# Introduction to Descriptive Complexity

**George Voutsadakis**[1]

[1]Mathematics and Computer Science
Lake Superior State University

LSSU Math 600

Subsection 1

# Complete Problems for PSPACE

# Completeness of QSAT for PSPACE

- We know, by a previous theorem, that PSPACE is equal to alternating polynomial time ATIME$[n^{O(1)}]$.
- Recall QSAT, the quantified satisfiability problem.

**Proposition**

The quantified boolean satisfaction problem (QSAT) is complete for PSPACE via first-order reductions.

- By a previous proposition, QSAT is in ATIME$[n]$.

  Therefore, QSAT is in PSPACE.

  To show completeness, let $M$ be an alternating machine.

  Suppose $M$ makes $n^k$ moves for inputs of size $n$.

  We know that $M$ can be put in a normal form in which:
  - It writes down its $n^k$ alternating choices $\overline{c} = c_1 c_2 \dots c_{n^k}$;
  - Then deterministically evaluates its input, using choice vector $\overline{c}$.

  Let the corresponding deterministic time $n^k$ machine be $D$.

# Completeness of QSAT for PSPACE (Cont'd)

- We have that, for all inputs $\mathcal{A}$,

$$M(\text{bin}(\mathcal{A}))\downarrow \Leftrightarrow\ (\exists c_1)(\forall c_2)\cdots(Q_{n^k}\, c_{n^k})(D(\overline{c}, \text{bin}(\mathcal{A}))\downarrow).$$

$D$ can be thought of as an NP machine.

By a previous theorem, there is a first-order reduction from the language accepted by $D$ to SAT.

Let $f$ be the first-order query such that, for all $\overline{c}$ and $\mathcal{A}$,

$$D(\overline{c}, \text{bin}(\mathcal{A}))\downarrow \Leftrightarrow\ f(\mathcal{A}) \in \text{SAT}.$$

Let the new boolean variables in $f(\mathcal{A})$ be $d_1 \ldots d_{t(n)}$.

Then, finally, we have,

$$M(\text{bin}(\mathcal{A}))\downarrow \Leftrightarrow\ \text{“}(\exists c_1)(\forall c_2)\cdots(Q_{n^k}\, c_{n^k})(\exists d_1 \ldots d_{t(n)})f(\mathcal{A})'' \in \text{QSAT}.$$

# The Problem $U_{\text{PSPACE}}$

- The **standard universal complete problem** for PSPACE is

$$U_{\text{PSPACE}} = \{M\#w\#^r : M \text{ accepts } w \text{ using at most } r \text{ tape cells}\}.$$

**Theorem**

$U_{\text{PSPACE}}$ is complete for PSPACE via first-order reductions.

# $k$-Local Graphs

- Another complete problem for PSPACE is a generalization of REACH to graphs with:
  - A polynomial-size description;
  - Exponentially many nodes.
- Define a $k$-**local graph** to be a graph on vertex set $\{0,1\}^n$, such that:
  - For each vertex $u$, there is a unique next vertex $v$;
  - Bit $i$ of $v$ is determined uniformly by bits $i-k, i-k+1, \ldots, i+k$ of $u$.
- Note that a $k$-local graph can be presented as a table of size $2^{2k+1}$.
- We wish to keep the tables size at most $n$ so that they can be encoded by a single unary input relation $R$.
- So we insist that $k \leq \left\lfloor \frac{\log n}{2} - 1 \right\rfloor$.

# $\mathrm{REACH}_{d\ell}$

- Let $\mathcal{A}$ be a structure of vocabulary

$$\tau_\ell = \langle R^1, S^1, T^1 \rangle$$

consisting of three unary relations.
  - $R$ encodes the transition relation;
  - $S$ is the source node;
  - $T$ is the terminal node.
- Define boolean query $\mathrm{REACH}_{d\ell}$ to be the set of structures $\mathcal{A} \in \mathrm{STRUC}[\tau_\ell]$, such that

there is an $R^{\mathcal{A}}$ path from $S^{\mathcal{A}}$ to $T^{\mathcal{A}}$.

# Expressing $\mathrm{REACH}_{d\ell}$

- Let unary relation variable $A$ denote our present position.
- To define the next move relation, we need, for each $i$, to:
  - Read the $\lfloor \log n \rfloor$ bits

$$w = A[[i-k \ldots i+k]];$$

  - Apply $R(w)$ to get bit $i$ of $A'$.
- Note that in the following:
  - $\log n$ can be computed, it being the largest $r$, such that $\mathrm{BIT}(\max, r)$ holds;
  - The additions are mod $n$, so we think of the $n$ bits of $A$ as being in a loop.

# Expressing $\mathrm{REACH}_{d\ell}$ (Cont'd)

- In the following, we also let:
  - $\alpha(i, w, A)$ mean that the binary representation of $w$ encodes bits $A[[i-k \ldots i+k]]$,

$$\alpha(i,w,A) \equiv \left(\exists k.k = \left\lfloor \frac{\log n}{2} - 1 \right\rfloor\right)(\forall j.j \leq \lfloor \log n \rfloor) \\ (A(i-k+j) \leftrightarrow \mathrm{BIT}(w,j)).$$

  - $\delta$ be the resulting edge relation on unary relations,

$$\delta(A,A') \equiv (\forall i)(\exists w)(\alpha(i,w,A) \wedge A'(i) \leftrightarrow R(w)).$$

- $\mathrm{REACH}_{d\ell}$ in the language SO(monadic, DTC) - second-order logic restricted to monadic relation variables, plus the deterministic transitive closure operator - is

$$\mathrm{REACH}_{d\ell} \equiv (\mathrm{DTC}_{AA'\delta})(S,T).$$

# Completeness of $\text{REACH}_{d\ell}$ for PSPACE

**Proposition**

Problem $\text{REACH}_{d\ell}$ is complete for PSPACE via first-order reductions. In fact, $\text{REACH}_{d\ell}$ remains complete when $k$ is restricted to a fixed constant.

- $\text{REACH}_{d\ell}$ is in PSPACE, and in fact in DSPACE$[n]$.

  The algorithm is the same as the one showing $\text{REACH}_d \in \text{L}$.

  We need $n$ bits to record the current position.

  We need another $n$ bits to keep a counter to avoid looping.

# Completeness of $\text{REACH}_{d\ell}$ for PSPACE (Cont'd)

- Let $M_0$ be a linear-space Turing machine that accepts $U_{\text{PSPACE}}$.

  $M_0$ has a fixed state set.

  So we may encode its instantaneous description in a way that is $k$-local for appropriate $k$, twice the log of the number of states suffices.

  Then a first-order reduction from $U_{\text{PSPACE}}$ to $\text{REACH}_{d\ell}$:
  - Maps input string $w$ to starting vertex $S$;
  - Lets $T$ be $M_0$'s fixed accept ID;
  - Lets $R$ be the encoding of $M_0$'s fixed transition relation.

Subsection 2

# Partial Fixed Points

# Monotonicity and Polynomiality of Fixed-Points

- We saw that, for all polynomially bounded and parallel time constructible $t(n)$,

$$\mathsf{CRAM}[t(n)] = \mathsf{IND}[t(n)] = \mathsf{FO}[t(n)].$$

- A relation of arity $k$ that is being defined has $n^k$ possible tuples.
- Monotonicity implies that after a tuple is added to a relation it can never be removed.
- Thus, an inductive computation cannot make more than $n^k$ steps.
- So monotone inductive definitions are polynomially bounded.
- It follows that, when $t(n)$ is super-polynomial, the theorem breaks down for $\mathsf{IND}[t(n)]$.

# Introducing Iterative Definitions

- We now generalize inductive definitions to **iterative definitions** in which the requirement of monotonicity is removed.
- Any tuple in a $k$-ary relation may be added or removed, depending on the whole relation at the previous time step.
- Thus, such an algorithm can usefully take no more than $2^{n^k}$ steps.

# Iterative Definitions

## Definition

Define ITER$[t(n), \text{arity } k]$ to be the set of properties definable by iterating the simultaneous first-order definitions of a set of $c$ relations of arity $k$ $t(n)$ times, for some constant $c$.
Let ITER$[t(n)]$ be the set of boolean queries definable by $O(t(n))$ iterations, regardless of the arity,

$$\text{ITER}[t(n)] = \bigcup_{k=1}^{\infty} \text{ITER}[t(n), \text{arity } k].$$

After $t(n) = 2^{cn^k}$ iterations, these relations will either reach a fixed point or be in a cycle. Thus, define

$$\text{ITER}[\text{arity } k] = \bigcup_{c=1}^{\infty} \text{ITER}[2^{cn^k}, \text{arity } k].$$

# Inductive and Iterative Definitions Coincide for Poly Time

- Observe that ITER$[t(n)]$ is exactly the generalization of IND$[t(n)]$ that removes the monotonicity requirement.

**Theorem**

For all polynomially-bounded $t(n)$,

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

- We can make an iterative definition of a relation $R$ inductive by adding a time component.

  We can simultaneously define the new values $R'(\overline{x}, \overline{t}+1)$ and $\overline{R}'(\overline{x}, \overline{t}+1)$ in terms of $R'(\cdot, t)$ and $\overline{R}'(\cdot, t)$.

# Example

- The following iterative definition proves $\mathrm{REACH}_{d\ell} \in \mathrm{ITER}[\text{arity } 1]$.
- In this definition, we simultaneously define:
  - Two booleans start and accept;
  - The relation $P$ indicating our current position.
- Recall that $\alpha(i, w, P)$ means that $w$ encodes the relevant bits of $P$, $P[[i-k \ldots i+k]]$.

$$
\begin{aligned}
\text{start}' &:= \textbf{false};\\
\text{accept}' &:= \text{accept} \vee (\forall x)(T(x) \leftrightarrow P(x));\\
P'(i) &:= (\text{start} \wedge S(i)) \vee (\neg\text{start} \wedge \text{accept} \wedge P(i))\\
&\quad \vee (\neg\text{start} \wedge \neg\text{accept} \wedge (\exists w)(\alpha(i, w, P) \wedge R(w))).
\end{aligned}
$$

# Introducing the Partial Fixed Point

- The iterative definitions in the preceding example were designed to stop once $T$ is reached.
- In general, however, such iterative definitions do not have to reach a fixed point, since they could cycle forever.
- Thus, the appropriate generalization of the least fixed point operator is called *partial fixed point* (*PFP*).

# Partial Fixed Point

## Definition (Partial Fixed Point)

Given an iterative, not necessarily monotone, definition

$$\varphi(R^k, x_1, \ldots, x_k),$$

define its partial fixed point as follows. For any structure $\mathcal{A}$,

$$(\mathrm{PFP}_{R,x_1\ldots x_k}\varphi)^{\mathcal{A}} = \begin{cases} (\varphi^{\mathcal{A}})^r(\varnothing), & \text{if } (\varphi^{\mathcal{A}})^r(\varnothing) = (\varphi^{\mathcal{A}})^{r+1}(\varnothing), \\ \varnothing, & \text{if there is no such } r. \end{cases}$$

The above definition defines all uses of partial fixed point, even if they define a loop. In practice, we never need to write a looping iterative definition.

Define FO(PFP) to be the closure of first -order logic under applications of the partial fixed point operator.

# Iteration of PFP

**Theorem**

For any parallel-time constructible $t(n)$,

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

In particular, $\mathrm{PSPACE} = \mathrm{FO}(\mathrm{PFP}) = \mathrm{FO}[2^{n^{O(1)}}]$.

- $\mathrm{CRAM}[t(n)] \subseteq \mathrm{ITER}[t(n)]$ has almost the same proof as that of $\mathrm{CRAM}[t(n)] \subseteq \mathrm{IND}[t(n)]$ for polynomially bounded $t(n)$.

  The only change is that before we kept track of the time $t$.

  This allowed us to avoid keeping track of memory.

  So, we obtained, as a corollary, that, for $t(n)$ polynomially bounded, having more than polynomial memory is not useful.

# Iteration of PFP (Cont'd)

- Now, we keep track of each of the polynomial memory locations in our iterative definition, just as we keep track of all the registers of all the processors.

  The contents of a memory location at the next step is the same as it was at the previous step, unless it was assigned at the previous step.

  With this change, we no longer need to maintain the time variable, and the proof goes through without further change.

# Iteration of PFP (Cont'd)

- The containment $\mathrm{IND}[t(n)] \subseteq \mathrm{FO}[t(n)]$ depended on the fact that the inductive definition was monotone and the following lemma:

  An $R$-positive first-order formula $\varphi$ can be written as

  $$\varphi(R, x_1, \ldots, x_k) \equiv (Q_1 z_1 . M_1) \cdots (Q_s z_s . M_s)(\exists x_1 \ldots x_k . M_{s+1}) R(x_1, \ldots, x_k),$$

  where the $M_i$'s are quantifier-free formulas in which $R$ does not occur.

  If the inductive definition $\varphi$ closes in at most $t(n)$ iterations, it follows from that lemma that

  $$\mathrm{LFP}(\varphi) \equiv [(Q_1 z_1 . M_1) \cdots (Q_s z_s . M_s)(\exists x_1 \ldots x_k . M_{s+1})]^{t(n)} \textbf{false}.$$

  The quantifier-free formula after the iterated quantifier block can be taken to be "**false**", since the inductive definition can be evaluated starting with the empty set and repeatedly applying $\varphi$.

# Iteration of PFP (Cont'd)

- Now, let

$$\varphi(R, x_1, \ldots, x_k)$$

be a, not necessarily monotone, first-order formula.

Let $b$ be a new boolean variable.

Let $R'$ be an arity $k+1$ relation symbol, where:

- $R'(\overline{x}, 0)$ will correspond to $R(\overline{x})$;
- $R'(\overline{x}, 1)$ will correspond to $\neg R(\overline{x})$.

We now generalize the iterative definition of $\varphi$ to $\varphi'$ by replacing in $\varphi$:

- All occurrences of $R(\overline{y})$ by $R'(\overline{y}, 0)$;
- All occurrences of $\neg R(\overline{y})$ by $R'(\overline{y}, 1)$.

We do the opposite for $\neg\varphi$

# Iteration of PFP (Cont'd)

- As a result, we obtain

$$\varphi'(R', \overline{x}, b) \equiv (b = 0 \wedge \varphi) \vee (b = 1 \wedge \neg\varphi).$$

Thus, $\varphi'$ is positive and a previous lemma applies.
Let QB be the resulting quantifier block, so that,

$$\varphi'(R', \overline{x}, b) \equiv [\mathrm{QB}]R'(\overline{x}, b).$$

If iterative definition $\varphi$ closes in at most $t(n)$ iterations, then

$$\mathrm{PFP}(\varphi) \equiv [\mathrm{QB}]^{t(n)}(b = 1).$$

Here the quantifier-free formula is $b = 1$, corresponding to the initial step of the iterative definition in which for all $\overline{x}$, $R'(\overline{x}, 0)$ is false and $R'(\overline{x}, 1)$ is true.
Thus, $\mathrm{ITER}[t(n)] \subseteq \mathrm{FO}[t(n)]$, as desired.
The containment $\mathrm{FO}[t(n)] \subseteq \mathrm{CRAM}[t(n)]$ goes through unchanged.

## Subsection 3

**DSPACE[$n^k$] = VAR[$k+1$]**

# The Class FO-VAR[$t(n), v$]

## Definition

A query is in class FO-VAR[$t(n), v$] iff it is in FO[$t(n)$] and the relevant quantifier-block contains only $v$ first-order variables - although it may contain some constant number of additional boolean variables.
As in the definition of ITER[arity $v$], the truth assignment of all the variables will cycle or stabilize after at most $t(n) = 2^{cn^{v-1}}$ iterations.
Thus,

$$\text{VAR}[v+1] = \bigcup_{c=1}^{\infty} \text{FO-VAR}[2^{cn^v}, v+1].$$

# Accommodating Conjunctions and Disjunctions

## Lemma

Suppose that we have two quantifier blocks with identical quantifiers in identical order (ignoring any boolean quantifiers):

$$\begin{aligned} \mathrm{QB}_1 &= [(Q_1 v_1.M_1)\cdots(Q_s v_s.M_s)], \\ \mathrm{QB}_2 &= [(Q_1 v_1.N_1)\cdots(Q_s v_s.N_s)]. \end{aligned}$$

Then the conjunction and disjunction of these quantifier blocks may be written in the same form.

- The conjunction, for example, can be written with an extra, universally quantified boolean variable:

$$[\mathrm{QB}_1]\varphi \wedge [\mathrm{QB}_2]\varphi \equiv [(\forall b)(Q_1 v_1.R_1)\cdots(Q_s v_s.R_s)]\varphi,$$

where,

$$R_i = (b = 0 \wedge M_i) \vee (b = 1 \wedge N_i).$$

# Number of Domain Variables

**Theorem**

For $k = 1, 2, \ldots,$

$$\mathrm{DSPACE}[n^k] = \mathrm{VAR}[k+1] = \mathrm{ITER}[\text{arity } k].$$

- The proof is accomplished using three lemmas proving the following containments:

$$\mathrm{DSPACE}[n^k] \subseteq \mathrm{VAR}[k+1] \subseteq \mathrm{ITER}[\text{arity } k] \subseteq \mathrm{DSPACE}[n^k].$$

# DSPACE[$n^k$] $\subseteq$ VAR[$k+1$]

**Lemma**

DSPACE[$n^k$] $\subseteq$ VAR[$k+1$].

- Let $M$ be a DSPACE[$n^k$] Turing machine.

  $M$'s work space consists of $n^k$ tape cells.

  Each cell holds a symbol from some finite alphabet $\Sigma$.

  The contents of $M$'s tape at time $t+1$ is a deterministic, local transformation of the contents at time $t$.

  Namely, the contents of cell $p$ at time $t+1$ is a function of the contents of cells $p-1, p, p+1$ at time $t$.

  We write a logical formula $C_t(\overline{x}, \overline{b})$ meaning that, after step $t$ of $M$'s computation, the cell at position $\overline{x}$ is $\overline{b}$, where:

  - $\overline{x} = x_1, \ldots, x_k$ is a $k$-tuple of variables ranging over $\{0, \ldots, n-1\}$;
  - $\overline{b}$ is a tuple of boolean variables coding an element of $\Sigma$.

# DSPACE[$n^k$] $\subseteq$ VAR[$k+1$] (Cont'd)

- The following is an iterative definition of $C_t$:

$$C_{t+1}(\overline{x},\overline{b}) = \bigvee_{\langle \overline{a}_{-1},\overline{a}_0,\overline{a}_1\rangle \to \overline{b}} (C_t(\overline{x}-1,\overline{a}_{-1}) \wedge C_t(\overline{x},\overline{a}_0) \wedge C_t(\overline{x}+1,\overline{a}_1)).$$

The disjunction is over the finite set of quadruples $(\overline{a}_{-1},\overline{a}_0,\overline{a}_1,\overline{b})$, such that the first three symbols lead to the fourth in one move of $M$.

This set of quadruples exactly represents $M$'s transition table.

We have already seen how to write $C_0$ with $k+1$ domain variables.

$M$ accepts its input iff it eventually reaches its accept state.

Let $\overline{\textbf{true}}$ code the appropriate accept symbol.

Thus, $M$ accepts its input iff eventually $C_t(\overline{0},\overline{\textbf{true}})$ holds.

# DSPACE[$n^k$] $\subseteq$ VAR[$k+1$] (Claim)

- The lemma will be proved once we show the following

**Claim**

There is a quantifier block $QB$ containing $k+1$ domain variables, such that the preceding equation may be rewritten as

$$C_{t+1}(\overline{x}, \overline{b}) = [QB]C_t(\overline{x}, \overline{b}).$$

- The proof is purely symbol manipulation.

  We first write quantifier blocks $\mathrm{QB}_+$ and $\mathrm{QB}_-$ whose job it is to replace $\overline{x}$ by $\overline{x}+1$ and $\overline{x}-1$ respectively.

  That is, for any formula $\varphi$, we have,

$$\varphi(\overline{x}+1) \equiv [\mathrm{QB}_+]\varphi(\overline{x}), \quad \varphi(\overline{x}-1) \equiv [\mathrm{QB}_-]\varphi(\overline{x}).$$

  These quantifier blocks can be written with $k+1$ domain variables.

# DSPACE[$n^k$] $\subseteq$ VAR[$k+1$] (Claim Cont'd)

- The idea is to add one to $\overline{x}$ by replacing $x_k$ with its successor, or, if $x_k = \max$, by replacing $x_k$ by 0 and $x_{k-1}$ by its successor, or, etc.

  We existentially quantify a tuple of boolean variables, $\overline{c}$, to guess for which $i$, $1 \leq i \leq k$, $x_i$ will be incremented.

  For $j > i$, it must be that $x_j = \max$ and $x'_j = 0$.

  The form of the quantifier block will be as follows,

$$(\exists \overline{c}.P)(\exists y.N_k)(\exists x_k.M_k)(\exists y.N_{k-1})(\exists x_{k-1}.M_{k-1})\cdots(\exists y.N_1)(\exists x_1.M_1).$$

  The quantifier-free conditions $P$, $N_i$ and $M_i$ are described next.

# DSPACE[$n^k$] $\subseteq$ VAR[$k+1$] (Claim Cont'd)

- The quantifier-free conditions $P$, $N_i$ and $M_i$ are as follows.

$$\begin{aligned} P &\equiv \bigvee_{i=1}^{k}(\overline{c} = i \wedge x_i \neq \max \wedge x_{i+1} = x_{i+2} = \cdots = x_k); \\ N_i &\equiv (i < \overline{c} \wedge y = x_k) \vee (i = \overline{c} \wedge \mathrm{SUC}(x_k, y)) \\ &\qquad \vee (i > \overline{c} \wedge y = \max); \\ M_i &\equiv x_i = y. \end{aligned}$$

Thus, we have $\mathrm{QB}_+, \mathrm{QB}_-$ and, trivially, $\mathrm{QB}_0$.

Observe that the desired QB of the claim is a positive boolean combination of these three quantifier blocks.

It follows from one of the preceding lemmas that QB exists and has $k+1$ domain variables, as desired.

This completes the proof of the claim and, thus, of the lemma.

# VAR[$k+1$] $\subseteq$ ITER[arity $k$]

**Lemma**

VAR[$k+1$] $\subseteq$ ITER[arity $k$].

- Here we have a quantifier block of the form

$$\text{QB} = [(Q_1 x_{i_1}.M_1)(B_1)\cdots(Q_r x_{i_r}.M_r)(B_r)],$$

where:

- $i_j \in \{1,\ldots,k+1\}$, for $j = 1,\ldots,r$;
- The $M_j$ are quantifier-free;
- The $B_j$ are blocks of boolean quantifiers over boolean variables $\{b_1,\ldots,b_c\}$.

We can convert the iteration of QB into an iterative definition of relations as follows.

Let $R_{s,t}$ be a set of $k$-ary relation symbols, for $1 \leq s \leq r$.

Let $t \in \{0,1\}^{\{1,\ldots,c\}}$.

Thus, $t$ specifies an assignment to all the boolean variables.

# VAR[$k+1$] $\subseteq$ ITER[arity $k$] (Cont'd)

- Intuitively, the iterative definition for the $R_{s,t}$'s is given as follows:

$$R_{s,t}(x_1, \ldots, \widehat{x_{i_s}}, \ldots, x_{k+1}) \equiv (Q_s x_{i_s}.M_s)(B_s) R_{s+1,t'}(x_1, \ldots, \widehat{x_{i_{s+1}}}, \ldots, x_{k+1}).$$

Here 1 is added to $s$ modulo $r$.

$R(x_1, \ldots, \widehat{x_i}, \ldots, x_{k+1})$ means that the variable $x_i$ is omitted.

In the formula the variable to be quantified next is safely omitted.

This is why arity $k$ suffices.

The above formula is a bit misleading in that we must write out boolean quantifier block $B_s$.

For example, the formula $(\forall b_j) R_{s+1,t'}(x)$ would be expanded to

$$R_{s+1,(t|b_j=0)}(\overline{x}) \wedge R_{s+1,(t|b_j=1)}(\overline{x}).$$

# ITER[arity $k$] $\subseteq$ DSPACE[$n^k$]

**Lemma**

ITER[arity $k$] $\subseteq$ DSPACE[$n^k$].

- This last inclusion is obvious because $O[n^k]$ bits suffice to record the current meaning of the bounded number of relations of arity $k$.

  Each bit of each relation in the next iteration may then be computed by evaluating a fixed first-order formula.

  This can be done in DSPACE[$\log n$].

  So, it can certainly be done in DSPACE[$n^k$].

  This completes the proof of the theorem.

Subsection 4

## Using Second-Order Logic to Capture PSPACE

# $\mathrm{SO}[t(n)]$

## Definition

A set $S \subseteq \mathrm{STRUC}[\tau]$ is a member of $\mathrm{SO}[t(n)]$ iff there exist:

- Quantifier-free formulas $M_i$, $0 \leq i \leq k$, from $\mathcal{L}(\tau)$;
- A tuple of constants $\overline{C}$;
- A quantifier block

$$\mathrm{QB} = [(Q_1 Z_1.M_1) \cdots (Q_k Z_k.M_k)],$$

such that, for all $\mathcal{A} \in \mathrm{STRUC}[\tau]$,

$$\mathcal{A} \in S \quad \text{iff} \quad \mathcal{A} \vDash ([\mathrm{QB}]^{t(\|\mathcal{A}\|)} M_0)(\overline{C}).$$

# SO$[t(n)]$ (Cont'd)

## Definition (Cont'd)

The only difference between SO$[t(n)]$ and FO$[t(n)]$ is that some of the variables $Z_i$ may be relation variables $S_i^{a_i}$, and the corresponding constant $C_i$ must be either an input or numeric relation, or a boolean.
Here:

- **false** denotes the empty relation;
- **true** denotes the full relations of appropriate arity.

Define

$$\text{SO}(\text{arity } a)[t(n)]$$

to be the restriction of SO$[t(n)]$ allowing quantification only of relations of arity $a$ or less.

# SO with Transitive Closure (Lemma)

## Lemma

Let SO(arity $a$)(TC) be the language SO(arity $a$) extended by the transitive closure operator. Any formula $\Phi \in$ SO(arity $a$)(TC) can be written in the normal form

$$\Phi \equiv (\mathrm{TC}_{A_1^a \ldots A_k^a, x_1 \ldots x_r, A_1'^a \ldots A_k'^a, x_1' \ldots x_r'} \alpha)(\overline{\mathbf{false}}, \overline{\mathbf{true}}),$$

where $\alpha$ is quantifier-free.

- The proof follows the strategy of the proof showing that every formula in FO(TC) is equivalent to a single application of transitive closure to a quantifier-free formula, $\varphi \equiv (\mathrm{TC}\alpha)(\overline{0}, \overline{\mathrm{max}})$.

# SO with Transitive Closure

**Proposition**

SO(arity $a$)(TC) $\subseteq$ SO(arity $a$)$[n^a]$.

- Let $\Phi \in$ SO(arity $a$)(TC).

  We may assume that $\Phi$ is in the form asserted by the lemma.

  We construct a second-order quantifier block similar to the first-order quantifier block, as done previously for transitive closure.

  Recall that, for transitive closure, $\varphi_{tc} \equiv [\mathrm{QB}_{tc}]R(x,y)$, where

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

  where,

$$\begin{aligned} M_1 &\equiv \neg(x = y \vee E(x,y)), \\ M_2 &\equiv (u = x \wedge v = z) \vee (u = z \wedge v = y), \\ M_3 &\equiv (x = u \wedge y = v). \end{aligned}$$

# SO with Transitive Closure (Cont'd)

- The difference here is that the expression $\overline{A} = \overline{A'}$ is no longer quantifier-free.

  We must replace it by

$$\forall z.((A_1(z) \leftrightarrow A_1'(z)) \wedge \cdots \wedge (A_k(z) \leftrightarrow A_k'(z))).$$

  We abbreviate this by

$$\forall z.(\overline{A}(z) \leftrightarrow \overline{A'}(z)).$$

# SO with Transitive Closure (Cont'd)

- The new quantifier block is as follows:

$$\mathrm{QB}_{SO(TC)} \equiv (\forall z.N_1)(\exists \overline{B})(\exists \overline{C}\ \overline{D})(\forall z)(\exists b.N_2)(\exists \overline{A}\ \overline{A'})(\forall z)(\exists b.N_3),$$

where

$$\begin{aligned} N_1 &\equiv \neg((\overline{A}(z) \leftrightarrow \overline{A'}(z)) \vee \alpha(\overline{A}, \overline{A'})), \\ N_2 &\equiv (b \wedge \overline{C}(z) \leftrightarrow \overline{A}(z) \wedge \overline{D}(z) \leftrightarrow \overline{B}(z)) \vee \\ &\quad (\neg b \wedge \overline{C}(z) \leftrightarrow \overline{B}(z) \wedge \overline{D}(z) \leftrightarrow \overline{A'}(z)), \\ N_3 &\equiv (\overline{A}(z) \leftrightarrow \overline{C}(z) \wedge \overline{A'} \leftrightarrow \overline{D}(z)). \end{aligned}$$

Finally, the length of the relevant $\alpha$-path can be at most $2^{kn^a}$.

It follows that,

$$\Phi \equiv [\mathrm{QB}_{SO(TC)}]^{kn^a}(\mathbf{false})[\mathbf{false}/A, \mathbf{true}/A'].$$

# Space and Second-Order Complexity

- The following theorem summarizes the relationship between:
  - Deterministic and nondeterministic space;
  - Second-order descriptive complexity.

**Theorem**

For $k = 1, 2, \ldots$:

1. $\mathrm{DSPACE}[n^k] = \mathrm{SO}(\text{arity } k)(\mathrm{DTC})$;
2. $\mathrm{NSPACE}[n^k] = \mathrm{SO}(\text{arity } k)(\mathrm{TC})$.

# Descriptive Characterizations of PSPACE

## Corollary

We have the following descriptive characterizations of PSPACE:

$$\begin{aligned} \text{PSPACE} &= \text{FO(PFP)} = \text{FO}[2^{n^{O(1)}}] \\ &= \text{SO(TC)} = \text{SO(DTC)} = \text{SO}[n^{O(1)}]. \end{aligned}$$

- From a parallel point of view, we have already seen that $\text{SO}[t(n)]$ is equal to $\text{CRAM-PROC}[t(n), 2^{n^{O(1)}}]$.
- This suggests that there is a striking tradeoff in parallel time versus hardware.

## Corollary

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