Friendly Logics, Fall 2015, Lecture Notes 1

Val Tannen

1 Some references

Course Web Page: `http://www.cis.upenn.edu/~val/CIS682`.

I have posted there the remarkable "On the Unusual Effectiveness of Logic in Computer Science", by Halpern, Harper, Immerman, Kolaitis, Vardi, and Vianu. I recommend that you read this paper at the beginning of this course, and skim it again at the end.

I will assume that you already have some background in computability. I may include a couple of problems in the first homework to help you remember. Just in case, I am posting my own **notes on computability** on the course web page. Do you need to review computability? Here is a short and incomplete checklist that you can use to figure this out. Specifically, do you know the answer to the following questions?

- What is a language of strings (words) over an alphabet?
- What is an r.e. language?
- What is a computable (partial) function?
- What is a decidable language?
- What is a co-r.e. language? How do r.e. and co-r.e. languages relate to decidable languages?
- Is the Halting Problem r.e.? Is it decidable?
- Can you think of an undecidable language that is not defined in terms of Turing Machines?
- What is a many-one reduction?
- What is an r.e.-complete (or co-r.e.-complete) language?

In addition to my notes, here are some books that I recommend for **computability/complexity**:

"Introduction to the Theory of Computation", (2nd Ed), Sipser, Thomson 2006.

"Introduction to Automata Theory, Languages, and Computability", (2nd Ed), Hopcroft, Motwani, Ullman, Addison-Wesley 2001.

"Computational Complexity", Papadimitriou, Addison-Wesley 1994.

"Computational Complexity", Arora and Barak, Cambridge Univ. Press 2009.

Books I recommend for **logic**:

"Logic for Computer Science" (2nd Ed), Jean Gallier, (Wiley 1986), reprinted (Dover 2015), also available online at `http://www.cis.upenn.edu/~jean/gbooks/logic.html`

"A mathematical introduction to logic", (2nd Ed), Enderton, Harcourt/Academic Press, 2001.

"Introduction to mathematical logic", Mendelson, various publishers, 1964, 1979, 1987, and 1997.

"Computability and logic", Boolos, Burgess, and Jeffrey, Cambridge Univ. Press, 2007.

The complete **classification** of the prefix fragments of FOL from the point of view of the decidability of satisfiability is in:

"The Classical Decision Problem", Boerger, Graedel, and Gurevich, Springer 1997.

For **finite model theory** and **descriptive complexity** you can consult

"Elements of Finite Model Theory", Libkin, Springer 2004.

"Finite Model Theory", Ebbinghaus, Flum, Springer 1999.

"Descriptive Complexity", Immerman, Springer 1999.

The **story** of the role of mathematical logic in the birth of computer science is told best in:

"Engines of Logic: Mathematicians and the Origins of the Computer", Davis, 2nd edition, Norton 2001.

2 Review of FOL syntax and semantics

A first-order (FO) **vocabulary/signature** consists of a set of function symbols and a set of relation symbols, each with an *arity* (a natural number). For function symbols the arity is the number of arguments while for relation symbols the arity is the number of components. It is often convenient to allow for infinite sets of such symbols. (Set theorists in the audience can assume that said infinity is countable and please refrain from giving me a hard time.) For example the relation symbols of arity 0 are propositional constants so they can be used as boolean variables. With this, propositional logic sits nicely inside FOL.

FO **terms**: $x \mid c \mid f(t_1, \ldots, t_n)$ where x ranges over variables, c over constants (function symbols of arity 0), n over positive integers, f over function symbols of arity n, and the t's over terms.

FO **atomic formulas**: $p \mid \mathsf{true} \mid \mathsf{false} \mid R(t_1, \ldots, t_n) \mid t = t'$ where p ranges over propositional constants (relation symbols of arity 0), n over positive integers, R over relation symbols of arity n, and the t's over terms.

FO **formulas**: $\alpha \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \Rightarrow \varphi_2 \mid \neg\, \varphi \mid \forall x\, \varphi \mid \exists x\, \varphi$ where α ranges over atomic formulas, x over variables, and the φ's over formulas.

Bound and free variables are defined as usual. We denote by $Var(\varphi)$ the set of variables free in φ. A formula such that $Var(\varphi) = \emptyset$ is called a **sentence**.

FO **provability**: Starting from the work of Frege (the Begriffsschrift), Peano, and Whitehead-Russell (Principia Mathematica) [1] , several equivalent proof/deduction systems (inference rules + axioms and/or axiom schemes) for FOL were formalized by Hilbert and others. We fix one of these FOL proof systems and provability will from now on be stated in terms of it. We need not quibble about the details of the proof system but there are some properties that all such systems share and that we will invoke as needed. Here is one

- A formula φ is provable iff the *sentence* $\forall x_1 \ldots \forall x_n \varphi$ is provable, where $\{x_1, \ldots, x_n\} = Var(\varphi)$.

This allows us to use, w.l.o.g. just sentences in the following definitions. If Σ is a set of *sentences*, and σ a single sentence we write

$$\Sigma \vdash \sigma$$

when there exists an FOL proof/deduction of σ that can use sentences from Σ as additional axioms. When $\Sigma = \emptyset$ we just write $\vdash \sigma$. An important property that FOL provability inherits from propositional logic is the following:

- If Φ is a *finite* set of sentences then $\Phi \vdash \sigma$ iff $\vdash \Phi^{\wedge} \Rightarrow \sigma$, where $\Phi^{\wedge}$ is the conjunction of all the sentences in Φ.

Further, we introduce notations for the set of sentences provable in FOL and for the provable/deductive consequences of a set of sentences.

$$PROV \stackrel{\text{def}}{=} \{\sigma \mid \vdash \sigma\} \qquad\qquad Ded(\Sigma) \stackrel{\text{def}}{=} \{\sigma \mid \Sigma \vdash \sigma\}$$

Note that $PROV = Ded(\emptyset)$.

Definition 2.1 *A set Σ of sentences is **inconsistent** if $\Sigma \vdash$ false and **consistent** otherwise.*

Exercise 2.1 *The following are equivalent to the* inconsistency *of Σ*

1. *$\Sigma \vdash \sigma$ and $\Sigma \vdash \neg\sigma$ for some sentence σ.*
2. *$Ded(\Sigma)$ contains* all *sentences.*
3. *$\Sigma \vdash p$ for some propositional constant p.*

State clearly all the assumptions about the FOL proof system that you are using (e.g., that it is closed under modus ponens*: if $\Sigma \vdash \sigma$ and $\Sigma \vdash \sigma \Rightarrow \rho$ then $\Sigma \vdash \rho$).*

The consistency of $\emptyset$ is the "consistency" of the FOL proof system (that we fixed) itself.

Because proofs are finite and because it is decidable when a finite object is a proof as well as what formula it proves, the concept of FOL provability is "computational" in the following sense:

[1] Check out `https://en.wikipedia.org/wiki/Logicomix`.

Proposition 2.1 *PROV is recursively enumerable (r.e.). If Σ is decidable then $Ded(\Sigma)$ is r.e.*

This observation is very general, as it applies to most deduction/inference formalisms used in Computer Science.

FO **models/structures/interpretations/instances**: $\mathcal{A}$ consists of a non-empty set A (the "universe" of $\mathcal{A}$ plus for each function symbol f in the vocabulary a function $f^{\mathcal{A}} : A^n \to A$ where n is the arity of f plus for each relation symbol R in the vocabulary a subset $R^{\mathcal{A}} \subseteq A^n$ where n is the arity of R. (Note that for propositional constants p we have that $p^{\mathcal{A}}$ is either a set with one element or the empty set. Identifying these with true and false we see that a model can also serve as truth assignment for boolean variables.

FO **truth**: $\mathcal{A}, v \models \varphi$ where $\mathcal{A}$ is a model with universe A, φ is a formula, and $v : Var(\varphi) \to A$, is a **valuation** (an assignment of values from the model $\mathcal{A}$ to the free variables of φ). This can be defined recursively on the syntactic structure of φ. It is read as "φ is true (or holds) in $\mathcal{A}$ under the valuation v" or, sometimes "$\mathcal{A}$ is a model of φ under the valuation v".

Exercise 2.2 *Give a definition of $\mathcal{A}, v \models \varphi$.*

Note that for sentences the valuation can be omitted and we can write just $\mathcal{A} \models \sigma$, read "$\mathcal{A}$ is a model of σ". W.l.o.g. we can use just sentences in the definitions that follow because:

- $\mathcal{A}, v \models \varphi$ for all valuations v iff $\mathcal{A}$ is a model of the sentence $\forall x_1 \ldots \forall x_n \, \varphi$, where $\{x_1, \ldots, x_n\} = Var(\varphi)$.

If Σ is a set of sentences we write $\mathcal{A} \models \Sigma$ when each of the sentences in Σ is true in $\mathcal{A}$, read "$\mathcal{A}$ is a model of Σ". We also define the set of **valid** FOL sentences:

$$VALID \stackrel{\text{def}}{=} \{\sigma \mid \mathcal{A} \models \sigma \ \text{ for all } \mathcal{A}\}$$

as well as the notion of **logical consequence**: [2]

$$Con(\Sigma) \stackrel{\text{def}}{=} \{\sigma \mid \mathcal{A} \models \sigma \ \text{ for all } \mathcal{A} \text{ such that } \mathcal{A} \models \Sigma\}$$

Note that $VALID = Con(\emptyset)$ and thus for any model $\mathcal{A}$ we have $\mathcal{A} \models VALID$.

Definition 2.2 *A sentence is **satisfiable** if it has a model. A set Σ of sentences is satisfiable if $\mathcal{A} \models \Sigma$ for some model $\mathcal{A}$.*

Here is a basic sanity check for semantics:

Exercise 2.3 `true` *is satisfiable. (This is just a fancy way of asserting the trivial fact that for any FO vocabulary, FO models exist.)*

[2] The clean treatment of semantics for logic (model, logical consequences) is due to Tarski. Hilbert-Ackermann had a peculiar concept of "universal validity". See below how Gödel bypassed the issue in his proof of completeness.

And here is another:

Exercise 2.4 *σ is valid iff $\neg\sigma$ is unsatisfiable.*

FOL is just a framework for specifying mathematical *theories* and the *theorems* of such a theory. This can be done both syntactically and semantically.

FO **theory axiomatized by** Σ: notation $Th(\Sigma)$, the theorems are the sentences in $Ded(\Sigma)$. So

$$Th(\Sigma) \stackrel{\text{def}}{=} Ded(\Sigma)$$

Examples: algebraic theories (groups, fields, etc.) and PA (first-order Peano arithmetic).

FO **theory defined by a class** [3] $\mathcal{M}$ **of models**: notation $Th(\mathcal{M})$, and the theorem are the sentences true in all the models in $\mathcal{M}$:

$$Th(\mathcal{M}) \stackrel{\text{def}}{=} \{\sigma \mid \mathcal{A} \models \sigma \text{ for all } \mathcal{A} \in \mathcal{M}\}$$

The most common way in which the second definition is used is to talk about the theory defined by a single model, i.e., $\mathcal{M} = \{\mathcal{A}\}$, written just as $Th(\mathcal{A})$.

Examples: Number theory (a.k.a. "true FO arithmetic"), Presburger arithmetic.

3 Some of the foundational theorems of FOL

When people designed a proof system they worried about its consistency. Hilbert-Ackermann used semantics to ensure consistency of FOL provability. With hindsight, this method is now embodied in the following sanity check:

Theorem 3.1 (Soundness) *If $\Sigma \vdash \sigma$ then $\Sigma \models \sigma$.*

Corollary 3.2 *FO provability is consistent, i.e., $\nvdash \mathsf{false}$.*

The next property to worry about is whether the proof system is powerful enough to match logical consequence. Answering a question of Hilbert, Gödel showed that this is so.

Theorem 3.3 (Gödel's Completeness Theorem) *If $\Sigma \models \sigma$ then $\Sigma \vdash \sigma$. Equivalently, in view of soundness, for all Σ we have $Con(\Sigma) = Ded(\Sigma)$.*

The following exercise explains how Gödel bypassed the need for a clean definition of validity.

Exercise 3.1 *Show that VALID $\subseteq$ PROV is equivalent to the following: each sentence is either* refutable *(i.e., its negation is provable) or satisfiable.*

[3] You won't go wrong in CIS 682 if you read "class" as " set".

Here is a way to understand the significance of the Completeness Theorem. What is the computational nature of *VALID*? Going by its definition, how can we "check" truth in *all* models?!? The Completeness Theorem tells that this highly complex concept is equivalent with a much friendlier one, given by proof, and as a consequence:

Corollary 3.4 *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* [4], and again asked by Hilbert, has a negative answer:

Theorem 3.5 (Church/Turing's Undecidability Theorem) *VALID is undecidable.*

See section 4.

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

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

Exercise 3.2 *Give an example of an FO sentence that is finitely valid but not valid.*

Unfortunately, finite validity is also unfriendly! We have:

Theorem 3.6 (Trakhtenbrot's Theorem) *FIN-VALID is undecidable.*

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

The situation with finite satisfiability is quite different:

Proposition 3.8 *The set of finitely satisfiable FO sentences is r.e.*

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

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 *model checking*.) $\Box$

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

Corollary 3.9 *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 3.8. This would make *FIN-VALID* decidable which contradicts Trakhtenbrot's Theorem. $\Box$

Exercise 3.3 *Show that FIN-VALID is co-r.e.-complete.*

Therefore, FOL " in finite models" is worse than standard FOL: it does not even admit a complete proof system! We say that it cannot be *recursively axiomatized*, i.e., expressed as the theory of a decidable set of sentences.

4 Proofs of the Church/Turing and Trakhtenbrot theorems

Here is a proof of Theorem 3.5.

Recall the String Rewriting Problem in my [[**lecture notes on computability**]]. A many-one reduction $K \leq SRP$ is given there, showing that SRP is undecidable. $\Leftarrow$

Here we show that $SRP \leq VALID$.

We will describe the total computable function that performs the reduction $SRP \leq VALID$ by the informal algorithm that computes it. This algorithm will take an input of the form $\langle R, u, v\rangle$ where R is a finite set of string rewrite rules over an alphabet Σ and u, v are strings over Σ and produce an output of the form φ. In order for this to be a many-one reduction, we need $u \overset{R}{\longrightarrow\!\!\!\!\!\rightarrow} v$ iff $\models \varphi$.

First, the reduction will have to construct the first-order language over which φ is built. This will contain a binary a binary predicate symbol, *Rew*. Moreover, let Σ be the alphabet over which the strings in R and u, v are built. For each $\sigma \in \Sigma$, the first-order language will contain a unary function symbol f_σ. Finally, we have a constant symbol c. From each rewrite rule $r \in R$,

$$r \ : \ \sigma_1 \cdots \sigma_m \longrightarrow \tau_1 \cdots \tau_n$$

the reduction will construct a sentence

$$\Phi_r \ \stackrel{\text{def}}{=} \ \forall x \ Rew(f_{\sigma_1}(\cdots f_{\sigma_m}(x)\cdots) \, , \, f_{\tau_1}(\cdots f_{\tau_n}(x)\cdots))$$

and then, if $R = \{r_1, \ldots, r_k\}$, the reduction will construct

$$\Phi_R \ \stackrel{\text{def}}{=} \ \Phi_{r_1} \wedge \cdots \wedge \Phi_{r_k}$$

Moreover, if $u \equiv \delta_1 \cdots \delta_p$ and $v \equiv \varepsilon_1 \cdots \varepsilon_q$, the reduction will construct

$$\Phi_{u,v} \stackrel{\text{def}}{=} Rew(f_{\sigma_1}(\cdots f_{\sigma_m}(c)\cdots), f_{\tau_1}(\cdots f_{\tau_n}(c)\cdots))$$

We also need to say something that ensures that the meaning of the predicate symbol *Rew* always "simulates" a rewriting relation. Taking

$$\Phi_{Rew} \stackrel{\text{def}}{=} (\forall x\ Rew(x,x)) \wedge (\forall x,y,z\ Rew(x,y) \wedge Rew(y,z) \rightarrow Rew(x,z)) \wedge$$
$$\wedge \bigwedge_{\sigma\in\Sigma} (\forall x,y\ Rew(x,y) \rightarrow Rew(f_\sigma(x), f_\sigma(y)))$$

the reduction will finally construct

$$\varphi \stackrel{\text{def}}{=} \Phi_R \wedge \Phi_{Rew} \rightarrow \Phi_{u,v}$$

This is clearly defining a total computable function. It remains to show that $u \stackrel{R}{\longrightarrow\!\!\!\!\!\rightarrow} v$ iff $\models \varphi$.

Claim 1. $u \stackrel{R}{\longrightarrow\!\!\!\!\!\rightarrow} v \Rightarrow \models \varphi$

Proof of Claim 1. By soundness, it is sufficient to show $u \stackrel{R}{\longrightarrow\!\!\!\!\!\rightarrow} v \Rightarrow \vdash \varphi$. This is shown by induction on the length of the rewriting sequence from u to v. More precisely, one can show that

- for any $w \in \Sigma^*$, $\vdash \Phi_R \wedge \Phi_{Rew} \rightarrow \Phi_{w,w}$
- for any $r \in R, w_1, w_2 \in \Sigma^*$, if $w_1 \stackrel{r}{\longrightarrow} w_2$ then $\vdash \Phi_R \wedge \Phi_{Rew} \rightarrow \Phi_{w_1,w_2}$, and
- for any $w_1, w_2, w_3 \in \Sigma^*$, if $\vdash \Phi_R \wedge \Phi_{Rew} \rightarrow \Phi_{w_1,w_2}$ and $\vdash \Phi_R \wedge \Phi_{Rew} \rightarrow \Phi_{w_2,w_3}$ then $\vdash \Phi_R \wedge \Phi_{Rew} \rightarrow \Phi_{w_1,w_3}$.

The details are omitted. **End of Proof of Claim 1.**

Claim 2. $\models \varphi \Rightarrow u \stackrel{R}{\longrightarrow\!\!\!\!\!\rightarrow} v$

Proof of Claim 2. Since $\models \varphi$ then $\mathcal{R} \models \varphi$ where $\mathcal{R}$ is the following structure: the underlying set is Σ^*, *Rew* is interpreted as the rewrite relation $\stackrel{R}{\longrightarrow\!\!\!\!\!\rightarrow}$, f_σ is interpreted as the function that concatenates σ at the beginning of its argument, and c is interpreted as nil, the empty string. Clearly $\mathcal{R} \models \Phi_R \wedge \Phi_{Rew}$. Since we also have $\mathcal{R} \models \varphi$, it follows that $\mathcal{R} \models \Phi_{u,v}$ hence $u \stackrel{R}{\longrightarrow\!\!\!\!\!\rightarrow} v$. **End of Proof of Claim 2.**

End of proof.

Note that the first-order language needs to contain at least one binary predicate symbol, one constant symbol, and a number of unary function symbols for the previous proof to go through. It turns out that the number of function symbols can be reduced to two, since the String Rewrite Problem over the alphabet $\{0,1\}$ is undecidable (to see this, recall that strings over an arbitrary alphabet can be encoded as strings of bits).

We now turn to Trakhtenbrot's Theorem (Theorem 3.6). 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 K, K_ϵ is undecidable, (in fact, it is r.e.-complete, just like K). 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 4.1 *Show how to modify the proof of Trakhtenbrot's Theorem in Libkin's book in order to prove the Church-Turing Theorem.*

Exercise 4.2 *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.*