Logic and Probability, EPFL, Fall 2011
Lecture Notes 1

Val Tannen

1 How Mathematical Logic Gave Birth to Computer Science

Cantor's set theory: infinite objects in proofs. Mathematicians worry and argue. Paradoxes.

The formalization of mathematical proofs from Leibniz to Boole to Frege. Frege formalizes the predicate calculus and a form of set theory. Russell's Paradox.

Hilbert's interest in the foundations of mathematics. His Second Problem.

Russell and Whitehead's "Principia Mathematica", a ramified theory of types. Zermelo-Fraenkel set theory.

Hilbert's Program. Relative consistency results. Gödel's Completeness Theorem. The Decision Problem (Entscheidungsproblem).

Schönfinkel and later Curry invent a "functional"formalism, combinatory logic. Church invents the lambda calculus as a skeleton for an alternative to the Principia system and to ZF set theory.

Gödel shatters Hilbert's Program with his Incompleteness Theorems.

Kleene encodes arithmetic in the lambda calculus. He and Rosser show that the Church and Curry proposals for logics based on functional formalisms are inconsistent.

Church proposes lambda definability as a formal definition of computability/decidability (Church's Thesis). He shows that there exist undecidable problems in arithmetic and that the Entscheidungsproblem is undecidable.

Kleene shows that lambda definability is equivalent to Gödel's proposal for computability based on an idea of Herbrand (definability by systems of arithmetic equations).

Independently of Church, Turing shows that the Entscheidungsproblem is undecidable. His proposal for computability relies, of course, on what we call now Turing Machines and, critically, on the Universal Turing Machine. After he becomes aware of the work by Church and Kleene he shows that his notion of computability is equivalent to lambda definability.

Inspired by his work on lambda calculus Kleene develops recursion theory, out of which complexity theory emerges later.

Church develops the simply typed lambda calculus as the foundation of "a simple theory of types". This leads eventually to modern proof assistants based on type theory.

Large-scale vacuum tube technology enables the construction of the building-size decimal calculator ENIAC by Eckert and Mauchly. Von Neumann is a consultant to the project.

Eckert and Mauchly design ENIAC's successor, the binary machine EDVAC. Inspired by the Universal Turing Machine, von Neumann designs for EDVAC the stored program architecture that

today bears his name.

2 Friendly and Unfriendly Aspects of First-Order Logic, in General

Our starting point is the semantics of first-order logic(FOL), specifically **logical consequence** assertions. These have the form $\Gamma \models \varphi$ where φ is an FO sentence and Γ is set of FO sentences. For example, we can think of Γ as an "axiomatization" of a mathematical theory or as part of a "modelization" of a physical or informatic system. So our initial question is: what kind of computing can we do with logical consequence assertions?

Some theories are axiomatized by finite sets of axioms but most make use of so-called *axiom schemes*, which are finite descriptions of infinite sets of axioms. Thus, we will consider so-called *recursively axiomatized* theories, given by a Γ that is decidable. [1]

Starting with the work of Frege, several equivalent **proof systems** for FOL were formalized by Hilbert and others. Answering a question of Hilbert, Gödel showed that the intuition captured by the proof systems was correct:

Theorem 2.1 (Gödel's Completeness Theorem) $\Gamma \models \varphi$ *iff* $\Gamma \vdash \varphi$.

The FOL proof systems and, in fact, any reasonable proof system has a nice computational nature: proofs are finite objects, it is decidable whether a finite object is a correct proof and the sentence proven by a proof can be computed from it. This implies that $\{(\Gamma, \varphi) \mid \Gamma \vdash \varphi\}$ is r.e. By Gödel's Completeness Theorem and defining

$$\mathit{VALID} \stackrel{\text{def}}{=} \{\varphi \mid \models \varphi\}$$

we have in particular:

Corollary 2.2 *VALID is r.e.*

This is good, but even better would be if we could actually *decide* first-order provability (hence logical consequence). This question, known as the *Entscheidungsproblem* [2], and again asked by Hilbert, has a negative answer:

Theorem 2.3 (Church/Turing's Undecidability Theorem) *VALID is undecidable.*

A proof of Theorem 2.3 can be found in section 5 of my lecture notes on computability.

We still wish to find cases in which logical consequence is decidable. and there are two strategies we use in what follows: to restrict ourselves to FO sentences of a special form and to restrict ourselves to special classes of models.

An obvious restriction of the second kind is to look at validity over **finite models**. In fact, in computer science we are concerned mostly with finite structures so this is very natural. Define

$$\mathit{FIN\text{-}VALID} \stackrel{\text{def}}{=} \{\varphi \mid \mathcal{A} \models \varphi \text{ for all finite } \mathcal{A}\}$$

[1] Note that when Γ is finite $\Gamma \models \varphi$ is equivalent to $\models \Gamma \rightarrow \varphi$ (interpreting Γ as the conjunction of its elements).

[2] "Decision problem" in German. At the beginning of the 20^{th} century most papers on mathematical logic were in German.

Exercise 2.1 *Give an example of an FO sentence that is finitely valid but not valid.*

Unfortunately, finite validity is even worse than validity! We have:

Theorem 2.4 (Trakhtenbrot's Theorem) *FIN-VALID is undecidable.*

In Libkin's "Elements of Finite Model Theory", pp 165–168, it is shown how to compute from the description of a Turing Machine M a FO sentence φ_M such that M halts on the empty string input iff φ_M is finitely satisfiable. Like the Halting Problem, this problem is also undecidable (in fact, it is also r.e.-complete). It follows that

$$\textit{FIN-SAT} \stackrel{\text{def}}{=} \{\varphi \mid \text{ there exists a finite } \mathcal{A} \text{ such that } \mathcal{A} \models \varphi\}$$

is undecidable. Using the reduction $\varphi \longmapsto \neg\varphi$ we conclude that $\overline{\textit{FIN-VALID}}$ and therefore *FIN-VALID* are also undecidable.

Exercise 2.2 *Show how to modify the proof of Trakhtenbrot's Theorem in Libkin's book in order to prove the Church-Turing Theorem.*

In fact *FIN-VALID* is not even r.e.! This is easy to see if we look at the **satisfiability** property. Recall that a sentence φ is (*finitely*) *satisfiable* if there exists a (finite) model $\mathcal{A}$ such that $\mathcal{A} \models \varphi$. Note that φ is satisfiable iff $\neg\varphi$ is invalid and φ is valid iff $\neg\varphi$ is unsatisfiable. Hence, it follows from the Church/Turing and Trakhtenbrot theorems that satisfiability and finite satisfiability are both undecidable. Moreover, it follows from Gödel's Completeness Theorem that

Corollary 2.5 *The set of satisfiable FO sentences is co-r.e.*

The situation with finite satisfiability is quite different:

Proposition 2.6 *The set of finitely satisfiable FO sentences is r.e.*

Proof Isomorphic models satisfy the same sentences so it's sufficient to consider finite models whose universe (domain) is of the form $\{1, 2, \ldots, n\}$. We can enumerate all pairs consisting of such a finite model and an FO sentence, and output the sentence if it holds in the model. (This assumes that for finite models $\mathcal{A} \models \varphi$ is decidable; see Theorem 5.2.) □

Note that although finite satisfiability is r.e., it is nonetheless undecidable, as follows from Trakhtenbrot's Theorem. Moreover,

Corollary 2.7 *FIN-VALID is not r.e.*

Proof If *FIN-VALID* was r.e. then finite satisfiability would be co-r.e., hence also decidable, by Proposition 2.6. This would make *FIN-VALID* decidable which contradicts Trakhtenbrot's Theorem. □

Therefore, FOL " in finite models" is worse than standard FOL: it does not even admit a complete proof system!

Exercise 2.3 *VALID and FIN-SAT are both r.e.-complete hence many-one reducible to each other. Describe as best you can two total computable functions that realize these two many-one reductions.*

3 Reduction Classes

Let us now look at the other strategy for finding decidable cases of logical consequence: restricting the class of sentences.

Unfortunately, we begin with a disappointment: even for (apparently) simple classes of sentences validity is undecidable.

Definition 3.1 A class C of FO sentences is called a **reduction class** if there is a computable function f that maps arbitrary FO sentences into C-sentences such that φ is valid iff $f(\varphi)$ is valid. It follows from the Church/Turing Theorem that the validity problem for C is also undecidable.

Intuitively, reduction classes have a validity problem that is "as hard as" the validity problem for all of FOL. Even before the Church/Turing result, various reduction classes were exhibited (note that you don't really need a formalized definition of computability for such results). We give an example in what follows.

Definition 3.2 A formula of the form $\varphi \stackrel{\text{def}}{=} Q_1 x_1 \cdots Q_n x_n \; \psi$ where each Q_i is either $\forall$ or $\exists$ and ψ is quantifier-free is called a *prenex* formula. If the quantifiers are all $\forall$ (all $\exists$) then φ is called a *universal* formula (an *existential* formula).

In what follows, by **equivalent** sentences we mean two sentences that are logical consequences of each other. This is denoted $\varphi \models\!\models \psi$ and it holds iff $\models \varphi \leftrightarrow \psi$.

Lemma 3.1 *For each sentence we can compute (in PTIME) an equivalent prenex sentence.*

Proof "Pull out" the quantifiers by repeatedly using transformations like

$$\neg\exists x\, \varphi \longmapsto \forall x\, \neg\varphi \qquad \varphi \wedge (\forall x\, \psi) \longmapsto \forall x(\varphi \wedge \psi) \qquad (\forall x\, \varphi) \rightarrow \psi \longmapsto \exists x(\varphi \rightarrow \psi)$$

etc., while renaming bound variables to avoid unintended scope capture. □

Exercise 3.1 *Analyze the complexity of the algorithm sketched above; for any FO sentence it computes an equivalent prenex FO sentence. You can choose the data structure used to represent sentences.*

Lemma 3.2 (Skolem) *Let $\mathcal{V}$ be a vocabulary and let $\bar{\mathcal{V}}$ be its extension with countably many fresh function symbols of each arity (including nullary functions i.e. constants). For each prenex sentence φ over $\mathcal{V}$ we can compute (in PTIME) a sentence $Sk(\varphi)$ over $\bar{\mathcal{V}}$ such that*

- *$Sk(\varphi)$ is a universal sentence.*
- *φ is true in the $\mathcal{V}$-restriction of any model of $Sk(\varphi)$.*
- *Any model of φ can be extended, keeping the same universe (domain), to a $\bar{\mathcal{V}}$-structure that satisfies $Sk(\varphi)$. (Hence, φ is satisfiable iff $Sk(\varphi)$ is satisfiable.)*

Proof Eliminate the existential quantifiers from left to right in the prenex sentence repeating the transformation

$$\forall x_1 \cdots \forall x_n \exists y\, \varphi(y) \longmapsto \forall x_1 \cdots \forall x_n\, \varphi(f(x_1, \ldots, x_n))$$

where f is a fresh functions symbol.

For example:

$$Sk(\exists u\,\forall x\,\exists v\,\forall y\,\exists w\,R(u,v) \wedge f(x,w) = g(v)) \stackrel{\text{def}}{=} \forall x\,\forall y\,R(r,s(x)) \wedge f(x,t(x,y)) = g(s(x))$$

□

$Sk(\varphi)$ is unique up to some renaming so it is called the *Skolem Normal Form* of φ. The transformation $\varphi \longmapsto Sk(\varphi)$ is called *skolemization*. Symbols like r, s and t in the proof example are called *Skolem functions*.

Exercise 3.2 *Analyze the complexity of the algorithm sketched above; for any prenex FO sentence it computes an equivalent Skolem normal form FO sentence. Again, you can choose the data structure used to represent sentences.*

Theorem 3.3 *The existential sentences form a reduction class.*

Proof Given an FO sentence φ and assuming that the transformations to prenex form are implicit, observe that $\neg Sk(\neg\varphi)$ is an existential sentence that is valid iff φ is valid. □

Warning! Our definition of reduction class is for the *validity* decision problem. It is more common to define reduction classes for the *satisfiability* decision problem. If the class is closed under negation the two coincide. But classes defined by the quantifier structure of prenex formula are typically *not* closed under negation! For example, the theorem above is often stated as "the universal sentences form a reduction class (wrt decidability of satisfiability)".

4 The Finite Model Property

Although finite validity has bad computational properties for the class of *all* FO sentences, the r.e.-ness of finite satisfiability can be exploited for classes of restricted FO sentences.

Definition 4.1 A class of sentences has the **finite model property** if any satisfiable sentence in the class is also finitely satisfiable.

The class of all FO sentences does not have the finite model property. Indeed, it is fairly easy to concoct sentences that are satisfiable but not finitely satisfiable . Such sentences are called *infinity axioms*. In fact, if the class of all FO sentences would have the finite model property then the next result would contradict the undecidability results shown earlier!

Proposition 4.1 *Let C be a class of sentences that is decidable (i.e., it is decidable whether an FO sentence φ is in C). If C has the finite model property then (finite or general) satisfiability of C-sentences is decidable.*

Proof Recall that for the class of all FO sentences satisfiability is co-r.e. and finite satisfiability is r.e. Since C is decidable it enjoys the same properties. But for the sentences in C satisfiability and finite satisfiability coincide! Hence they are both r.e. and co-r.e. and thus decidable. □

The decision procedure provided by the previous proof is very inconvenient: it consists of trying, in parallel, to finitely satisfy the sentence and to prove its negation (if the sentence is satisfiable then the first thread succeeds; if not then the second thread does). A better procedure is given by the following.

Definition 4.2 A class of sentences has the **small model property** if there is a total recursive function u such that any satisfiable sentence in the class has a model with less than $u(\mid \varphi \mid)$ elements. (Here $\mid \varphi \mid$ denotes the **size** of φ.)

The small model property implies the finite model property. However, a decidable class of sentences with the small model property has a decision procedure for satisfiability that is potentially simpler. There is no need for the annoying attempt to prove the negation of a sentence φ; it suffices to check all models with less than $u(\mid \varphi \mid)$ elements.

Note that we only said *potentially* simpler: it all depends on how easy to compute u is. Indeed, if C is decidable then the finite model property implies the small model property! Consider the following algorithm for u:

> On input n, generate the (finitely many) sentences in C of size n. For each of them, in parallel, check for finite satisfiabilty and try to prove the negation, and thus compute either the size of a model or 0 (if unsatisfiable). Return as $u(n)$ the largest of these.

Using this u the small model property gives a decision procedure that is just as inconvenient as the one given by the finite model property. If we consider classes C that are not decidable a general observation is that there exist such classes that have the finite model property but do not have the small model property: simply take the class of all finitely satisfiable sentences. Indeed, suppose there is a total recursive function u such that any satisfiable sentence φ in this class has a model with less than $u(\mid \varphi \mid)$ elements. But all the sentences in this class are satisfiable! Hence φ is finitely satisfiable iff it has a model with less than $u(\mid \varphi \mid)$ elements. This would make finite satisfiability decidable.

Anyway, what happens for concrete classes of sentences is that a better and more specific u is derived which moreover gives a useful complexity upper bound for the decision procedure. Here is an example:

Theorem 4.2 *The existential FO sentences have the small model property. In fact, any satisfiable existential sentence φ has a model with at most $\mid \varphi \mid$ elements.*

Proof Let $\varphi \stackrel{\text{def}}{=} \exists x_1 \cdots \exists x_m \, \psi$ be an existential sentence where ψ is quantifier-free. We replace each non-variable functional term occurring in ψ with a fresh existentially quantified variable and an additional equality atom. We do this bottom-up for all subterms, including constants, so if $t \equiv f(t_1, \ldots, t_k)$ is such a term and $t_1, \ldots, t_k$ are replaced by $y_1, \ldots, y_k$ then t is replaced by a fresh variable y and we add the equality atom $f(y_1, \ldots, y_k) = y$. Therefore, φ is transformed into a sentence $\bar{\varphi}$ of the form

$$\bar{\varphi} \stackrel{\text{def}}{=} \exists x_1 \cdots \exists x_m \, \exists y_1 \cdots \exists y_n \, \bar{\psi} \wedge f_1(\ldots) = y_1 \wedge \cdots \wedge f_n(\ldots) = y_n$$

where $f_1, \ldots, f_n$ are the functional symbol occurrences in ψ (we treat constants as nullary functions).

For example, $\exists x \, R(x, f(c, x))$ is transformed into $\exists x \, \exists y_1 \exists y_2 \, R(x, y_2) \wedge c = y_1 \wedge f(y_1, x) = y_2$.

It is not hard to see that φ and $\bar{\varphi}$ hold in the same models and that if $\bar{\varphi}$ is satisfiable then it has a model with at most $m + n$ elements. Since $n \leq \mid \psi \mid$ we conclude that if φ is satisfiable then it has a model with at most $\mid \varphi \mid$ elements. $\square$

Corollary 4.3 *Satisfiability of existential sentences is NP-complete.*

Proof For membership in NP, let the sentence be $\varphi \stackrel{\text{def}}{=} \exists x_1 \cdots \exists x_m \, \psi(x_1, \ldots, x_m)$. We guess a model $\mathcal{A}$ with less then $\mid \varphi \mid$ elements, we also guess a valuation v that maps each of $x_1, \ldots, x_m$ to some element of $\mathcal{A}$, and then we check $\mathcal{A}, v \models \psi$. Checking the truth of a quantifier-free formula can be done in PTIME in the size of the formula, see Theorem 5.1.

For NP-hardness we provide a reduction from boolean satisfiability. Given a boolean formula β with propositional variables $p_1, \ldots, p_n$ we construct the existential sentence $\varphi \stackrel{\text{def}}{=} \exists x_1 \cdots \exists x_n \, \bar{\beta}$ over a vocabulary with one unary predicate symbol R where $\bar{\beta}$ is obtained by replacing each p_i with $R(x_i)$. Now consider the "binary" model $\mathcal{B}$ whose universe is $\{0, 1\}$ and where R is interpreted as $\{1\}$. Clearly β is satisfiable iff $\mathcal{B} \models \varphi$. It is also easy to see that if φ is satisfiable then it holds true in $\mathcal{B}$. Therefore, β is satisfiable iff φ is. $\square$

And this finally gives a class of sentences for which the validity problem is decidable:

Corollary 4.4 *Validity of universal sentences is decidable and coNP-complete.*

This means that the satisfiability of *existential* sentences is also decidable and in fact NP-complete. (Note that Theorem 3.3 says that *validity* is undecidable for this same class.)

For certain simple existential sentences satisfiability is trivially decidable: they are in fact all satisfiable!

Exercise 4.1 *An* existential-conjunctive *sentence is a sentence of the form $\exists x_1 \cdots \exists x_n \, \varphi$ where φ is a conjunction of atomic formulas (equalities are allowed). Prove that any existential-conjunctive sentence is satisfiable in a model with one element.*

5 Model Checking for FOL

Given a vocabulary $\mathcal{V}$, the **model checking** [3] problem consists of deciding if $\mathcal{A} \models \varphi$ for an FOL $\mathcal{V}$-sentence φ and a finite $\mathcal{V}$-structure $\mathcal{A}$.

We shall be interested in the complexity of the model checking problem, in fact in three different situations:

Combined complexity The input is $(\mathcal{A}, \varphi)$, its size is $\mid \mathcal{A} \mid + \mid \varphi \mid$.

Expression complexity $\mathcal{A}$ is fixed, the input is just φ.

Data complexity φ is fixed, the input is just $\mathcal{A}$.

We must be clear about what is meant by $\mid \mathcal{A} \mid$. This is the size of the complete description of $\mathcal{A}$. But how do we give such a description as input to a Turing machine? We fix an enumeration of the elements of the (non-empty) universe of $\mathcal{A}$: $a_1, a_2, \ldots a_n$ then we represent (encode) the model on Turing machine tapes with respect to this enumeration. Specifically, we represent a_k by the number k, in *binary*, therefore by using $\log k$ bits. We encode tuples and then relations using some separating symbols. The tuples in a relation will be ordered lexicographically, based on the ordering of domain elements given by the enumeration. For example, consider the enumerated domain a, b, c. Then $R = \{(c, b), (a, c), (a, b)\}$ $S = \{(c, a, c), (a, a, b)\}$ can be encoded by the string `R/0-1/0-10/10-1/S/0-0-1/10-0-10/`. The interpretation of function symbols of arity m

[3] This name comes from automated verification. We shall see later that model checking for many verification logics is a particular case of FOL model checking.

is represented as $m+1$-relations in which the tuples group the arguments and the corresponding result of the function. Finally, we make the size of the universe, in *unary* (to avoid pathologies) part of the representation.

If the universe of $\mathcal{A}$ has n elements and if the vocabulary contains an m-ary relation symbol R then the model description contains the encoding of as many as n^m tuples. In general, for a fixed vocabulary $\mid \mathcal{A} \mid$ is no more than polynomial in the size of the universe of $\mathcal{A}$.

We begin with with the quantifier-free case and it will be useful to consider a slightly more general problem, involving formulas and valuations rather than just sentences: decide if $\mathcal{A}, v \models \psi$ where ψ is a quantifier-free formula and v is a valuation defined for a set of variables that includes the free variables of ψ. In this case, we consider v, together with ψ, to be the input for expression complexity.

Theorem 5.1 *The data complexity of quantifier-free formula checking is in LOGSPACE. The expression and combined complexities of the same problem are in PTIME.*

Proof The procedure relies on evaluating the functional terms in the formula, determining the truth value of the atoms, and evaluating the resulting boolean expression. Term evaluation is done akin to arithmetic expression evaluation, using a stack. This gives us the PTIME expression and combined complexities. For the data complexity note that the size of the stacks does not depend on $\mid \mathcal{A} \mid$. We just need to lookup tuples in the description of $\mathcal{A}$ and this can be done with pointers of size $\log \mid \mathcal{A} \mid$. This gives the LOGSPACE bound on data complexity. □

Remark It has been shown that boolean expressions can be evaluated in LOGSPACE. (Indeed, to evaluate, e.g., $a \vee b$ we do not need to record both the value of a and b.) We would not be evaluating b at all if a resulted in true. Therefore, in the absence of function symbols, i.e., when the vocabulary consists only of relation symbols and constants, the expression and combined complexity of quantifier-free model checking are also in LOGSPACE [4]

Theorem 5.2 *The data complexity of FOL model checking is in LOGSPACE. The expression and combined complexities of FOL model checking are PSPACE-complete.*

Proof First we put the sentence in prenex form [5]. Let $Q_1 x_1 \cdots Q_k x_k \ \psi$ be the result.

We evaluate this sentence in $\mathcal{A}$ using k nested loops iterating each of $x_1, \ldots, x_k$ through all the elements of the universe of $\mathcal{A}$. If x_i is universally quantified then the loop corresponding to x_i computes a big conjunction (disjunction if existentially quantified). In the innermost loop we evaluate $\mathcal{A}, v \models \psi$ where v is the valuation recording the current values of $x_1, \ldots, x_k$. If m is the number of elements of $\mathcal{A}$ then the time complexity of doing all this is

$$O(m^k \ poly(\mid \mathcal{A} \mid, \mid \psi \mid))$$

where *poly* is a two-variable polynomial. For the space complexity, note that we can keep track of the current v by using k pointers of size $\log m$. Thus the space complexity adds $O(k \log m)$ to the space complexity of the quantifier-free case. This gives LOGSPACE for data complexity and PSPACE for expression and combined complexities.

[4] However, it seems that we cannot evaluate functional terms in LOGSPACE. (Think what goes wrong with the stack-based procedure.) It is known that there exist context-free languages that are NLOGSPACE-complete. I am still trying to find out whether such languages can be reduced to the problem of functional term evaluation.

[5] I don't know if this can be done in LOGSPACE but for data complexity the sentence is fixed. It certainly can be done in low degree PTIME hence PSPACE.

To show PSPACE-completeness we reduce from *QBF*, the problem of checking the truth of a (fully) quantified boolean formula. This is essentially the same reduction performed in the proof of Corollary 4.3.

Given a fully (no free propositional variables) quantified boolean formula $\gamma \stackrel{\text{def}}{=} Q_1 p_1 \cdots Q_k p_k \ \beta$ we construct the FO sentence $\varphi \stackrel{\text{def}}{=} Q_1 x_1 \cdots Q_k x_k \, \bar{\beta}$ over a vocabulary with one unary predicate symbol R where $\bar{\beta}$ is obtained by replacing each p_i with $R(x_i)$. Clearly γ is true iff $\mathcal{B} \models \varphi$ where $\mathcal{B}$ is defined in the proof of Corollary 4.3. Since $\mathcal{B}$ is a fixed model, this gives a lower bound for both combined complexity and expression complexity.