Chapter 3

First-Order Reductions

First-order reductions are simple translations that have very little computational power of their own. Thus it is surprising that the vast majority of natural complete problems remain complete via first-order reductions. We provide examples of such complete problems for many complexity classes. All the complexity classes and descriptive languages studied in this book are closed under first-order reductions. Once we express a complete problem for some complexity class $\mathcal{C}$ in a language $\mathcal{L}$, it will follow that $\mathcal{L}$ contains all queries computable in $\mathcal{C}$.

3.1 FO $\subseteq$ L

Recall that FO is the set of first-order definable boolean queries (Definition 1.26). It may be expected that the computational complexity of easy-to-describe queries is low. It was very surprising to us at first that descriptive classes like FO are identical to natural complexity classes.

We will see in Chapter 5 that FO is equal to the set of boolean queries computable in constant parallel time. The following theorem will get us started comparing descriptive versus machine characterizations of complexity. We show that first-order definable queries are all computable in logspace. We will see in Chapter 13 that this containment is strict.

Theorem 3.1 *The set of first-order boolean queries is contained in the set of boolean queries computable in deterministic logspace:* FO $\subseteq$ L.

Proof Let $\sigma = \langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s \rangle$. Let $\varphi \in \mathcal{L}(\sigma)$ determine a first-order boolean query, $I_\varphi : \text{STRUC}[\sigma] \to \{0,1\}$,

$$\varphi \equiv (\exists x_1)(\forall x_2) \ldots (Q_k x_k)\alpha(\bar{x})$$

where α is quantifier-free. We must construct a logspace Turing machine M such that for all $\mathcal{A} \in \text{STRUC}[\sigma]$, $\mathcal{A}$ satisfies φ iff M accepts the binary encoding of $\mathcal{A}$. In symbols,

$$\mathcal{A} \models \varphi \qquad \Leftrightarrow \qquad M(\text{bin}(\mathcal{A}))\downarrow \tag{3.2}$$

We construct the logspace Turing machine M inductively on k, the number of quantifiers occurring in φ. If $k = 0$, then $\varphi = \alpha$ is a quantifier-free sentence. Thus, α is a fixed, finite boolean combination of atomic formulas. The atomic formulas are either occurrences of input relations $R_i(p_1, \ldots, p_{a_i})$ or numeric relations $p_1 = p_2$, $p_1 \leq p_2$, or BIT(p_1, p_2), and the p_i's are members of $\{c_1, \ldots, c_s, 0, 1, \mathit{max}\}$. Once we know that M can determine, on input $\mathcal{A}$, whether $\mathcal{A}$ satisfies each of these atomic formulas, M can then determine whether $\mathcal{A} \models \alpha$, by performing the fixed, finite boolean combination using its finite control.

The reader should convince herself that a logspace machine that knows its input is of the form bin$(\mathcal{A})$, for some $\mathcal{A} \in \text{STRUC}[\sigma]$, can calculate the values n and $\lceil \log n \rceil$. Then, by counting, the machine can go to the appropriate constants and copy the p_i's that it needs onto its worktape. To calculate one of the input predicates, M can just look up the appropriate bit of its input.

For example, to calculate $R_3(c_2, \mathit{max}, c_1)$, M first copies the values $c_2, n-1, c_1$ to its worktape. Next it moves its read head to location $n^{a_1} + n^{a_2} + 1$, which is the beginning of the array encoding R_3. Finally, it moves its read head $n^2 \cdot c_2 + n \cdot (n-1) + c_1$ spaces to the right. The bit now being read is "1" iff $\mathcal{A} \models R_3(c_2, \mathit{max}, c_1)$.

It is easy to see that a logspace Turing machine may test the numeric predicates. This completes the construction of M in the base case.

Inductively, assume that all first-order queries with $k-1$ quantifiers are logspace computable. Let

$$\psi(x_1) \quad = \quad (\forall x_2) \ldots (Q_k x_k)\alpha(\bar{x}) \; .$$

Let M_0 be the logspace Turing machine that computes the query $\psi(c)$. Note that c is a new constant symbol substituted for the free variable x_1. To compute the query $\varphi \equiv (\exists x_1)(\psi(x_1))$ we build the logspace machine that cycles through all possible

values of x_1, substitutes each of these for c, and runs M_0. If any of these lead M_0 to accept, then we accept, else we reject. Note that the extra space needed is just $\log n$ bits to store the possible values of x_1. Simulating a universal quantifier is similar. □

3.2 Dual of a First-Order Query

A first-order query I from STRUC$[\sigma]$ to STRUC$[\tau]$ maps any $\mathcal{A} \in$ STRUC$[\sigma]$ to $I(\mathcal{A}) \in$ STRUC$[\tau]$. It does this by defining the relations of $I(\mathcal{A})$ via first-order formulas. In a similar way, I has a natural dual $\widehat{I}$, which translates any formula in $\mathcal{L}(\tau)$ to a formula in $\mathcal{L}(\sigma)$.

In this section we define and characterize this dual mapping. The dual is useful in showing that relevant languages and complexity classes are closed under first-order reductions.

Let I be a k-ary first-order query. For each formula $\varphi \in \mathcal{L}(\tau)$, the formula $\widehat{I}(\varphi) \in \mathcal{L}(\sigma)$ is constructed as follows: Replace each variable by a k-tuple of variables. Replace each symbol of τ by its definition in I. It follows that the length of $\widehat{I}(\varphi)$ is linear in the length of φ. In the following, we give the details.

Definition 3.3 (Dual of I) Let $I = \lambda_{x_1 \ldots x_d}\langle \varphi_0, \ldots, \psi_s \rangle$ be a k-ary first-order query from STRUC$[\sigma]$ to STRUC$[\tau]$. Then I also defines a dual map, which we call $\widehat{I} : \mathcal{L}(\tau) \to \mathcal{L}(\sigma)$, as follows:

Let $\tau = \langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s \rangle$. For $\varphi \in \mathcal{L}(\tau)$, $\widehat{I}(\varphi)$ is the result of replacing all relation and constant symbols in φ by the corresponding formulas in I, using a map f_I defined as follows:

- Each variable is mapped to a k-tuple of variables: $f_I(v) = v^1, \ldots, v^k$
- Input relations are replaced by their corresponding formulas:

$$f_I(R_i(v_1, \ldots, v_{a_i})) = \varphi_i(f_I(v_1), \ldots, f_I(v_{a_i}))$$

- Constant c_i is replaced by a special k-tuple of variables[1]:

$$f_I(c_i) = z_i^1, \ldots, z_i^k$$

[1]For many applications, each constant from τ may be replaced by a corresponding k-tuple of constants from σ.

- Quantifiers are replaced by restricted quantifiers:

$$f_I(\exists v) = (\exists f_I(v) \,.\, \varphi_0(f_I(v)))$$

- The equality relation and the other numeric relations are replaced by their appropriate formulas as in Exercise 1.33.

- Second-order quantifiers — which we will need in Chapter 7 — have the arities of the relations being quantified multiplied by k:

$$f_I(\exists R^a) = (\exists R^{ka})$$

- On boolean connectives, f_I is the identity.

The only thing to add is that the variables $z_i^1, \ldots, z_i^k$ corresponding to the constant symbol c_i must be quantified before they are used. It does not matter where these quantifiers go because the values are uniquely defined. Typically, we can place these quantifiers at the beginning of a first-order formula. (For a second-order formula, they would be placed just after the second-order quantifiers.)

Thus, the mapping $\widehat{I}$ is defined as follows, for $\theta \in \mathcal{L}(\tau)$,

$$\widehat{I}(\theta) \;=\; (\exists z_1^1 \ldots z_1^k \,.\, \psi_1(z_1^1 \ldots z_1^k)) \cdots (\exists z_s^1 \ldots z_s^k \,.\, \psi_s(z_s^1 \ldots z_s^k))(f_I(\theta)) \qquad \square$$

Exercise 3.4 To get the idea of what the dual mapping does, consider the query I_{PM} from Example 2.12. Here is one sample value of the map $\widehat{I}_{PM}$:

$$\begin{aligned} \widehat{I}_{PM}(A(c)) &\equiv (\exists z_1 z_2 \,.\, z_1 = 0 \wedge z_2 = \mathit{max})(z_2 = \mathit{max} \wedge S(z_1)) \\ &\equiv S(0) \end{aligned}$$

Compute the value of $\widehat{I}_{PM}$ on the following,

1. $(\forall v)(A(v) \leftrightarrow B(v))$
2. $A(\mathit{max})$
3. $A(0)$ $\square$

It follows immediately from Definition 3.3 that:

Proposition 3.5 *Let σ, τ, and I be as in Definition 3.3. Then for all sentences $\theta \in \mathcal{L}(\tau)$ and all structures $\mathcal{A} \in \text{STRUC}[\sigma]$,*

$$\mathcal{A} \models \widehat{I}(\theta) \qquad \Leftrightarrow \qquad I(\mathcal{A}) \models \theta$$

Remark 3.6 *Proposition 3.5 goes through for formulas with free variables as well. In this case, I must behave appropriately on interpretations of variables. That is, $I(\mathcal{A}, i) = (I(\mathcal{A}), i')$, where $i'(x)$ is defined iff all of $i(x^1), \ldots, i(x^k)$ are defined. In this case, $i'(x) = \langle i(x^1), \ldots, i(x^k)\rangle$.*

In the following exercise you are asked to show that structures of any vocabulary σ may be transformed to graphs via a first-order, invertible query. It then follows from Proposition 3.5 that every formula in $\mathcal{L}(\sigma)$ may be translated into the language of graphs. This is true even without the ordering relation. Exercise 2.3 shows the same thing about binary strings, i.e., that everything can be coded in a first-order way as a binary string; but in the case of strings, ordering and arithmetic are required.

Exercise 3.7 (Everything is a Graph) Let σ be any vocabulary, and let $\tau_e = \langle E^2\rangle$ be the vocabulary with one binary relation symbol, i.e., the vocabulary of graphs with no specified points. In this exercise, you will show that every structure may be thought of as a graph.

Show that there exist first-order queries $I_\sigma : \text{STRUC}[\sigma] \to \text{STRUC}[\tau_e]$ and $I_\sigma^{-1} : \text{STRUC}[\tau_e] \to \text{STRUC}[\sigma]$ with the following property,

$$\text{for all } \mathcal{A} \in \text{STRUC}[\sigma], \qquad I_\sigma^{-1}(I_\sigma(\mathcal{A})) \cong \mathcal{A} \tag{3.8}$$

[Hint: to build the graph $I_\sigma(\mathcal{A})$, you can construct "gadgets", i.e., small recognizable graphs to label different sorts of vertices, e.g., those corresponding to elements of $|\mathcal{A}|$, those corresponding to tuples from each relation $R_i^{\mathcal{A}}$, etc.] Note that this exercise does not require ordering, which is why Equation (3.8) says only that the two objects are isomorphic. If we include ordering, we can require that equality holds. □

If A and B are boolean queries and A is first-order reducible to B ($A \leq_{\text{fo}} B$), then intuitively the complexity of A is not greater than the complexity of B. The following definition makes this intuitive idea explicit.

Definition 3.9 (Closure Under First-Order Reductions) A set of boolean queries $\mathcal{S}$ is *closed under first-order reductions* iff whenever there are boolean queries A

and B such that $B \in \mathcal{S}$ and $A \leq_{\text{fo}} B$, we have that $A \in \mathcal{S}$. We say that a language $\mathcal{L}$ is closed under first-order reductions iff the set of boolean queries definable in $\mathcal{L}$ is closed. □

The following proposition follows immediately from Theorem 3.1.

Proposition 3.10 *Let $\mathcal{S}$ be any set of boolean queries that is closed under logspace reductions. Then $\mathcal{S}$ is also closed under first-order reductions.*

The next proposition cannot be proved immediately. It is a global exercise. It is striking that with the exception of the dynamic-complexity classes, all complexity classes we discuss in the book are closed under first-order reductions. Every time a new language or complexity class is introduced, the reader should check that it is closed under first-order reductions. For languages, we mean that the set of boolean queries definable in that languages is closed under first-order reductions. This is immediate if the language is closed under quantification and boolean operations. Most languages we consider are so closed. Some languages such as SO$\exists$ — see Theorem 7.8 — do not seem to be closed under negation. However, they still allow full use of first-order logic at their bottom levels and are thus closed under first-order reductions.

Meta-Proposition 3.11 *With the exception of the dynamic complexity classes defined in Chapter 14, all the complexity classes $\mathcal{C}$ that we discuss in this book are closed under first-order reductions. All the languages $\mathcal{L}$ that we discuss in this book are closed under first-order reductions.*

Framework 3.12 Here is a method for proving this proposition whenever a new complexity class or logical language is encountered. For complexity classes we can usually use Proposition 3.10 as most complexity classes are closed under logspace reductions.

For languages, let $A \leq_{\text{fo}} B$ be two boolean queries, where B is expressible as the formula φ_B in language $\mathcal{L}$. Let I_{AB} be the first-order reduction from A to B. We know that for all structures $\mathcal{S}$,

$$\mathcal{S} \in A \qquad \Leftrightarrow \qquad I_{AB}(\mathcal{S}) \in B$$

It follows from Proposition 3.5 that

$$\mathcal{S} \in A \qquad \Leftrightarrow \qquad \mathcal{S} \models \widehat{I}_{AB}(\varphi_B)$$

So if $\widehat{I}_{AB}(\varphi_B)$ is in $\mathcal{L}$ then the proof is complete. Since the definition of $\widehat{I}(\varphi)$ (Definition 3.3) is a simple substitution that doesn't change the structure of φ very much, we will find that for the languages we consider $\widehat{I}_{AB}(\varphi_B)$ will be in $\mathcal{L}$ as desired. □

First-order reductions are simple and natural reductions. It is very surprising that they seem to suffice in almost all complexity theoretic settings. We will see that "natural" problems that are complete via polynomial-time reductions for some complexity class tend to remain complete via first-order reductions.

Suppose that we know that a boolean query A is complete via first-order reductions for a complexity class $\mathcal{C}$. Suppose further that A is expressible in a language $\mathcal{L}$ which is closed under first-order reductions. It follows immediately that $\mathcal{L}$ expresses everything in $\mathcal{C}$.

Suppose that $\mathcal{L}$ is a set of boolean queries describable in some language and that $\mathcal{C}$ is a complexity class, that is, a set of boolean queries computable in some complexity bound. In the sequel our paradigm for proving that $\mathcal{L} = \mathcal{C}$ will be the following four steps:

1. Show that $\mathcal{L} \subseteq \mathcal{C}$ by producing for each formula φ from the language an algorithm in $\mathcal{C}$ that computes the boolean query,
$$\text{MOD}[\varphi] \quad = \quad \{\mathcal{A} \mid \mathcal{A} \models \varphi\}$$
2. Produce a boolean query T that is complete for $\mathcal{C}$ via first-order reductions.
3. Show that $\mathcal{L}$ is closed under first-order reductions.
4. Express T in the language, thus showing that $T \in \mathcal{L}$.

A typical example is in a proof of Theorem 7.8, which says that NP = SO∃. We can show: (1) Each SO∃ formula can be checked by an NP machine; (2) The problem SAT is complete for NP via $\leq_{\text{fo}}$; (3) SO∃ is closed under first-order reductions; and finally, (4) SAT is expressible in SO∃. (Actually, in Chapter 7 we present a different proof for Theorem 7.8 that provides more information.)

In the remainder of this chapter, we give several examples of first-order reductions as we produce natural complete problems for the complexity classes L, NL, and P. The proofs encode Turing machine computations using first-order formulas. The proofs are quite intricate. For this reason, it would be fine to skim the remainder of this chapter on first reading. On the other hand, since many of the results on capturing complexity classes by logics depend on this material, at some point this material should be read.

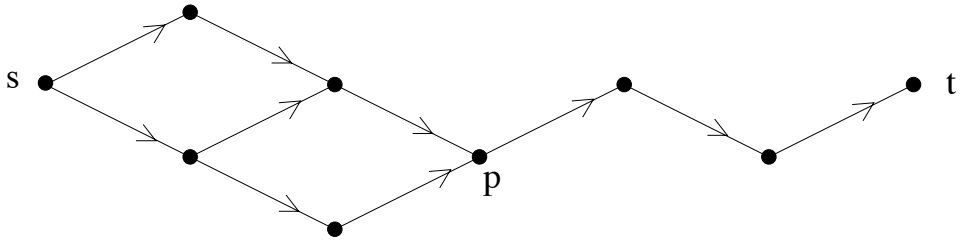

Figure 3.14: A graph that is in REACH but not REACH_d

3.3 Complete problems for L and NL

Natural complete problems for L and NL are the REACH_d and REACH problems.

Definition 3.13 Define REACH to be the set of directed graphs G such that there is a path in G from s to t. Define REACH_d to be the subset of REACH such that the path from s to t is deterministic. This means that for each edge (u, x) on the path, this is the unique edge in G leaving u. See Figure 3.14 for a directed graph that is in REACH but not REACH_d. Note that there is a directed path in this figure from p to t. Also, define REACH_u — the undirected graph reachability problem — to be the restriction of REACH to undirected graphs,

$$\text{REACH}_u \quad = \quad \{G \in \text{REACH} \mid G \models (\forall xy)(E(x,y) \rightarrow E(y,x))\} \qquad \square$$

The following NSPACE[$\log n$] algorithm recognizes REACH. Note that the space used is just the $O(\log n)$ bits needed to name the two vertices a and b.

Algorithm 3.15 Recognizing REACH in NL

1. $b := s$
2. **while** $(b \neq t)$ **do** $\{$
3. $\quad a := b$
4. $\quad$ nondeterministically choose new b
5. $\quad$ **if** $(\neg E(a,b))$ **then** reject $\}$
6. accept

Theorem 3.16 REACH *is complete for* NL *via first-order reductions.*

Proof Let $S \subseteq \text{STRUC}[\sigma]$ be a boolean query in NL. Let N be the nondeterministic logspace Turing machine that accepts S. We construct a first-order reduction $I : \text{STRUC}[\sigma] \to \text{STRUC}[\tau_g]$ such that for all $\mathcal{A} \in \text{STRUC}[\sigma]$,

$$N(\text{bin}(\mathcal{A}))\downarrow \quad \Leftrightarrow \quad I(\mathcal{A}) \in \text{REACH} \tag{3.17}$$

Let c be such that N uses at most $c \log n$ bits of worktape for inputs $\text{bin}(\mathcal{A})$, with $n = \|\mathcal{A}\|$. Let $\sigma = \langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s \rangle$ and let $a = \max\{a_i \mid 1 \leq i \leq r\}$. Let $k = 1 + a + c$. Consider a run of N on input $\text{bin}(\mathcal{A})$. We code an instantaneous description (ID) of N's computation as a k-tuple of variables:

$$\text{ID} \quad = \quad (p, r_1, \ldots r_a, w_1, \ldots, w_c)$$

The idea is that variables $r_1, \ldots, r_a$ encode where in one of the input relations the read head of N is looking. If for example it is looking at relation R_i, then,

$$N\text{'s read head is looking at a "1"} \quad \Leftrightarrow \quad \mathcal{A} \models R_i(r_1, \ldots, r_{a_i}) \tag{3.18}$$

Variables $w_1, \ldots, w_c$ encode the contents of N's work tape. Remember that each variable represents an element of $\mathcal{A}$'s n-element universe, so it corresponds to a $\log n$-bit number. We are assuming the presence of the numerical relations $\leq$ and BIT. Of these, $\leq$ is necessary (see Proposition 6.14), but BIT is merely convenient (see Proposition 9.16). Finally, we need $O(\log \log n)$ bits of further information to encode: (1) the state of N, (2) which input relation or constant symbol the read head is currently scanning, and (3) the position of the work head. We assume that n is sufficiently large that all of this information can be encoded into a single variable, p.

Now we start to build the desired k-ary first-order query I and show that it satisfies Equation (3.17). I will be constructed as follows:

$$I \quad = \quad \lambda_{\text{ID},\text{ID}'} \langle \textbf{true}, \varphi_N, \alpha, \omega \rangle$$

where

1. The universe relation being "**true**" indicates that for any $\mathcal{A} \in \text{STRUC}[\sigma]$, the universe of $I(\mathcal{A})$ consists of *all* k-tuples from the universe of $\mathcal{A}$, $|I(\mathcal{A})| = |\mathcal{A}|^k$.
2. $\mathcal{A} \models \varphi_N(\text{ID}, \text{ID}')$ iff (ID, ID') is a valid move of N on input $\text{bin}(\mathcal{A})$,
3. $\mathcal{A} \models \alpha(\text{ID}_i)$ iff ID_i is the unique initial ID of N, for inputs of size $\|\mathcal{A}\|$, and,

4. $\mathcal{A} \models \omega(\mathrm{ID}_f)$ iff ID_f is the unique accept ID of N for inputs of size $\|\mathcal{A}\|$.

Formulas α and ω are the following,

$$\begin{array}{lcl} \alpha(x_1, \ldots, x_k) & \equiv & x_1 = x_2 = \ldots = x_k = 0 \\ \omega(x_1, \ldots, x_k) & \equiv & x_1 = x_2 = \ldots = x_k = \mathit{max} \end{array} \tag{3.19}$$

Formula φ_N is not hard, but it is more tedious. It is essentially a disjunction over N's finite transition table.

A typical entry in the transition table is $(\langle q, b, w\rangle, \langle q', i_d, w', w_d\rangle)$. This says that in state q, looking at bit b with the input head and bit w with the work head, N may go to state q', move its input head one step in direction i_d, write bit w' on its work tape and move its work head one step in direction w_d. The corresponding disjunct in φ_N must decode the old state from variable p and must decode from p which input relation is being read. Say it is R_i. Then the bit b is "1" iff $R_i(r_1, \ldots, r_{a_i})$ holds. Similarly, we must extract from p the segment j of the work tape that is currently being scanned together with the position s on that worktape. Thus, bit w is "1" iff $\mathrm{BIT}(w_j, s)$ holds.

With these details completed, it now follows that for any $\mathcal{A} \in \mathrm{STRUC}[\sigma]$, $I(\mathcal{A})$ is the computation graph of N on input $\mathrm{bin}(\mathcal{A})$. It follows that N accepts $\mathrm{bin}(\mathcal{A})$ iff there is a path in $I(\mathcal{A})$ from s to t, i.e., Equation (3.17) holds. □

Exercise 3.20 There are several gaps left in the proof of Theorem 3.16 that the reader should now fill in.

1. Using numeric relation BIT, write first-order formulas to uniquely identify elements $l_1 = \lceil \log n \rceil$ and $l_2 = \lceil \log \log n \rceil$ of the universe.
2. Show that since the coding is somewhat arbitrary, we may use Equation (3.19) as our definitions of α and ω.
3. Assume that the first l_2 bits of p encode the work head's position, s. Write a formula to uniquely identify element s.
4. Do the same problem as (3) but this time assume that the bits of s are encoded in the last l_2 bits of p. In order to do this you will need addition, which is available (Theorem 1.17). □

We next show that REACH_d is complete for L via first-order reductions. We first must show that REACH_d is in L. A modification of Algorithm 3.15 recognizes REACH_d in logspace. Since a deterministic path has at most one edge leaving each vertex, nondeterminism is no longer needed. We add a counter to detect cycles:

Algorithm 3.21 Recognizing REACH_d in L

1. $b := s;\ i := 0;\ n := \|G\|$
2. **while** $b \neq t \wedge i < n \wedge (\exists!a)(E(b,a))$ **do** $\{$
3. $\quad b :=$ the unique a for which $E(b,a)$
4. $\quad i := i + 1\ \}$
5. **if** $b = t$ **then** accept **else** reject

The definition of REACH_d was made just so that the following theorem would be true:

Theorem 3.22 REACH_d *is complete for* L *via first-order reductions.*

Proof This proof is similar to the proof of Theorem 3.16. In fact, we copy the whole construction with $S \subseteq \mathrm{STRUC}[\sigma]$ an arbitrary boolean query from L. The only difference is that now N is a deterministic logspace Turing machine that computes S. Since N is deterministic, for any $\mathcal{A} \in \mathrm{STRUC}[\sigma]$, the graph $I(\mathcal{A})$ has at most one edge leaving any vertex. It follows that $I(\mathcal{A})$ is in REACH iff it is in REACH_d. Thus,

$$N(\mathrm{bin}(\mathcal{A}))\downarrow \qquad \Leftrightarrow \qquad I(\mathcal{A}) \in \mathrm{REACH}_d \qquad \square$$

3.4 Complete Problems for P

Alternation provides a nice way to characterize P, namely, $\mathrm{P} = \mathrm{ASPACE}[\log n]$ (Theorem 2.25). This leads to a natural analogue of the reachability problem that is complete problem for P.

Definition 3.23 Let an *alternating graph* $G = (V, E, A, s, t)$ be a directed graph whose vertices are labeled universal or existential. $A \subseteq V$ is the set of universal vertices. Let $\tau_{ag} = \langle E^2, A^1, s, t\rangle$ be the vocabulary of alternating graphs.

Alternating graphs have a different notion of accessibility. Let $P_a^G(x,y)$ be the smallest relation on vertices of G such that:

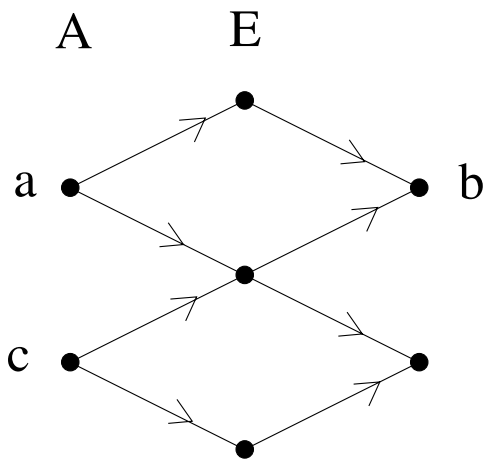

Figure 3.24: An alternating graph with two universal nodes: a, c.

1. $P_a^G(x, x)$
2. If x is existential and $P_a^G(z, y)$ holds for some edge (x, z) then $P_a^G(x, y)$.
3. If x is universal, there is at least one edge leaving x, and $P_a^G(z, y)$ holds for all edges (x, z) then $P_a^G(x, y)$.

See Figure 3.24 where $P_a^G(a, b)$ holds but $P_a^G(c, b)$ does not. Let

$$\text{REACH}_a \quad = \quad \{G \mid P_a^G(s, t)\} \qquad \square$$

Observe that the following marking algorithm computes REACH_a in linear time.

Algorithm 3.25 Recognizing REACH_a in linear time on a RAM.

1. make QUEUE empty; mark(t); insert t into QUEUE
2. **while** QUEUE not empty **do** {
3. remove first element, x, from QUEUE
4. **for** each unmarked vertex y such that $E(y, x)$ **do** {
5. delete edge (y, x)

6. **if** y is existential or y has no outgoing edges
7. **then** mark(y); insert y into QUEUE } }
8. **if** s is marked then **accept else reject**

Not surprisingly we have,

Theorem 3.26 REACH_a *is complete for* P *via first-order reductions.*

Proof This proof is similar to the proof of Theorem 3.16. Let $S \subseteq \mathrm{STRUC}[\sigma]$ be an arbitrary boolean query. Assume that $S \in$ P and let T be the alternating, logspace Turing machine that computes S. We construct a first-order reduction $I_a : \mathrm{STRUC}[\sigma] \to \mathrm{STRUC}[\tau_{ag}]$ such that for all $\mathcal{A} \in \mathrm{STRUC}[\sigma]$,

$$T(\mathrm{bin}(\mathcal{A}))\downarrow \quad \Leftrightarrow \quad I_a(\mathcal{A}) \in \mathrm{REACH}_a \tag{3.27}$$

Indeed, the only difference between I, the query from the proof of Theorem 3.16, and I_a is that I_a must also describe the relation A that identifies the universal states of T. Assume for simplicity that the universal states are exactly the odd-numbered states. Assume further that the variable p in an ID encodes its state in its low-order bits. Thus the state of an ID is universal iff the corresponding p is odd. This occurs iff $\mathrm{BIT}(p, 0)$ holds. Thus, let

$$\psi_A \;=\; \mathrm{BIT}(p, 0), \qquad I \;=\; \lambda_{\mathrm{ID},\mathrm{ID}'}\langle \mathbf{true}, \varphi_T, \psi_A, \alpha, \omega\rangle$$

where φ_T, α, and ω are defined exactly as in the proof of Theorem 3.16.

It follows that,

$$T(\mathrm{bin}(\mathcal{A}))\downarrow \quad \Leftrightarrow \quad I_a(\mathcal{A}) \in \mathrm{REACH}_a \qquad \square$$

Exercise 3.28 Recall the circuit value problem (CVP) and the monotone circuit value problem (MCVP) from Definition 2.27.

1. Produce a first-order reduction from REACH_a to MCVP.
2. Conclude that MCVP is complete for P via first-order reductions.
3. Conclude that CVP is also complete via first-order reductions. Be slightly careful because it is certainly not true that for any two boolean queries S, T, if S is complete for $\mathcal{C}$ and $T \in \mathcal{C}$ and $S \subseteq T$ then T is complete for $\mathcal{C}$. $\square$

Historical Notes and Suggestions for Further Reading

Savitch proved that REACH is complete for NL via logspace reductions, [Sav73]. Hartmanis, Immerman, and Mahaney showed that REACH_d is complete for L via one-way logspace reductions, [HIM78]. The completeness of these problems via first-order reductions was proved in [I83]. In these references, REACH was called "GAP" and REACH_d was called "1GAP".

The complexity of REACH_u is also a rich subject. Aleliunas, Karp, Lipton, Lovász, and Rackoff, proved that taking a random walk in an undirected graph will — with very high probability — quickly reach all reachable vertices [AKL79]. It follows that boolean query REACH_u is computable in "random logspace". Lewis and Papadimitriou define a restriction of nondeterministic space called "symmetric space" and prove that REACH_u is complete via logspace reductions for symmetric logspace [LP82]. Reingold proved in [Rei05] the breakthrough result that REACH¡sub¿u¡/sub¿ is in L, and thus that symmetric logspace is equal to L."

REACH_a was shown to be complete for P via logspace reductions in [I80] and via first-order reductions (and in fact quantifier-free projections) in [I83].

Exercise 3.28 shows that the monotone circuit value problem is complete for P via first-order reductions. The completeness of CVP for P via logspace reductions was first shown by Ladner in [L75]. The completeness of MCVP via logspace reductions was originally shown by Goldschlager in [Go77].