Lecture 24

1 Truth and Complexity

Recall that while discussing propositional logic, we were particularly interested in the complexity of deciding the truth of sentences. The two kinds of questions were:

1. Given τ, φ does $\tau \models \varphi$? — LOGSPACE
2. Given φ, is $\models \varphi$? — co-NP complete

It is natural to consider similar questions for first order logic. We would like to know how difficult is it, algorithmically, to find out whether a sentence is valid or whether a given structure satisfies a sentence. We also would like to know whether such questions can be answered algorithmically.

The analogous questions for first order logic are:

1. Given A, α, φ, does $A, \alpha \models \varphi$? — For a finite A, this is exactly the quary-evauation problem. We saw that data complexity (fixed query) is LOGSPACE, while query complexity (fixed database) is PSPACE-complete.
2. Given φ, is $\models \varphi$? — This is *undecidable.* This result is known as the Church-Turing Theorem.

What happens if A is infinite in Problem 1 above? As an example, consider $A = (N, +, *, exp, =)$. Then Fermat's Last Theorem can be expressed as a sentences ψ and we know (because of Andrew Wiles' celebrated proof) that $A \models \psi$. So if problem 1 above was decidable, the algorithm could be used to decide Fermat's Tast Theorem, and for that matter any theorem in number theory. Thus the decidability of problem 1 has very important consequences for mathematics. At the turn of the century, it was hoped that this question would be answered in the affirmative. For this particular choice of infinite structure, the problem is undecidable. However, in general, the answer depends on the structure.

- $A = (N, +, *, =)$ is undecidable. Even the highly restricted form of $\varphi = (\exists x P(x) = 0)$, where $P(x)$ denotes a polynomial, is undecidable. That is, there is no algorithm to determine whether a given polynomial has an integer root.

- $A = (R, +, *, <)$: decidable. Since Euclidean Geometry can be formulated over $A = (R, +, *, <)$, this means that Euclidean Geometry is decidable.
- $A = (N, +, *, <)$: decidable. This is called *Presburger's Arithmetics*

2 Undecidability

Consider the two questions:

1. Given a program, does it terminate?
2. Given a program, does it diverge?

There is a difference between the two. Given that a program terminates, we can observe it terminating (by just running it and waiting). However, if the program does not terminate, then we can never verify this fact. This immediately shows that there are different degrees of decidability. We say a problem is *semi-decidable* if there is an algorithm for it that always halts on correct inputs. So semi-decidability has positive witnesses (like NP). This notion is also called *computable enumerability.*

So, how we can prove undecidability for logical reasoning? Remember that if L_1 is reducible to L_2, and L_1 is undecidable, then L_2 is undecidable. As in NP-completeness, we need the “first” undecidable problem. Such a problem is the Halting problem, and it is proved by the diagonalizing method.

3 Simple Programming Language

Consider the Simple Programming Language–SPL, having the following instructions

1. Assignments: $x = x + 1$, and $x = x - 1$
2. Branching: goto line l
3. Conditional branching: If $x = 0$, then goto line
4. Termination: *stop*

We also call this a counter machine (CM), because it essentially counts up or down using one or more variables. The decision problem for a CM is: Starting at statement 0, with all variables initialized to 0, will the machine reach a statement k? We denote the decision problem for a CM with two variables as 2CM. The Halting problem for 2CM is undecidable.

Consider $S_i(t_1, t_2)$ to mean that the machine is in line i with t_1, t_2 in its counters. We can model the machine using the set of the following axioms:

1. $S_0(0, 0)$

2. $(\forall x)(\forall y)(S_i(x,y)) \to S_{i+1}(x+1,y))$, if Line i is "$x = x + 1$".
3. $(\forall x)(\forall y)(S_i(x+1,y)) \to S_{i+1}(x,y))$, if Line i is "$x = x - 1$".
4. $(\forall x)(\forall y)(S_i(x,y)) \to S_j(x,y)$ if Line i is "goto j".
5. $(\forall x)(\forall y)(S_i(0,y)) \to S_j(0,y)$ if Line i is "$x = 0$, goto j".

Axioms for variable y are similar to those above. Let $Axioms_M$ correspond to the set of axioms representing the counter machine M.

Lemma 1 *The machine M reaches statement k iff*

$$\models (Axioms_M \to (\exists x(\exists y)S_k(x,y)).$$

To reduce divergence to logical truth, we can make the relations $S_0,\ldots,S_k$ ternary relations, where the third argument represents time. The axiom should express the fact that every machine step takes one time unit. For example:

- $(\forall x)(\forall y)(S_i(x,y,t)) \to S_{i+1}(x+1,y,t+1))$, if Line i is "$x = x + 1$".

Now we have

Lemma 2 *The machine M diverges iff*

$$\models (Axioms_M \to (\forall t)(\exists t')(t < t') \land (\exists x)(\exists y)S_k(x,y,t')).$$

Note that here we have added to Arithmetics relations $S_0,\ldots,S_k$. We can actually get rid of these relations. This requires encoding sequences of numbers by numbers.

Theorem 1 *Arithmetical truth is not even semidecidable.*

This strong negative follows from the fact that if a problem and its complement are both semi-decidable, then they are both decidable. Since both termination and divergence reduce to arithmetical truth, it cannot be semidecidable. This theorem is known as the *Incompleteness* Theorem, since it implies that there is no sound and complete proof system for Arithmetics.

In practice, we have termination-checking tools that work quite well in practice. This is because most real-life programs terminate or diverge for fairly simply reasons and not for deep mathematical reasons.

See `cacm.acm.org/magazines/2011/7/109895-solving-the-unsolvable/fulltext`.