ITCS 532:
5. The Entscheidungsproblem

Rob Egrot

The Entscheidungsproblem

- We are now in a position to answer the question that we started with.
- That is, Hilbert's Entscheidungsproblem.
- Given a set of first-order sentences Γ, and another sentence ϕ, does $\Gamma \models \phi$?
- Is there an algorithm that will always give us the correct answer?
- This was the problem that Turing invented Turing machines to settle.
- The answer turns out to be 'no'.
- We will prove this in this class.

First-Order Logic - Languages

Definition 1 (first-order language)

- A first-order language $\mathscr{L}$ is a union of three disjoint sets R, F, and C.
- Every element of R and F is associated with a natural number $n \geq 1$ known as its *arity*.
- Symbols in R represent n-ary relations.
- Symbols in F represent n-ary functions.
- Symbols in C represent constants.

First-Order Logic - Examples

Example 2 (graphs)

The first-order language of graphs contains a single binary relation symbol E. We want $E(x, y)$ to represent the existence of an edge connecting x and y.

Example 3 (arithmetic)

We could create a first-order language for arithmetic containing two binary function symbols $+$ and $\times$, and two constant symbols 0 and 1. These symbols are intended to have their usual meanings.

First-Order Logic - Logical Symbols

- To build statements in first-order logic we not only need a first-order language but also a set of logical symbols.
- We will use the following symbols:

$$\{\forall, \exists, \neg, \vee, \wedge, \rightarrow, =\}$$

- This is a maximalist approach, in the sense that we could use a smaller set of symbols if we wanted.
- E.g. $\{\neg, \vee\}$ has the same expressive power as $\{\neg, \vee, \wedge, \rightarrow\}$.
- Also, e.g. $\exists$ can be written as $\neg\forall\neg$.
- We also have the brackets (and) which we use to arrange our statements so that they can be read unambiguously.
- We also have a countably infinite pool of variables $\{x_0, x_1, x_2, \ldots\}$.

First-Order Logic - Terms and Atomic Formulas

We build up statements (which we usually call *formulas*) recursively using the following rules:

- A *term* is any constant c or variable x, and everything of form $f(t_1, \ldots, t_n)$ where f is an n-ary function symbol and $t_1, \ldots, t_n$ are all terms.
- If $t_1, \ldots, t_n$ are terms and r is an n-ary relation symbol then the following are *atomic formulas*:
  - $t_1 = t_2$,
  - $r(t_1, \ldots, t_n)$.

First-Order Logic - Formulas

We can now define *formulas* for a given language $\mathscr{L}$:

- Every atomic formula is a formula.
- If ϕ and ψ are formulas and x is a variable then:
  - $\neg\phi$ is a formula.
  - $(\phi \vee \psi)$ is a formula.
  - $(\phi \wedge \psi)$ is a formula.
  - $(\phi \rightarrow \psi)$ is a formula.
  - $\forall x \phi$ is a formula.
  - $\exists x \phi$ is a formula.

We may add or omit brackets to make formulas easier to read.

First-Order Logic - Free and Bound Variables

- If an occurrence of a variable x occurs in a formula ϕ we say that it is *bound* if it appears in the scope of a quantifier.
- I.e. an occurrence of x in ϕ is bound if it appears inside some subformula ψ of ϕ and either $\forall x\psi$ or $\exists x\psi$ is also a subformula of ϕ.
- If an occurrence of x in ϕ is not bound then we say it is *free*.
- When a formula contains no free occurrences of variables we say it is a *sentence*.
- In this course we are only interested in first-order sentences.

Example 4

Let $\phi = \forall x(r(x, y)) \to (x = y \wedge \forall z(f(z) = x))$. Then z only occurs bound in ϕ, while y occurs only free. On the other hand x occurs both bound and free.

First-Order Logic - Deduction and Consistency

- There are various equivalent systems for defining deduction in first-order logic.
- Deduction rules formalize the intuitive idea that some things are logical consequences of others.
- We write $\Gamma \vdash \phi$ if ϕ is provable from Γ.
- We say a set of sentences Γ is *inconsistent* if there is ϕ such that $\Gamma \vdash \phi$ and $\Gamma \vdash \neg\phi$.
- If there is no such ϕ then Γ is *consistent*.
- A set of consistent sentences in a language $\mathscr{L}$ is called a *theory* for $\mathscr{L}$.
- We sometimes call the sentences in a theory *axioms*.
- We use theories to specify the kinds of structures we are interested in.

First-Order Logic - Models and Semantics

- The goal of first-order logic is to use formal methods to reason about real mathematical systems.
- To do this we need a way to assign meaning to sentences using mathematical structures.
- Given $\mathscr{L} = R \cup F \cup C$ we define a *model* M for $\mathscr{L}$ to be a set X and appropriate relations, functions and elements of X.
- Sentences from $\mathscr{L}$ will then be true or false in M.
- When ϕ is true in M we say M *satisfies* ϕ and write $M \models \phi$.
- If for all models M we have $M \models \phi \implies M \models \psi$ then we write $\phi \models \psi$. We say ψ is a *logical consequence* of ϕ.
- If Γ is a set of sentences and $M \models \Gamma \implies M \models \phi$ for all models M of $\mathscr{L}$ then we write $\Gamma \models \phi$.
- If Γ is a theory and $M \models \Gamma$ we say M is a model of Γ.
- If a sentence ϕ is true in all models of $\mathscr{L}$ we say it is *valid*, and if it is true in at least one model of $\mathscr{L}$ we say it is *satisfiable*.

First-Order Logic - Soundness and Completeness

Theorem 5 (soundness)

If $\Gamma \cup \{\phi\}$ *is a set of* $\mathscr{L}$*-sentences and* $\Gamma \vdash \phi$ *then* $\Gamma \models \phi$*.*

Theorem 6 (completeness)

If $\Gamma \cup \{\phi\}$ *is a set of* $\mathscr{L}$*-sentences and* $\Gamma \models \phi$ *then* $\Gamma \vdash \phi$*.*

Corollary 7

A first-order theory is consistent if and only if it has a model.

Corollary 8

ϕ *is valid if and only if* $\neg\phi$ *is not satisfiable.*

First-Order Logic - The Peano Axioms

- The Peano axioms define the natural numbers and their arithmetic properties in first-order logic.
- The language of Peano arithmetic (PA) is based on the signature $(0, s, +, \times)$.
- Here 0, $+$ and $\times$ are meant to correspond to their usual roles in arithmetic, and s is meant to be the successor function.
- We don't actually need the functions $+$ and $\times$, as they can be defined using s.
- A simple fragment of the first-order theory for Peano arithmetic is defined by the following axioms:
  1. $\forall n \neg(0 = s(n))$.
  2. $\forall m \forall n((s(m) = s(n)) \to (m = n))$.
- Call the theory defined by these axioms S.
- The full theory PA also has axioms for $+$ and $\times$, and also an infinite set of axioms formalizing the principle of mathematical induction.
- We don't need that here.

First-Order Logic - Models of S

- Whenever we have a first-order theory one of the most basic questions we can ask is whether it has a model.
- In this case, the natural numbers $\mathbb{N}$ with the obvious interpretations of s and 0 form a model for S.
- Are the natural numbers the only model for S?
- No. E.g., take the natural numbers $\mathbb{N}$ and a disjoint copy of the natural numbers $\mathbb{N}'$.
- Then we can interpret 0 as zero in $\mathbb{N}$, and we can interpret the successor function naturally in $\mathbb{N}$ and $\mathbb{N}'$ separately (i.e. $s(n) = n + 1$ for $n \in \mathbb{N}$ and $s(n') = n' + 1$ for $n' \in \mathbb{N}'$).
- It's easy to check that this is also a model for S.
- We can construct alternative models to the full theory of PA too, though this is technically more difficult.

First-Order Logic - Intended Models

- The fact that there are multiple models for PA reveals a more general phenomenon.
- It is actually very rare for a set of axioms to have only one model.
- This can lead to confusion as there is often an *intended model*, i.e. the model we intend the axioms to describe.
- If there are other models it means that not every property of our intended model has been captured by our axioms.
- Sometimes no recursively enumerable first-order theory has the structure of interest as its only model.
- Gödel's first incompleteness theorem proves this for arithmetic.
- While there are other models for PA other than $\mathbb{N}$, PA is still special in that every model for PA must contain an isomorphic copy of $\mathbb{N}$.
- So we can think of $\mathbb{N}$ as being a kind of minimal model for PA.

First-Order Logic - Incompleteness

Related to the concept of non-standard models of arithmetic we have Gödel's famous *incompleteness* theorems:

Theorem 9 (Gödel's first incompleteness theorem)

If Γ *is a consistent and recursively enumerable set of* $\mathscr{L}$*-sentences such that* Γ *defines a theory powerful enough to do elementary arithmetic, then there is an* $\mathscr{L}$*-sentence* ϕ *(the Gödel sentence) such that* $\Gamma \nvdash \phi$ *and* $\Gamma \nvdash \neg\phi$*.*

Theorem 10 (Gödel's second incompleteness theorem)

If Γ *is a consistent and recursively enumerable set of* $\mathscr{L}$*-sentences such that* Γ *defines a theory powerful enough to do elementary arithmetic, and if* $\mathbf{cons}(\Gamma)$ *is the* $\mathscr{L}$*-sentence expressing the consistency of* Γ*, then* $\Gamma \nvdash \mathbf{cons}(\Gamma)$*.*

The Entscheidungsproblem Revisited

- A decision problem: Given a set of first-order sentences Γ, and another sentence ϕ, does $\Gamma \models \phi$?
- Is this decidable?
- No.
- We will see why using Turing machines, reduction, and the Empty Tape Halting Problem.
- We will define a special first-order language for talking about Turing machines.
- Then we will reduce instances of ETHP to instances of the Entscheidungsproblem.
- It follows that the Entscheidungsproblem is undecidable.

A Language for Turing Machines - Outline

- The idea is to use first-order logic to describe the state of the tape at position x and time y for all $x, y \in \mathbb{N}$.
- This is kind of like a 2-dimensional grid. The rows of the grid represent the state of the state of the tape at each step in the computation.
- The symbols in each square, the state of the machine, and the position of the tape head will be represented by predicates.
- We will use first-order sentences in this language to model the transition function and make sure the grid represents the operation of the Turing machine correctly.

A Language for Turing Machines - Defining $\mathscr{L}$

Given a Turing machine $T = (Q, \Sigma, q_0, H, \delta)$ with $Q = \{q_0, \ldots, q_m\}$, $\Sigma = \{\sigma_0, \ldots, \sigma_n\}$, and $H = \{q_m\}$ we define a language $\mathscr{L}$ as follows:

- $\mathscr{L}$ contains the following predicates:
  - One binary predicate σ for every symbol $\sigma \in \Sigma$. The idea is that $\sigma(x, y)$ holds if σ is written on the tape at position x at step y of the computation.
  - A binary predicate h. The idea is that $h(x, y)$ holds if the tape head is at position x at step y of the computation.
  - One unary predicate q for every state $q \in Q$. We will use this to represent the state of the machine. I.e. $q(y)$ holds if the machine is in state q at step y of the computation.
- $\mathscr{L}$ contains a single function symbol s. The idea is that s is a successor function. I.e. we want $s(x)$ to be $x + 1$. We will use this to control how the machine transitions.
- $\mathscr{L}$ contains a single constant symbol 0. This is used to represent the starting point for our grid's number system.

A Language for Turing Machines - the State of the Tape

With $\sigma_0 =:$, and $\sigma_1 = \llcorner\!\lrcorner$ we define the following $\mathscr{L}$-sentences to describe the state of the tape:

- $T_0 = \forall x \forall y \big(\bigvee_{i=0}^{n} \sigma_i(x, y)\big) \wedge \forall x \forall y \big(\bigwedge_{i \neq j}^{n} (\sigma_i(x, y) \rightarrow \neg \sigma_j(x, y))\big)$.
  This sentence is supposed to guarantee that at every step of the computation there is one and only one symbol written in every space of the tape (possibly the blank symbol).
- $T_1 = \forall x \forall y \big((\sigma_0(x, y)) \leftrightarrow (x = 0)\big)$.
  This sentence is to make sure that at every step of the computation the symbol : is written in the first square of the tape, and nowhere else.
- $T_2 = \forall x (x \neq 0 \rightarrow \sigma_1(x, 0))$.
  This is to make sure that at step 0 every square of the tape other than square 0 contains the blank symbol.

A Language for Turing Machines - Basic Behaviour

We define sentences corresponding to the basic operation of T.

- $S_0 = q_0(0)$.
  This is to ensure that the machine starts in the start state.
- $S_1 = \forall y \Big(\big(\bigvee_{i=0}^{m} q_i(y) \big) \wedge \big(\bigwedge_{i \neq j}^{m} (q_i(y) \rightarrow \neg q_j(y)) \big) \Big)$.
  This says that at every step in the computation the machine must be in exactly one state.
- $H_0 = h(0, 0)$.
  This makes sure the tape head starts at square 0.
- $H_1 = \forall y \exists x \big(h(x, y) \big) \wedge \forall y \forall x \forall z \big((h(z, y) \wedge h(x, y)) \rightarrow (z = x) \big)$.
  This says that at every step the tape head is at exactly one place on the tape.
- $F = \forall x \forall y \Big(\bigwedge_{i=1}^{n} \big((\neg h(x, y) \wedge \sigma_i(x, y)) \rightarrow \sigma_i(x, s(y)) \big) \Big)$.
  This says that symbols on the tape can only change when the tape head acts on them.

A Language for Turing Machines - the Transition Function

Now we deal with the transition function δ.

- For every tuple $t = (q, \sigma, q', \sigma')$ in δ such that $\sigma' \notin \{\leftarrow, \rightarrow\}$ we get a sentence F_t defined by

$$F_t = \forall x \forall y \Big(\big(q(y) \wedge h(x,y) \wedge \sigma(x,y)\big) \rightarrow \big(q'(s(y)) \wedge \sigma'(x, s(y)) \wedge h(x, s(y))\big)\Big).$$

This controls ‘writing’ instructions.

- For every tuple $t = (q, \sigma, q', \sigma')$ in δ such that $\sigma' = \leftarrow$ we get a sentence F_t defined by

$$F_t = \forall x \forall y \Big(\big(q(y) \wedge h(x,y) \wedge \sigma(x,y)\big) \rightarrow \big(q'(s(y)) \wedge \sigma(x, s(y)) \wedge h(p(x), s(y))\big)\Big).$$

This controls ‘move left’ instructions.

- For every tuple $t = (q, \sigma, q', \sigma')$ in δ such that $\sigma' = \rightarrow$ we get a sentence F_t defined by

$$F_t = \forall x \forall y \Big(\big(q(y) \wedge h(x,y) \wedge \sigma(x,y)\big) \rightarrow \big(q'(s(y)) \wedge \sigma(x, s(y)) \wedge h(s(x), s(y))\big)\Big).$$

This controls ‘move right’ instructions.

A Language for Turing Machines - Final Details

Now some final details.

- Recall the halting state is q_m. We define $\phi = \forall y(\neg q_m(y))$. This sentence says that the computation does not halt.
- Finally we take a first-order sentence P demanding that our variables follow the selection S of Peano axioms from earlier. I.e.

  1. $\forall n \neg(0 = s(n))$.
  2. $\forall m \forall n((s(m) = s(n)) \to (m = n))$.

The Theorem

Theorem 11

There is no algorithm that can take a first-order sentence and decide whether it has a model.

Proof outline

- We consider the complement of this problem.
- I.e. yes instances are first-order sentences with no model and no instances are those with a model.
- This is equivalent because a decision problem is decidable if and only if its complement is decidable.
- We prove the complement problem is undecidable by reducing the empty tape halting problem to it.

The Argument

- Given a Turing machine M let
$$\phi_M = T_0 \wedge T_1 \wedge T_2 \wedge S_0 \wedge S_1 \wedge H_0 \wedge H_1 \wedge F \wedge \bigwedge_{t \in \delta} F_t \wedge \phi \wedge P$$
- We can construct ϕ_M (in coded form) by checking **code**(M).
- We must prove that M halts (on empty input) if and only if ϕ_M has no model.
- Suppose first that M halts.
- Any model of ϕ_M must contain a model of the infinite grid structure we use to represent the state of the tape and TM at every step of the computation.
- This is because $\mathbb{N}$ is a minimal model for S.
- The state of this $\mathbb{N} \times \mathbb{N}$ grid is completely determined by the code of the machine.
- If M reaches a halt state then we will have $q_m(y)$ for some $y \in \mathbb{N}$.
- But this would contradict ϕ and thus ϕ_M. We conclude that if M halts then ϕ_M has no model.

The Argument Continued

- Conversely, suppose M does not halt.
- Then we can represent its run on an $\mathbb{N} \times \mathbb{N}$ grid.
- Since M does not halt, every row of this grid will be well defined and we will never have $q_m(y)$.
- This grid is a model for ϕ_M using the intended interpretations of the symbols from the language of ϕ_M.
- So the conversion $M \mapsto \phi_M$ is a reduction, and the complement decision problem, so also the original decision problem, is undecidable.

Conclusion

Corollary 12

The Entscheidungsproblem is not decidable.

Proof.

- Suppose we have an algorithm for deciding whether $\Gamma \models \phi$ for arbitrary Γ and ϕ.
- We will show we could use this to decide whether an arbitrary sentence ψ has a model, contradicting theorem 11.
- Now, ψ has no models (i.e. ψ is not satisfiable) if and only if $\neg\psi$ is true in all models (i.e. if $\neg\psi$ is valid), by corollary 8.
- And $\neg\psi$ is valid if and only if $\emptyset \models \neg\psi$, by definition of validity.
- So, assuming we have an algorithm for deciding whether $\Gamma \models \phi$ for arbitrary Γ and ϕ, we can decide if $\neg\psi$ is valid, and thus whether ψ has a model.
- This produces a contradiction, as claimed.

$\square$