PREDICATE LOGIC: UNDECIDABILITY
HUTH AND RYAN 2.5, SUPPLEMENTARY NOTES

(Source: Neil D. Jones, DIKU, September 2005, Nils Anderson 2004 - minor adaptations by WMB)

Outline

- Gödel's completeness theorem
- Undecidability
  - Computability
  - Problem reduction
  - Post's correspondence problem
  - Undecidability of predicate logic
- Implications for the deductive paradigm

SOUNDNESS AND COMPLETENESS

Let M be a model (interpretation) for predicate calculus formulas.

Definition Formula φ is *valid* iff M |= φ holds for every model M.

Definition

- Proof system |- is *sound* iff for any closed predicate formula φ we have:

  |- φ implies M |= φ for every model M

  In brief: any provable formula is valid.

- Proof system |- is *complete* iff for any closed predicate formula f :

  |- φ if M |= φ for every model M

  In brief: any valid formula is provable.

Remark: Validity is a very strong condition to place on a formula: φ must hold *in all models*.
In contrast to most mathematical reasoning: about *one model at a time*.

GÖDEL'S COMPLETENESS THEOREM

The proof system |- described in Huth and Ryan's book is both sound and complete:

> If φ is a closed predicate formula, then |- φ iff it is valid, i.e., M |= φ for every model M

THE DECIDABILITY PROBLEM FOR PREDICATE LOGIC

For a sentence φ, obviously either

- φ is true in all models, or
- φ is false in some model

Now φ is *satisfiable* if M |= φ for some model M, so
φ is unsatisfiable if and only if ¬φ is valid.

But *how can we find out* by some algorithmic procedure whether or not |-φ in predicate logic?

The **truth-table** method works for propositional logic: enumerate all possible values of propositional variables.

Alas it is no solution for predicate logic, since formula φ will always have infinitely many models, and so infinitely many valuations for individual variables.

Surprise: even though provability and validity are equivalent, *neither one is decidable*.
In other words, there is no algorithmic procedure to determine whether or not |-φ in predicate logic !

DECISION PROBLEMS (An aside added by WMB)

Definition A **decision problem** is a problem of the form "Given an entity x and a class of entities A, determine whether x belongs to A".

Examples of decision problems

1. Given a proposition p decide whether p is a tautology.

2. Given a proposition p in 2-CNF (expressed in CNF in such a way that every clause contains no more than 2 literals) decide whether p is satisfiable.

3. Given a graph G decide whether G can be vertex-coloured using 3 colours.

4. Given a computer program P and an input t decide whether P acting on input t terminates.

In all four examples, we can encode instances of the decision problem as the input for a computer program, and try to write a program that will always terminate and correctly output the answer **Y** or **N**.

1. *Given a proposition p to decide whether p is a tautology.* We can write a program to do this, but it won't be very efficient - it needs to check all truth assignments to the propositional atoms in general.

2. *Given a proposition p in 2-CNF (expressed in CNF in such a way that every clause contains no more than 2 literals) decide whether p is satisfiable.* We can write an efficient program to check this.

3. *Given a graph G decide whether G can be vertex-coloured using 3 colours.* We can write a program for this, but as in case 1, so far as we know there is no efficient solution.

4. *Given a computer program P and an input t decide whether P acting on input t terminates.* In this case, there is - provably - no computer program that can give the correct answer in finite time on every instance.

Decision problems 1,2 and 3 are said to be *decidable*, though only 2 is computationally tractable for "large" instances. Decision problem 4 is **undecidable**. It is the 'practical computing' counterpart of the Halting Problem.

COMPUTABLE REDUCTION

Definition For two decision problems A over Σ and B over Δ we say that: *A can be (recursively) reduced to B*, if there is a computable function $f : \Sigma^* \rightarrow \Delta^*$ such that

$$\forall\ x \in \Sigma^* \ (x \in A \Leftrightarrow f(x) \in B)$$

Theorem Assume A can be reduced to B. If B is decidable then so is A. If A is undecidable, then so is B. (Informally: "The decision problem B is at least as difficult as the decision problem A.")

Mother of all problems unsolvable by computer - the **Halting Problem**: deciding, given a Turing machine T and an input x, whether or not T halts on x.

POST'S CORRESPONDENCE PROBLEM

Given: a finite sequence $(s_1,t_1),\ldots, (s_k, t_k)$ of pairs of strings. To decide:

> is there a non-empty sequence $i_1,i_2, \ldots, i_n$ of indices such that
> $s_{i_1}s_{i_2} \ldots s_{i_n} = t_{i_1}t_{i_2} \ldots t_{i_n}$?

Theorem PCP is undecidable: There is no systematic terminating procedure that can reliably determine in finite time whether or not an instance of the PCP has a solution or not.

(This theorem can be proved by reducing the Halting Problem to it. We are only concerned here with the result of the theorem - the proof is omitted. The main ideas that you need to assimilate are set out in the rest of this subsection, which has been adapted from the original source by WMB.)

The theorem asserts that there is no general algorithm that can successfully determine whether or not any given instance of the PCP, as defined by a set of pairs of strings, does or doesn't have a solution. Without loss of generality we may assume that the strings are binary, consisting of 0s and 1s. PCP is undecidable even with this restriction in place.

Note that PCP is a decision problem; but if we **know** the answer is positive, the actual sequence of indices can be found after a finite systematic search. If we don't know in advance whether there is or isn't a solution to an instance of the PCP, we can initiate a systematic exhaustive search, but we don't in general know whether it will ever terminate. The systematic search procedure doesn't qualify as an algorithm to solve the decision problem, since it doesn't terminate when there is no solution to the given instance of the PCP, and this means that the search procedure is too weak to give a definite Y/N answer in finite time. There is a close parallel here with the Halting Problem - the systematic procedure for executing the Turing Machine on an input illustrates precisely the same problem!

Examples of problem instances:

C = ((1, 101), (10, 00), (011, 11))

and the challenge

D = ((001, 0), (01, 011), (01, 101), (10, 001))

Illustration for the PCP (WMB): In the case of the instance C of the PCP above, the correct answer is **Y**. To see this, note that k = 3, and

C = ((s_1=1, t_1=101), (s_2=10, t_2=00), (s_3=011, t_3=11))

If we now look at the sequences
$s_1s_3s_2s_3$ and $t_1t_3t_2t_3$
they turn out to give the same string, viz. 101110011. To see how the string has been composed from the corresponding s and t substrings, we can highlight the contributions made from the different (s,t) pairs:

1 ***0 1 1*** 1 0 ***0 1 1*** = s_1 ***s_3*** s_2 ***s_3***

1 0 1 ***1 1*** 0 0 ***1 1*** = t_1 ***t_3*** t_2 ***t_3***

We say that the instance C of the PCP has the solution {1,3,2,3}.

The purpose of the examples is to help you get familiar with the nature of the PCP. The fact that for certain specific instances of the PCP it is easy to find a solution, and that for others you may be able to prove that there is none, doesn't detract from the fact there is no systematic terminating procedure that can be applied in general. The parallel with the Halting Problem is again helpful here: it is of course possible to show that certain specific programs will terminate and that others will not, but this doesn't resolve the halting issue, which concerns the existence of a general decisive algorithmic technique covering **all** programs.

PREDICATE LOGIC IS UNDECIDABLE

Proof concept: Reduce PCP (over the alphabet {0, 1}) to the decision problem for predicate logic. The "reduction", i.e., translation, takes an instance C for the PCP

$((s_1,t_1),\ldots, (s_k, t_k))$

to the formula φ:

$$P (f_{s_1}(e), f_{t_1}(e)) \wedge \ldots \wedge P (f_{s_k}(e), f_{t_k}(e))$$
$$\wedge \forall v \forall w (P (v, w) \rightarrow P (f_{s_1}(v), f_{t_1}(w)) \wedge \ldots \wedge P (f_{s_k}(v), f_{t_k}(w)))$$
$$\rightarrow \exists z P (z, z)$$

where - if s is the binary string $b_0b_1 \ldots b_k$, then $f_s \equiv f_{b_0b_1\ldots b_k}(x)$ abbreviates $f_{b_k}(\ldots (f_{b_1}(f_{b_0}(x))) \ldots)$

(This formula uses a binary predicate symbol P, constant e and two unary function symbols f_0 and f_1.)

To complete the proof, it suffices to show that the instance C of PCP has a solution iff the formula φ is true in all models (see H&R p133-136).

IMPLICATIONS FOR THE DEDUCTIVE PARADIGM

- *We know:*

  If φ is a closed predicate formula, then |-φ iff M |= φ for every model M.

- *We also know:*

  The question |-φ is undecidable.

Consequence: There exists no perfect theorem-proving system for predicate logic.

Note also that:

- The set of true (valid) formulas can be enumerated: just enumerate all possible proofs using, for example, Huth and Ryan's proof system.

- But ... there exists no way to enumerate the set of false formulas (else truth would be decidable).
