The Decision Problem for FOPC

The decision problem for first-order logic is:

> Given an arbitrary set Σ of first-order formulae, and a first-order formula ϕ, determine whether or not $\Sigma \models \phi$.

This is Hilbert's *Entscheidungsproblem*.

What is wanted is an algorithm for the decision problem. Equivalently:

Is logical consequence in first-order logic a decidable relation?

We shall show that this problem is *equivalent* to the Halting Problem for Turing machines.

Turing Machine Definition

The machine has tape alphabet {0,1}, and its states are numbered 0, 1, 2, ..., n, where 1 is the start state and 0 is the halt state. The machine is fully defined by three functions T, U, D.

Given current state q and currently scanned symbol a,

$T : \mathbb{N} \times \{0,1\} \to \mathbb{N}$ determines the next state.
If no transition for (q,a), put $T(q,a) = q$.

$U : \mathbb{N} \times \{0,1\} \to \{0,1\}$ determines the new symbol.
If no transition for (q,a), put $U(q,a) = a$.

$D : \mathbb{N} \times \{0,1\} \to \{-1,0,1\}$ determines the displacement.
If no transition for (q,a), put $D(q,a) = 0$.

Turing Machine Configuration

$$\cdots 000 b_n \cdots b_3 b_2 b_1 \underset{q}{\boxed{a}} c_1 c_2 c_3 \cdots c_m 000 \cdots$$

A *configuration* C for the machine M can be specified by means of four items:

$q_C \in \mathbb{N}$, the current state;

$a_C \in \{0,1\}$, the currently scanned symbol;

$b_C = b_1 b_2 \ldots b_n$
$c_C = c_1 c_2 \ldots c_m$
} the strings of symbols running left/right of currently scanned square as far as leftmost/rightmost '1'.

If $q_C = 1$ then C is an *initial configuration*.
If $q_C = 0$ then C is a *halting configuration*.

Turing Machine Operation I

When started in an initial configuration $C(0)$, the machine runs through a sequence of configurations

$$C(0), C(1), C(2), C(3), C(4), C(5), \ldots$$

C is a function

$$C : \mathbb{N} \to \mathcal{C} = \mathbb{N} \times \{0,1\} \times \{0,1\}^* \times \{0,1\}^*.$$

$C(i+1)$ is determined from $C(i)$ by means of the functions T, U, D defining the operation of the Turing machine. (Details on next slide.)

If no transitions are defined for $(q_{C(i)}, a_{C(i)})$, then all later configurations after $C(i)$ will be the same as $C(i)$.

In particular this will hold if $C(i)$ is a halting configuration.

Turing Machine Operation II

Let $C(i) = (q, a, b, c)$, where $b = b_1 b_2 \ldots b_n$ and $c = c_1 c_2 \ldots c_m$.

Then $C(i+1) = (q', a', b', c')$, where

$q' = T(q,a)$

$$a' = \begin{cases} b_1 & (\text{if } D(q,a) = -1) \\ U(q,a) & (\text{if } D(q,a) = 0) \\ c_1 & (\text{if } D(q,a) = 1) \end{cases}$$

$$b' = \begin{cases} b_2 \ldots b_n & (\text{if } D(q,a) = -1) \\ b & (\text{if } D(q,a) = 0) \\ U(q,a) b_1 \ldots b_n & (\text{if } D(q,a) = 1) \end{cases}$$

$$c' = \begin{cases} U(q,a) c_1 \ldots c_m & (\text{if } D(q,a) = -1) \\ c & (\text{if } D(q,a) = 0) \\ c_2 \ldots c_m & (\text{if } D(q,a) = 1) \end{cases}$$

C is a computable function.

The Halting Problem

Given a configuration C, the value of q_C determines whether or not it is a halting configuration.

Let the formula $H(n)$ say that when machine M is started in configuration $C(0)$ it will be in a halting configuration after n steps.

Then H is a decidable predicate:
We can reliably determine, for any Turing machine and any initial tape, whether or not it has reached the halt state after any given number of steps.

But we cannot in general determine, for any Turing machine and any initial tape, whether or not it *ever* reaches the halt state.

How did Turing show that the Entscheidungsproblem is unsolvable?

Turing showed that a certain class of inferences in first-order logic was equivalent to the halting problem for Turing machines.

For each inference in the class, validating it is equivalent to determining whether or not a particular Turing machine halts when given a particular input tape.

This means that there can be no decision procedure for that class of inferences, and hence for first-order logic in general, since any such decision procedure would enable us to solve the Halting problem.

A first-order language for describing Turing Machines

We use the following non-logical vocabulary:

Constants:	0, 1, -1, and nil.
Function symbols:	$suc, head, tail, cons,$ $config, T, U, D, C.$
Predicate:	H.

Use suc with 0 to generate labels for machine states q.

Use nil, $head$, $tail$ and $cons$ to construct strings b and c.

Use $config$ to construct a configuration out of its components.

T, U, D, C, H are as before.

Use lower-case letters for variables $(a, b, c, q, \ldots)$.

Premisses I

(1) A set of $6n$ formulae defining a particular Turing machine with n non-halting states:

$$T(q,a) = q'$$
$$U(q,a) = a'$$
$$D(q,a) = d$$

for $q = suc(0), \ldots, suc^n(0)$ and $a = 0, 1$, where $q' \in \{0, suc(0), \ldots, suc^n(0)\}$, $a' \in \{0,1\}$, and $d \in \{-1, 0, 1\}$.

(2) A formula giving the initial configuration, of the form

$$C(0) = config(1, a, b, c)$$

where a is 0 or 1, and both b and c are strings constructed from $cons$, 0, and 1.

Premisses II

(3) Three formulae defining transitions between machine configurations, one formula for each displacement:

$$\begin{aligned}\forall q,a,b,c,d,i[\ & C(i) = config(q,a,b,c) \wedge \\ & D(q,a) = -1 \ \rightarrow \\ & C(suc(i)) \ = \\ & config(T(q,a), head(b), tail(b), cons(U(q,a), c))]\end{aligned}$$

$$\begin{aligned}\forall q,a,b,c,d,i[\ & C(i) = config(q,a,b,c) \wedge \\ & D(q,a) = 0 \ \rightarrow \\ & C(suc(i)) \ = \\ & \quad config(T(q,a), U(q,a), b, c))]\end{aligned}$$

$$\begin{aligned}\forall q,a,b,c,d,i[\ & C(i) = config(q,a,b,c) \wedge \\ & D(q,a) = 1 \ \rightarrow \\ & C(suc(i)) \ = \\ & config(T(q,a), head(c), cons(U(q,a), b), tail(c))]\end{aligned}$$

Premisses III

(4) Some general rules to ensure correct behaviour of $suc, head, tail, cons$:

$$\forall x(\neg suc(x) = 0)$$
$$\forall x, y(suc(x) = suc(y) \rightarrow x = y)$$
$$\forall x, y(head(cons(x,y)) = x)$$
$$\forall x, y(tail(cons(x,y) = y)$$
$$head(nil) = 0$$
$$tail(nil) = nil$$

(5) A formula defining what it is for the computation to halt after a given number of steps:

$$\forall i(H(i) \leftrightarrow \exists a, b, c\ C(i) = config(0, a, b, c)).$$

The punch-line

The premisses we have listed completely define the computation performed by a given machine with a given starting configuration. (And such a set of premisses can be set up for any machine/configuration pair.)

The computation either halts or it does not, so exactly one of the formulae $\exists i H(i)$ and $\neg\exists i H(i)$ must be a logical consequence of the premisses.

Suppose there is a decision procedure for first-order logic. Then this procedure could be used to determine which of these two cases holds—so we could solve the Halting Problem.

But we can't, so there isn't.

All is not lost . . .

Although the general decision problem is insoluble, there are many classes of formulae which *are* decidable, for example:

- formulae containing only one-place predicates;
- formulae whose prenex form contains only existential quantifiers;
- formulae whose prenex form does not have an existential quantifier in front of a universal quantifier;
- formulae whose prenex form does not contain more than one existential quantifier

Examples of Prenex Form

$\forall x(P(x) \rightarrow \exists y Q(x,y))$
$\forall x \exists y(P(x) \rightarrow Q(x,y))$

$\forall x P(x) \wedge \forall x Q(x)$
$\forall x(P(x) \wedge Q(x))$

$\exists x P(x) \wedge \exists x Q(x)$
$\exists x \exists y(P(x) \wedge Q(y))$

$\exists x P(x) \vee \exists x Q(x)$
$\exists x(P(x) \vee Q(x))$

$\forall x P(x) \vee \forall x Q(x)$
$\forall x \forall y(P(x) \vee Q(y))$

$\exists x P(x) \rightarrow \forall x Q(x)$
$\forall x \forall y(P(x) \rightarrow Q(y))$

$\forall x P(x) \rightarrow \exists y Q(y)$
$\exists x \exists y(P(x) \rightarrow Q(y))$