

§14. Logic and Complexity. We have seen in various parts of this course that there is a close connection between logic and computation. This is hardly surprising since both areas formalize certain closely connected processes; proof in the case of logic, algorithm in the case of computation. In this final note we describe an even closer connection by considering clocked computation and in particular the connection between logic and the classes P and NP. In order to simplify matters we will focus on problems in graph theory; we will not give complete details as this would take more time than is available. For a very good discussion of the issues see C. H. Papadimitriou, *Computational Complexity*, Addison-Wesley.

§14.1. First order logic. Each type of logic consists of formal formulae and rules for manipulating them. The truth value of formulae is determined by an interpretation in an agreed structure (more accurately class of structures). Thus if we want to talk about graphs we might consider

$$\phi = \forall x \neg E(x, x) \wedge \forall x \forall y (E(x, y) \Rightarrow E(y, x)). \quad (\dagger)$$

We can interpret this as a statement about graphs as follows:

1. the variables (x and y) range over vertices,
2. the symbol E stands for the edge relation of the graph³²; $E(x, y)$ is true if and only if there is an edge in the graph from the vertex that x denotes to the vertex that y denotes (note that this is a directed edge).

(The logic is said to be *first order* because variables are allowed to range only over individual elements of the domain of interpretation and not over subsets or relations.) Apart from the relation symbol E we also allow the use of equality (denoted in the usual infix way as $x = y$)³³. Given a set Φ of formulae of first order logic and an interpretation we say that a structure (graph for us) is a *model* of Φ if each formula of Φ is true under the intended interpretation. A *sentence* is a formula in which each variable is bound by a quantifier; sentences have the distinguishing feature that we do not need to give an explicit interpretation of individual variables. For example a graph G is a model for the sentence ϕ if and only if the graph is undirected and there are no loops (edges joining a vertex to itself). Figure 7 illustrates the idea. Given a formula (not necessarily a sentence) ϕ with an interpretation we have the decision problem ϕ -GRAPHS:

³² E is not itself the relation but a symbol for it; after all we want to be able to interpret the same formula for many different graphs. You might like to compare this with the notion of formal parameters of a procedure in a programming language.

³³We can allow other relation symbols as well; the interpretation must provide the actual relation on graphs that each one denotes (the relation symbols are not variables). However we will stick to the simple situation described in the main text.

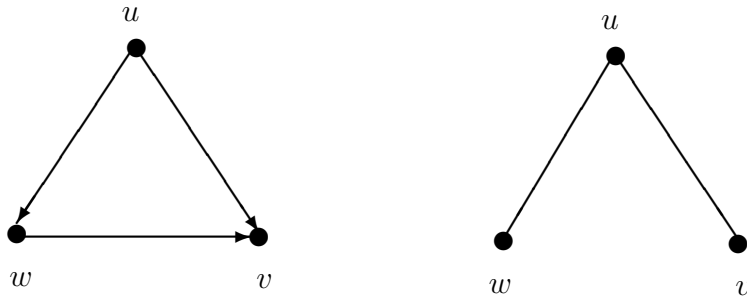


Figure 7: The first graph does not model ϕ in (†) but the second does.

INSTANCE: A finite graph G .

QUESTION: Is G a model of ϕ ? (i.e., does ϕ express a true property of G ?)

This classifies what we can say about finite graphs using first order logic.

THEOREM 14.1 *For any formula ϕ the problem ϕ -GRAPHS is in P .*

PROOF. The claim is that having fixed ϕ , we then have an algorithm that for each finite graph G : (i) decides if ϕ is true in G and (ii) runs in time that is bounded by a polynomial in the size of G (which we take to be the number of vertices).

We prove the claim by induction on the size of ϕ . The base case is that ϕ is of the form $E(x, y)$ or $E(x, x)$ or $x = y$ or $x = x$ where we have an interpretation for x, y as vertices of the graph. So this case amounts to checking that there is an edge as described or that two vertices are equal (of course $x = x$ is automatically true).

The induction step proceeds by examining the structure of ϕ . If $\phi = \neg\psi$ then we simply work with ψ and invert the decision (the runtime is that for ψ , a polynomial by induction, plus the time taken to invert the decision which is just a small constant). If $\phi = \psi_1 \vee \psi_2$ then we decide the question for ψ_1 . If this turns out to be true we can finish, otherwise we move on to ψ_2 . Clearly the runtime is at most that for ψ_1 plus that for ψ_2 and a small constant overhead. The situation is similar for the case $\phi = \psi_1 \wedge \psi_2$.

There are only two remaining cases³⁴, either $\phi = \forall x \psi$ or $\phi = \exists x \psi$. Consider the first case. We cycle through all the vertices v of G and in each case replace the free occurrences of x in ϕ by v . In order to understand this point suppose that $\phi = \forall x (\exists y E(x, y) \wedge \exists x \exists y E(x, y))$. Then at the vertex v we consider the formula

³⁴In fact we can assume there is only one if we note that $\exists x \psi$ can be avoided in formulae since it is equivalent to $\neg \forall x \neg \psi$.

$\exists y E(v, y) \wedge \exists x \exists y E(x, y)$; the second x is *not* replaced by v (the scope of a variable in a formula is similar to the case of variables in a programming language with lexical scoping). Now the graph models ϕ if and only if it models each of the formulae obtained by substitution. Furthermore each subproblem can be solved in polynomial time (by induction) and our runtime is at most n (the number of vertices) times the worst runtime for each of the subproblems. The new runtime is still a polynomial, but with higher degree. The case $\phi = \exists x \psi$ is similar. \square

It might seem that we have overlooked at least one case in the preceding proof, e.g., we never considered implication. In fact there are no missing cases since all other constructs can be expressed in terms of the ones considered in the proof, e.g., $A \Rightarrow B$ is equivalent to $\neg A \vee B$.

EXERCISE Consider the formula

$$\phi = \forall x \neg E(x, x) \wedge \exists x \exists y \exists z (E(x, y) \wedge E(y, z) \wedge E(z, y)).$$

Explain what property of (directed) graphs is expressed by ϕ and use Theorem 14.1 to develop a polynomial time algorithm. What is the runtime?

The preceding Theorem is a good, if rather simple, start to linking logic with complexity (the study of resource bounded computation). We might be tempted to conjecture that a property of graphs is in P if and only if it can be expressed in first order logic, i.e., it is ϕ -GRAPHS for some ϕ . But this is not the case; let REACHABILITY be

INSTANCE: A directed graph G and two vertices u, v of G .

QUESTION: Is there a (directed) path from u to v in G ?

It is an easy matter to show that this is in P [make sure you understand why]. If it is indeed the case that first order logic ‘captures’ P (so far as graphs are concerned) then there must be a first order formula $\phi(x, y)$ with two free variables x, y such that for a given graph G and vertices u, v there is a path from u to v if and only if $\phi(u, v)$ is true. Before we can prove that there is no such formula we need a standard result from the model theory of first order logic.

THEOREM (Löwenheim-Skolem Theorem) *If a sentence ϕ has models of arbitrarily large size then it has an infinite model.*

The proof of this result is beyond the scope of this course.

THEOREM 14.2 REACHABILITY *is not expressible in first order logic.*

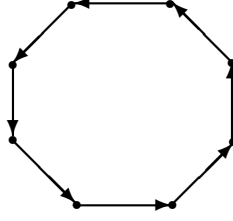


Figure 8: A cycle of size 8.

PROOF. We assume the contrary and derive a contradiction. We are thus assuming the existence of a first order formula ϕ as described above. We will use ϕ to construct a first order sentence ψ that expresses the notion of a directed cycle, i.e., a strongly connected graph such that each vertex has exactly one edge leaving it and exactly one edge entering it (see Figure 8). It is clear that there are cycles of any size we care to choose. This simple fact, taken together with the existence of ψ implies, by the Löwenheim-Skolem Theorem, that there are infinite cycles. However such cycles do not exist and this gives us the required contradiction.

The non-existence of infinite cycles is seen as follows: consider an arbitrary vertex of a cycle and call it v_0 . Since there is exactly one edge out of v_0 , it leads to a unique vertex v_1 . Arguing in this way we obtain a sequence $v_0, v_1, v_2 \dots$ of vertices of the graph such that (v_i, v_{i+1}) is an edge of the graph. Since we have assumed that the graph is a cycle, this sequence of vertices must account for all the vertices of the graph (a formal proof uses the fact that the graph is strongly connected and starting at any vertex there is a unique path out of it). So far, vertex v_0 has no edge entering it; but it is a requirement that there must be such an edge. There is therefore an s such that (v_s, v_0) is an edge. But then there cannot be a vertex v_{s+1} since the graph is strongly connected and we have no means of joining any of the vertices v_0, v_1, \dots, v_s to the supposed v_{s+1} . Thus every cycle is finite as claimed.

We proceed to express the three defining properties of a cycle.

1. The graph G is strongly connected:

$$\psi_1 = \forall x \forall y \phi(x, y).$$

2. Every vertex has exactly one edge leaving it:

$$\psi_2 = \forall x \exists y E(x, y) \wedge \forall x \forall y \forall z ((E(x, y) \wedge E(x, z)) \Rightarrow y = z).$$

3. Every vertex has exactly one edge entering it:

$$\psi_3 = \forall x \exists y E(y, x) \wedge \forall x \forall y \forall z ((E(y, x) \wedge E(z, x)) \Rightarrow y = z).$$

Now we can take $\psi = \psi_1 \wedge \psi_2 \wedge \psi_3$. Obviously the existence of ψ depends on the assumption that $\phi(x, y)$ exists. Since the existence of ψ leads to a contradiction we must abandon the assumption that ϕ exists. \square

This result has implications for Database applications. The Relational Model for databases in its pure form does not go beyond first order logic; but reachability is important (think of it as transitive closure which is relevant to queries of the form ‘starting from item A find all items related to it and all items related to them etc.’). In any realistic system this problem is circumvented by embedding queries in a programming language that gets us beyond first order logic.

§14.2. Second order logic. We saw in the preceding section that first order logic is too weak to express reachability and so has no chance of capturing P. The problem is that reachability asserts the existence of a *set* of vertices with a special property; in fact it asserts the existence of a relation. Moreover there is no *a priori* upper bound to the size of the set; if we know ahead of time that there are, e.g., 10 vertices in the set, we can assert their existence by starting with $\exists x_1 \exists x_2 \dots \exists x_{10}$.

We can boost the power of first order logic by introducing an extra relation symbol R in formulae and allowing formulae of the form $\exists R \phi$ where ϕ looks exactly like a first order formula if we treat R as just an unbound variable. The idea is that R stands for a relation on r -tuples of elements of the structure in which we intend to interpret formulae (r is agreed and fixed ahead of time for each formula, but we are allowed to change it between formulae). It might seem rather restrictive to allow only one relation, but it is not hard to see that a formula of the form $\exists R_1 \exists R_2 \dots \exists R_n \phi$ where each R_i is a relation symbol on r_i -tuples can be expressed by a formula of form $\exists R \phi$ (essentially we ‘paste’ the various relations into a single one).

The form of logic described here is called *existential second order logic*. Note that the negation of a formula of this logic is not necessarily expressible by a formula of the logic (we will return to this point below). Thus if we can express a property by an existential second order formula we cannot automatically deduce that the negation of the property can be so expressed; we must prove this separately (assuming it is the case of course).

As a simple example consider the second order sentence:

$$\phi = \exists R \forall x \forall y (R(x, y) \Rightarrow E(x, y)).$$

This simply states the existence of a subgraph of a given graph; the relation R (for which $r = 2$) gives the edges of the subgraph. The sentence is always true since every graph has at least one subgraph (itself!). We can assert the existence of a subgraph with fewer edges by:

$$\exists R (\forall x \forall y (R(x, y) \Rightarrow E(x, y)) \wedge \exists x \exists y (E(x, y) \wedge \neg R(x, y))).$$

Of course this formula is true only for graphs that have at least one edge.

We now show how to express reachability in this framework. The underlying idea is that we can go from vertex x to vertex y if and only if

1. there is a linear ordering of (some of) the vertices of the graph (these make up, in strict order, the path from x to y where we have removed loops),
2. any two *consecutive* vertices in the order are joined by an edge in the graph (going from the smaller vertex to the larger one),
3. the first vertex is x , the last is y and x, y are in the order.

We will denote the order whose existence is to be asserted, by a binary relation symbol L . Now we build up the formula by:

1. L is a linear order on a subset of the vertices:

$$\begin{aligned}\psi_1 = & \forall u \neg L(u, u) \\ & \wedge \forall u \forall v (L(u, v) \Rightarrow \neg L(v, u)) \\ & \wedge \forall u \forall v \forall w (L(u, v) \wedge L(v, w) \Rightarrow L(u, w)).\end{aligned}$$

The first conjunct expresses the requirement that no vertex is smaller than itself. The second conjunct expresses the fact that we cannot have *both* $a < b$ and $b < a$. Finally the third conjunct expresses transitivity (familiar to us in the form $a < b$ and $b < c$ implies $a < c$).

2. Any two consecutive vertices in the order must be joined by an edge in G :

$$\psi_2 = \forall u \forall v ((L(u, v) \wedge \forall w (\neg L(u, w) \vee \neg L(w, v))) \Rightarrow E(u, v)).$$

Note that the notion of u, v being consecutive is captured simply by requiring the absence of any vertex that comes after u and before v .

3. The first vertex in the order is x and the last is y and x, y are in the order:

$$\psi_3 = (\forall u \neg L(u, x)) \wedge (\forall v \neg L(y, v)) \wedge L(x, y).$$

It is tempting to take $\exists L(\psi_1 \wedge \psi_2 \wedge \psi_3)$ as our formula but there is a problem when the start vertex x is the same as the end vertex y . To overcome this we take

$$\phi(x, y) = \exists L(x = y \vee (\psi_1 \wedge \psi_2 \wedge \psi_3)).$$

EXERCISE Do we really need the second conjunct of ψ_1 ?

Given a formula $\exists R \phi$ we can formulate the decision problem $\exists R \phi$ -GRAPHS in the same way as the problem ϕ -GRAPHS for first order logic. Then:

THEOREM 14.3 *Let $\exists R \phi$ be an expression of existential second order logic. Then the problem $\exists R \phi$ -GRAPHS is in NP.*

PROOF. Suppose R is a relation symbol on r -tuples. Given a graph $G = (V, E)$ guess a relation R^G that gives R on V^r . Note that the size of this is at most n^r where $n = |V|$, the number of vertices of the graph. After this we can check if ϕ is satisfied in polynomial time in the same way as for first order logic (see the proof of Theorem 14.1); the only difference is that for the base case we also have expressions of the form $R^G(v_1, v_2, \dots, v_r)$ where v_1, v_2, \dots, v_r are vertices of G . \square

Can we do better? Can we put $\exists R \phi$ -GRAPHS in P? Well, a simple modification of the formula that expresses REACHABILITY shows that we can also express DHP (directed Hamiltonian paths). We simply assert that the order is total:

$$\psi_4 = \forall u \forall v (L(u, v) \vee L(v, u) \vee u = v).$$

Then the formula

$$\exists L(\psi_1 \wedge \psi_2 \wedge \psi_4),$$

is satisfied by a graph if and only if it has a directed Hamiltonian path. However we know that DHP is NP-complete and so we cannot hope to put $\exists R \phi$ -GRAPHS in P unless we have $P = NP$. The fact that we can express at least one NP-complete problem in existential second order logic is no accident for in 1974 R. Fagin proved the converse of Theorem 14.3.

THEOREM 14.4 *A property of graphs is in NP if and only if it is expressible in existential second order logic.*

Impressive as this result is, one must not be carried away. The proof is essentially an encoding of polynomial time nondeterministic Turing machine computations in the appropriate logic. The role of the polynomial time bound for nondeterministic computation is played by the requirement that a relation exists (remember that a relation on r -tuples over a base set of size n has at most n^r members). Even so the result is very important in that it provides a means of translating between certain problems in logic (Fagin's original starting point was a purely logical question) and problems in complexity. For example the question of whether existential second order logic is closed under negation (i.e., if $\exists P \phi$ is a formula of the logic is there a formula that is equivalent to $\neg(\exists P \phi)$?) is equivalent to the complexity theoretic question of whether $NP = co-NP$ (a language is in $co-NP$ if and only if its complement is in NP). Note that if $NP \neq co-NP$ then $P \neq NP$ (because $P = co-P$ [why?]). However it is possible that $NP = co-NP$ even if $P \neq NP$.

What about capturing P by purely logical means? Here the situation is not quite so satisfactory. It is possible to capture P but all known methods involve the introduction of a concept that is extraneous to the logic. One method is as follows. We consider expressions of the form

$$\exists R \forall x_1 \forall x_2 \dots \forall x_n \phi$$

where

1. ϕ has no quantifiers,
2. ϕ is a conjunction of clauses that contain at most one un-negated instance of the relation symbol R .

Such formulae are called *Horn existential second order formulae*. Now it can be shown that for such restricted formulae the problem $\exists R \phi$ -GRAPHS is in P. So far this is all pure logic. However in order to capture P it seems necessary to introduce a complication: we must allow a second relation symbol L which is to be interpreted as a linear order on the vertices of any graph used to interpret ϕ . (Thus in order to interpret ϕ we are forced to place an ordering on the vertices of the graph. This seems rather arbitrary since most graph properties do not themselves refer to an order; this is why the requirement of an order takes us outside pure logic. It is instructive to compare this with what happens when we encode problems as binary strings; again we introduce some linear order on structures that are not themselves ordered.) This form of augmented logic is called *Horn existential second order logic with successor*. It is then possible to show:

THEOREM 14.5 *A property of graphs is in P if and only if it is expressible in Horn existential second order logic with successor.*

Putting aside the concerns about the extraneous linear order we now have a rephrasing of the P versus NP problem: $P \neq NP$ if and only if DHP (or any other NP-complete problem on graphs) cannot be expressed in Horn existential second order logic with successor. We have already seen a non-expressibility proof: the fact that REACHABILITY cannot be expressed in first order logic. Unfortunately the situation once we move into second order logic is vastly more complicated. Indeed one can take the close tie up of logical questions with complexity ones as another indication of why certain questions of logic seem so difficult to solve. Studies of this form, and questions originating from databases, have led to the systematic study of *Finite Model Theory* which is in sharp contrast to traditional model theory where structures are normally infinite.

EXERCISE We saw earlier that DHP can be expressed by a formula of existential second order logic. The quantifiers in this formula can be collected together at the front: is the formula then in Horn form? Would you expect it to be?