,y)) \end{aligned}$$

$S_n(d)$ = space to check paths of dist. d in n-nodegraphs

$$\begin{aligned} S_n(n) &= \log n + S_n(n/2) \\ &= O((\log n)^2) \end{aligned}$$

Savitch's Theorem

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

proof: Let $A \in \mathrm{NSPACE}[s(n)]$; $\quad A = \mathcal{L}(N)$

$$w \in A \quad \Leftrightarrow \quad \mathrm{CompGraph}(N, w) \in \text{REACH}$$

$$|w| = n; \qquad |\mathrm{CompGraph}(N, w)| = 2^{O(s(n))}$$

Testing if $\mathrm{CompGraph}(N, w) \in$ REACH takes space,

$$\begin{aligned}(\log(|\mathrm{CompGraph}(N, w)|))^2 &= (\log(2^{O(s(n))}))^2 \\ &= O((s(n))^2)\end{aligned}$$

From w build $\mathrm{CompGraph}(N, w)$ in DSPACE[s(n)]. □

$\overline{\text{REACH}} \in \text{NL}$

proof: Fix G, let $N_d = \left|\{v \mid \text{distance}(s,v) \le d\}\right|$

Claim: The following problems are in NL:

1. $\text{dist}(x,d)$: $\text{distance}(s,x) \le d$
2. $\text{NDIST}(x,d;m)$: if $m = N_d$ then $\neg\text{dist}(x,d)$

proof:

1. Guess the path of length $\le d$ from s to x.
2. Guess m vertices, $v \ne x$, with $\text{dist}(v,d)$.

```
c := 0;
for v := 1 to n do {    // nondeterministically
    ( dist(v, d) && v ≠ x; c++ )    ||
    ( no-op )
}
if (c == m) then ACCEPT
```

□

Claim. We can compute N_d in NL.

proof: By induction on d.

Base case: $N_0 = 1$

Inductive step: Suppose we have N_d.

1. $c := 0;$
2. **for** $v := 1$ to n **do** $\{$ // nondeterministically
3. $(\ \mathrm{dist}(v, d+1); c++)\quad ||$
4. $(\forall z\,(\mathrm{NDIST}(z, d; N_d) \ \vee\ (z \neq v \wedge \neg E(z, v))))$
5. $\}$
6. $N_{d+1} := c$

$\square$

$$G \in \overline{\mathrm{REACH}} \quad \Leftrightarrow \quad \mathrm{NDIST}(t, n; N_n)$$

$\square$

Thm. $\mathrm{NSPACE}[s(n)] = \text{co-}\mathrm{NSPACE}[s(n)]$.

proof: Let $A \in \mathrm{NSPACE}[s(n)]$; $A = \mathcal{L}(N)$

$$w \in A \quad \Leftrightarrow \quad \mathrm{CompGraph}(N, w) \in \text{REACH}$$

$$|w| = n; \qquad |\mathrm{CompGraph}(N, w)| = 2^{O(s(n))}$$

Testing if $\mathrm{CompGraph}(N, w) \in \overline{\text{REACH}}$ takes space,

$$\begin{aligned} \log(|\mathrm{CompGraph}(N, w)|) &= \log(2^{O(s(n))}) \\ &= O(s(n)) \end{aligned}$$

□

What We Know

- **Diagonalization**: more of the same resource gives us more:

  $\mathrm{DTIME}[n] \subsetneq \mathrm{DTIME}[n^2]$,

  same for DSPACE, NTIME, NSPACE, ...

What We Know

- **Diagonalization**: more of the same resource gives us more:

  $\mathrm{DTIME}[n] \subsetneq \mathrm{DTIME}[n^2]$,

  same for DSPACE, NTIME, NSPACE, ...

- **Natural Complexity Classes have Natural Complete Problems**

  SAT for NP, CVAL for P, QSAT for PSPACE, ...

What We Know

- **Diagonalization**: more of the same resource gives us more:

  $\mathrm{DTIME}[n] \subsetneq \mathrm{DTIME}[n^2]$,

  same for DSPACE, NTIME, NSPACE, ...

- **Natural Complexity Classes have Natural Complete Problems**

  SAT for NP, CVAL for P, QSAT for PSPACE, ...

- **Major Missing Idea**: concept of work or conservation of energy in computation, i.e,

  in order to solve SAT or other hard problem we must do a certain amount of computational work.

Strong Lower Bounds on $\mathrm{FO}[t(n)]$ for small $t(n)$

- [Sipser]: strict first-order alternation hierarchy: FO.

Strong Lower Bounds on $\mathrm{FO}[t(n)]$ for small $t(n)$

- [Sipser]: strict first-order alternation hierarchy: FO.
- [Beame-Håstad]: hierarchy remains strict up to $\mathrm{FO}[\log n / \log\log n]$.

Strong Lower Bounds on $\mathrm{FO}[t(n)]$ for small $t(n)$

- [Sipser]: strict first-order alternation hierarchy: FO.
- [Beame-Håstad]: hierarchy remains strict up to $\mathrm{FO}[\log n / \log\log n]$.
- $\mathrm{NC}^1 \subseteq \mathrm{FO}[\log n / \log\log n]$ and this is tight.

Strong Lower Bounds on $\mathrm{FO}[t(n)]$ for small $t(n)$

- [Sipser]: strict first-order alternation hierarchy: FO.
- [Beame-Håstad]: hierarchy remains strict up to $\mathrm{FO}[\log n/\log\log n]$.
- $\mathrm{NC}^1 \subseteq \mathrm{FO}[\log n/\log\log n]$ and this is tight.
- Does REACH require $\mathrm{FO}[\log n]$? This would imply $\mathrm{NC}^1 \neq \mathrm{NL}$.

Does It Matter? How important is P $\neq$ NP?

- Much is known about approximation, e.g., some NP complete problems, e.g., Knapsack, Euclidean TSP, can be approximated as closely as we want, others, e.g., Clique, can't be.

Does It Matter? How important is P $\neq$ NP?

- Much is known about approximation, e.g., some NP complete problems, e.g., Knapsack, Euclidean TSP, can be approximated as closely as we want, others, e.g., Clique, can't be.
- We conjecture that SAT requires $\mathrm{DTIME}[\Omega(2^{\epsilon n})]$ for some $\epsilon > 0$, but no one has yet proved that it requires more than $\mathrm{DTIME}[n]$.

Does It Matter? How important is P $\neq$ NP?

- Much is known about approximation, e.g., some NP complete problems, e.g., Knapsack, Euclidean TSP, can be approximated as closely as we want, others, e.g., Clique, can't be.
- We conjecture that SAT requires $\mathrm{DTIME}[\Omega(2^{\epsilon n})]$ for some $\epsilon > 0$, but no one has yet proved that it requires more than $\mathrm{DTIME}[n]$.
- Basic trade-offs are not understood, e.g., trade-off between time and number of processors. Are any problems inherently sequential? How can we best use mulitcores?

Does It Matter? How important is P $\neq$ NP?

- Much is known about approximation, e.g., some NP complete problems, e.g., Knapsack, Euclidean TSP, can be approximated as closely as we want, others, e.g., Clique, can't be.
- We conjecture that SAT requires $\mathrm{DTIME}[\Omega(2^{\epsilon n})]$ for some $\epsilon > 0$, but no one has yet proved that it requires more than $\mathrm{DTIME}[n]$.
- Basic trade-offs are not understood, e.g., trade-off between time and number of processors. Are any problems inherently sequential? How can we best use mulitcores?
- **SAT solvers** are impressive new general purpose problem solvers, e.g., used in model checking, AI planning, code synthesis. How good are current SAT solvers? How much can they be improved?

Descriptive Complexity

Fact: For constructible $t(n)$, $\mathrm{FO}[t(n)] \;=\; \mathrm{CRAM}[t(n)]$

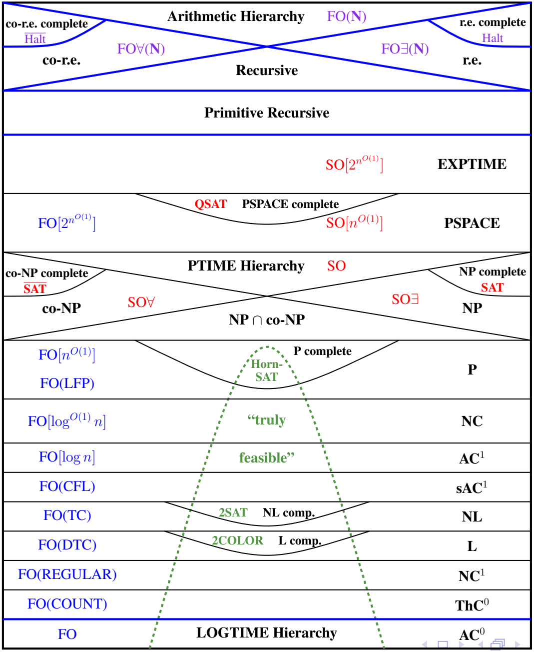
Fact: For $k = 1, 2, \ldots,$ $\mathrm{VAR}[k+1] \;=\; \mathrm{DSPACE}[n^k]$

The complexity of computing a query is closely tied to the complexity of describing the query.

$$\mathrm{P} = \mathrm{NP} \quad \Leftrightarrow \quad \mathrm{FO(LFP)} = \mathrm{SO}$$

$$\mathrm{ThC}^0 = \mathrm{NP} \quad \Leftrightarrow \quad \mathrm{FO(MAJ)} = \mathrm{SO}$$

$$\mathrm{P} = \mathrm{PSPACE} \quad \Leftrightarrow \quad \mathrm{FO(LFP)} = \mathrm{SO(TC)}$$


Arithmetic Hierarchy FO(N)
co-r.e. complete
$\overline{\text{Halt}}$
r.e. complete
Halt
FO∀(N)
FO∃(N)
co-r.e.
r.e.
Recursive
Primitive Recursive
$SO[2^{n^{O(1)}}]$
EXPTIME
QSAT PSPACE complete
$FO[2^{n^{O(1)}}]$
$SO[n^{O(1)}]$
PSPACE
PTIME Hierarchy SO
co-NP complete
$\overline{\text{SAT}}$
NP complete
SAT
SO∀
SO∃
co-NP
NP
NP ∩ co-NP
$FO[n^{O(1)}]$
P complete
Horn-SAT
FO(LFP)
P
$FO[\log^{O(1)} n]$
"truly
NC
$FO[\log n]$
feasible"
AC^1
FO(CFL)
sAC^1
FO(TC)
2SAT NL comp.
NL
FO(DTC)
2COLOR L comp.
L
FO(REGULAR)
NC^1
FO(COUNT)
ThC^0
FO
LOGTIME Hierarchy
AC^0