

Logic and Modelling

— Undecidability and Incompleteness of Predicate Logic —

Jörg Endrullis

VU University Amsterdam

Decidability and Undecidability

Decision Problems: Examples

Prime Problem

Determine whether a **number is prime**:

- ▶ input: a natural number n
- ▶ output: **yes** if n is prime, **no** otherwise

Termination Problem

Decide whether a **program terminates**:

- ▶ input: a program P and input w
- ▶ output: **yes** if P started with input w terminates, **no** else

(Termination means that the program does not run forever.)

Validity Problem

Determine whether a **formula is valid**:

- ▶ input: a formula ϕ of predicate logic
- ▶ output: **yes** if ϕ is valid, **no** otherwise

Decision Problems

A **decision problem** consists of a set I and a predicate $Y \subseteq I$.

Prime Problem

- ▶ $I = \mathbb{N}$
- ▶ $Y = \{ n \in \mathbb{N} \mid n \text{ is prime} \}$

Termination Problem

- ▶ $I =$ set of all pairs $\langle \text{program}, \text{input} \rangle$
- ▶ $Y = \{ \langle P, w \rangle \mid P \text{ terminates on input } w \}$

Validity Problem

- ▶ $I =$ set of formulas of predicate logic
- ▶ $Y =$ set of valid formulas in predicate logic

Can we write a program that on the input of $i \in I$ tells if $i \in Y$?

If such program exists, the predicate Y is called **decidable**.

Decidability

Decidability

A decision problem $Y \subseteq I$ is called **solvable** or **decidable** if there exists a program that tells for every $i \in I$ whether $i \in Y$.

In other words, the program has the following behaviour:

- ▶ input: $i \in I$
- ▶ output: **yes** if $i \in Y$
no if $i \notin Y$

Clearly, the **prime problem** is decidable.

We can write a program deciding whether a number is prime.

Is there such a program for every decision problem?

Can we write a program that decides termination?

Termination Problem (Halting Problem)

Termination

Assume there was a **program** T that decides termination:

- ▶ input: a program P and input w
- ▶ output: **yes** if P terminates on input w , **no** otherwise

Then using T we can create a program T_{self} :

Then there exists a **program** T_{self} with the behaviour:

- ▶ input: a program P
- ▶ output: **yes** if P terminates on input P , **no** otherwise

Note that T_{self} is a special case of T where $w = P$.

Programs started on itself: `/bin/cat /bin/cat`

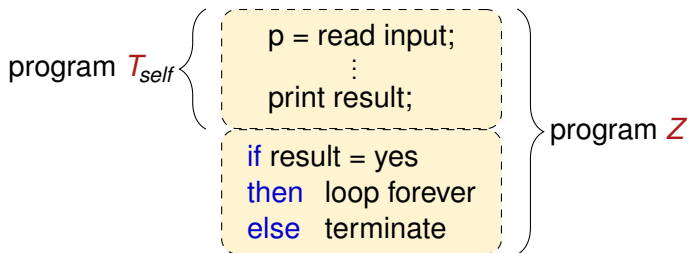
We show that T_{self} does not exist.

Then it follows that T does not exist.

Termination is undecidable!

Assume there would be a **program** T_{self} with the behaviour:

- ▶ input: a program P
- ▶ output: **yes** if P terminates on input P , **no** otherwise



What happens if we run Z with input Z ?

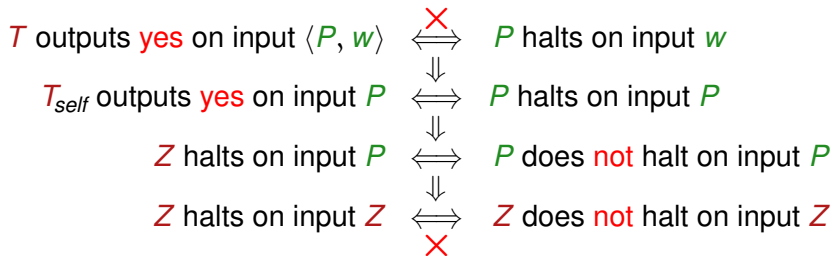
- ▶ initial part T_{self} decides whether Z terminates on input Z
- ▶ if the result is **yes**, then Z runs forever ×
- ▶ if the result is **no**, then Z terminates ×

Thus T_{self} has made a mistake! And thus also T !

Termination is Undecidable (Structure of Argument)

Theorem

The **termination problem** is **unsolvable** (**undecidable**).



Termination is Undecidable (Structure of Argument)

Theorem

The **termination problem** is **unsolvable** (undecidable).

Z halts on input P	$\overset{\times}{\iff}$	P does not halt on input P
	\Downarrow	
Z halts on input Z	$\overset{\times}{\iff}$	Z does not halt on input Z

Compare with **Russell's barber paradox**:

barber shaves x	$\overset{\times}{\iff}$	x does not shave x
	\Downarrow	
barber shaves barber	$\overset{\times}{\iff}$	barber does not shave barber

Post's Correspondence Problem

Post's Correspondence Problem

Post Correspondence Problem (PCP)

Given n pairs of words:

$$\langle w_1, v_1 \rangle, \dots, \langle w_n, v_n \rangle$$

Are there indices i_1, i_2, \dots, i_k ($k \geq 1$) s.t.

$$w_{i_1} w_{i_2} \cdots w_{i_k} = v_{i_1} v_{i_2} \cdots v_{i_k} ?$$

- ▶ $\langle 1, 101 \rangle, \langle 10, 00 \rangle, \langle 011, 11 \rangle$ solution $\langle 1, 3, 2, 3 \rangle$
- ▶ $\langle 110, 0 \rangle, \langle 00, 1 \rangle$ no solution
- ▶ $\langle 1, 111 \rangle, \langle 10111, 10 \rangle, \langle 10, 0 \rangle$ solution $\langle 2, 1, 1, 3 \rangle$
- ▶ $\langle 001, 0 \rangle, \langle 01, 011 \rangle, \langle 01, 101 \rangle, \langle 10, 001 \rangle$ solution length 66

PCP as decision problem

- ▶ $\text{PCP} = \{ \langle \langle w_1, v_1 \rangle, \dots, \langle w_k, v_k \rangle \rangle \mid k \geq 1, w_i, v_i \text{ bin. words} \}$
- ▶ $Y = \{ i \in \text{PCP} \mid i \text{ has a solution} \}$

PCP is Undecidable

Theorem

Post's Correspondence Problem is **undecidable**.

Idea of the Proof.

The termination problem can be **reduced to** PCP.

More precisely, there is a computable function r that maps instances of the termination problem to instances of PCP:

$$r: \langle P, w \rangle \longmapsto I_{\langle P, w \rangle}$$

such that it holds:

$$P \text{ terminates on input } w \iff I_{\langle P, w \rangle} \text{ has a solution}$$

Then if we had a **PCP-solver** (decides solvability of PCP-inst.) we would obtain a **solver for the termination problem**. \times \square

Meta-Theorems of Predicate Logic (continued)

Validity is Undecidable

Theorem

The **validity problem** in predicate logic is **undecidable**.

There cannot be a program that, given any formula ϕ , decides whether or not $\models \phi$ holds.

Proof structure.

PCP can be **encoded into (reduced to)** the validity problem.

We will describe a computable function r that maps instances of PCP to instances of the validity problem:

$$r: I \mapsto \phi_I$$

such that it holds:

$$I \text{ has a solution} \iff \models \phi_I \text{ (i.e. } \phi_I \text{ is valid)}$$

Then if we had a program **deciding validity** for predicate logic, we would obtain a **PCP-solver**. \times

Encoding of Binary Words as Terms

The formula ϕ_I will be defined over functions and predicate in:

$$\mathcal{F} = \{ e/0, f_0/1, f_1/1 \} \quad \mathcal{P} = \{ P/2 \}$$

We consider the encoding of binary strings into \mathcal{F} -terms:

binary word	term encoding
ϵ (empty word)	e
0	$f_0(e)$
1	$f_1(e)$
01	$f_0(f_1(e))$
10	$f_1(f_0(e))$
\vdots	\vdots
$b_1 b_2 \dots b_{l-1} b_l$	$f_{b_1}(f_{b_2}(\dots f_{b_{l-1}}(f_{b_l}(e)) \dots))$

For binary word $w = b_1 b_2 \dots b_l$ and \mathcal{F} -terms t we abbreviate:

$$f_w(t) \stackrel{\text{def}}{=} f_{b_1}(f_{b_2}(\dots f_{b_l}(t) \dots))$$

Encoding a PCP-Instance into a Formula

Now given a PCP instance

$$I = \langle \langle w_1, v_1 \rangle, \langle w_2, v_2 \rangle, \dots, \langle w_k, v_k \rangle \rangle$$

we encode I into a formula ϕ_I of predicate logic as follows:

$$\phi_I \stackrel{\text{def}}{=} \phi_1 \wedge \phi_2 \rightarrow \phi_3$$

$$\phi_1 \stackrel{\text{def}}{=} \bigwedge_{i=1, \dots, k} P(f_{w_i}(e), f_{v_i}(e))$$

$$\phi_2 \stackrel{\text{def}}{=} \forall x \forall y (P(x, y) \rightarrow \bigwedge_{i=1, \dots, k} P(f_{w_i}(x), f_{v_i}(y)))$$

$$\phi_3 \stackrel{\text{def}}{=} \exists z P(z, z)$$

The idea behind P is:

$$P(x, y) \iff \langle x, y \rangle \text{ can be constructed using dominos in } I$$

Encoding a PCP-Instance into a Formula

To understand the formulas we consider the model \mathcal{M} :

- ▶ domain $B =$ set of binary words
- ▶ $e^{\mathcal{M}} = \epsilon$ (empty word)
- ▶ $f_0^{\mathcal{M}}(w) = 0w$
- ▶ $f_1^{\mathcal{M}}(w) = 1w$
- ▶ $P^{\mathcal{M}} = \{ \langle x, y \rangle \mid \text{there are indices } \langle i_1, i_2, \dots, i_n \rangle \text{ such that} \\ x = w_{i_1} w_{i_2} \cdots w_{i_n} \text{ and } y = v_{i_1} v_{i_2} \cdots v_{i_n} \}$

For all $w \in B$ it holds: $(f_w(e))^{\mathcal{M}} = w$.

Encoding a PCP-Instance into a Formula

To understand the formulas we consider the model \mathcal{M} :

- ▶ domain B = set of binary words
- ▶ $e^{\mathcal{M}} = \epsilon$ (empty word)
- ▶ $f_0^{\mathcal{M}}(w) = 0w$
- ▶ $f_1^{\mathcal{M}}(w) = 1w$
- ▶ $P^{\mathcal{M}} = \{ \langle x, y \rangle \mid \text{there are indices } \langle i_1, i_2, \dots, i_n \rangle \text{ such that } x = w_{i_1} w_{i_2} \dots w_{i_n} \text{ and } y = v_{i_1} v_{i_2} \dots v_{i_n} \}$

For all $w \in B$ it holds: $(f_w(e))^{\mathcal{M}} = w$.

Interpreting $\phi_1 \stackrel{\text{def}}{=} \bigwedge_{i=1, \dots, k} P(f_{w_i}(e), f_{v_i}(e))$ in \mathcal{M} we find:

$$\begin{aligned} \mathcal{M} \models_{\ell} \phi_1 &\iff \forall i \text{ with } 1 \leq i \leq k: \mathcal{M} \models_{\ell} P(f_{w_i}(e), f_{v_i}(e)) \\ &\iff \forall i \text{ with } 1 \leq i \leq k: \langle (f_{w_i}(e))^{\mathcal{M}}, (f_{v_i}(e))^{\mathcal{M}} \rangle \in P^{\mathcal{M}} \\ &\iff \forall i \text{ with } 1 \leq i \leq k: \langle w_i, v_i \rangle \in P^{\mathcal{M}} \quad \checkmark \end{aligned}$$

Encoding a PCP-Instance into a Formula

To understand the formulas we consider the model \mathcal{M} :

- ▶ domain B = set of binary words
- ▶ $e^{\mathcal{M}} = \epsilon$ (empty word)
- ▶ $f_0^{\mathcal{M}}(w) = 0w$
- ▶ $f_1^{\mathcal{M}}(w) = 1w$
- ▶ $P^{\mathcal{M}} = \{ \langle x, y \rangle \mid \text{there are indices } \langle i_1, i_2, \dots, i_n \rangle \text{ such that } x = w_{i_1} w_{i_2} \dots w_{i_n} \text{ and } y = v_{i_1} v_{i_2} \dots v_{i_n} \}$

For all $w \in B$ it holds: $(f_w(e))^{\mathcal{M}} = w$.

Interpreting $\phi_2 \stackrel{\text{def}}{=} \forall x \forall y (P(x, y) \rightarrow \bigwedge_{i=1, \dots, k} P(f_{w_i}(x), f_{v_i}(y)))$:

$\mathcal{M} \models \phi_2 \iff \dots$

$\iff (\forall w, v \in B : \langle w, v \rangle \in P^{\mathcal{M}} \implies \bigwedge_{i=1, \dots, k} \langle w_i w, v_i v \rangle \in P^{\mathcal{M}})$

$\iff \left(\langle w_{i_1} \dots w_{i_n}, v_{i_1} \dots v_{i_n} \rangle \in P^{\mathcal{M}} \implies \langle w_i w_{i_1} \dots w_{i_n}, v_i v_{i_1} \dots v_{i_n} \rangle \in P^{\mathcal{M}_i} \right) \checkmark$

Encoding a PCP-Instance into a Formula

To understand the formulas we consider the model \mathcal{M} :

- ▶ domain B = set of binary words
- ▶ $e^{\mathcal{M}} = \epsilon$ (empty word)
- ▶ $f_0^{\mathcal{M}}(w) = 0w$
- ▶ $f_1^{\mathcal{M}}(w) = 1w$
- ▶ $P^{\mathcal{M}} = \{ \langle x, y \rangle \mid \text{there are indices } \langle i_1, i_2, \dots, i_n \rangle \text{ such that } x = w_{i_1} w_{i_2} \cdots w_{i_n} \text{ and } y = v_{i_1} v_{i_2} \cdots v_{i_n} \}$

For all $w \in B$ it holds: $(f_w(e))^{\mathcal{M}} = w$.

Interpreting $\phi_3 \stackrel{\text{def}}{=} \exists z P(z, z)$ in \mathcal{M} we find:

- $\mathcal{M} \models_{\ell} \phi_3 \iff$ there is a $w \in B$ such that $\langle w, w \rangle \in P^{\mathcal{M}}$
- \iff there is a sequence $\langle i_1, i_2, \dots, i_n \rangle$ of indices such that $w_{i_1} w_{i_2} \cdots w_{i_n} = v_{i_1} v_{i_2} \cdots v_{i_n}$
- \iff PCP instance I is solvable

Encoding a PCP-Instance into a Formula

So we have shown:

$$\mathcal{M} \models \phi_1$$

$$\mathcal{M} \models \phi_2$$

$$\mathcal{M} \models \phi_3 \iff I \text{ is solvable}$$

Hence we conclude:

$$\mathcal{M} \models \phi_I \iff \mathcal{M} \models \phi_1 \wedge \phi_2 \rightarrow \phi_3$$

$$\iff I \text{ is solvable}$$

and furthermore:

$$\phi_I \text{ is valid} \implies \mathcal{M} \models \phi_I$$

$$\implies I \text{ is solvable}$$

Validity is Undecidable

Theorem

The **validity problem** in predicate logic is **undecidable**.

Proof structure.

There is a computable function r (see previous slides) that maps instances of PCP to instances of the validity problem:

$$r: I \longmapsto \phi_I$$

such that it holds:

$$I \text{ has a solution} \iff \models \phi_I \quad (\text{i.e. } \phi_I \text{ is valid}) \quad (*)$$

Then if we had a program **deciding validity** for predicate logic, we would obtain a PCP-solver. \times

' \iff ' in (*): just shown \checkmark

' \implies ' in (*): see Huth & Ryan p. 134, 135

Alan Turing



Alan Mathison Turing (1912–1954)

Undecidability of Validity and Provability

Theorem (Church, Turing, 1936/37)

The **validity problem** in predicate logic is **undecidable**.

There cannot be a program that, given any formula ϕ , decides whether or not $\models \phi$ holds.

Using the soundness and completeness theorem we obtain:

Corollary (Undecidability of Provability)

The **provability problem** in predicate logic is **undecidable**.

There cannot be a program that, given any formula ϕ , decides whether or not $\vdash \phi$ holds.

- ▶ limits the power of theorem provers
- ▶ building better theorem provers is an open-ended endeavour (creativity will always be needed)

Also Satisfiability is Undecidable

Proposition

For sentences ϕ it holds:

ϕ is unsatisfiable $\iff \neg\phi$ is valid

ϕ is satisfiable $\iff \neg\phi$ is not valid

Since this defines an easy reduction of the validity problem to the satisfiability problem. It follows immediately:

Theorem

The **satisfiability problem** in predicate logic is **undecidable**.

Undecidability of \models and \vdash

Deduction Theorem

Easy connection between \models and validity:

$$\begin{aligned}\phi_1, \phi_2, \dots, \phi_n \models \psi &\iff \models \phi_1 \wedge \phi_2 \wedge \dots \wedge \phi_n \rightarrow \psi \\ &\iff \phi_1 \wedge \phi_2 \wedge \dots \wedge \phi_n \rightarrow \psi \text{ is valid}\end{aligned}$$

and between \vdash and provability:

$$\begin{aligned}\phi_1, \phi_2, \dots, \phi_n \vdash \psi &\iff \vdash \phi_1 \wedge \phi_2 \wedge \dots \wedge \phi_n \rightarrow \psi \\ &\iff \phi_1 \wedge \phi_2 \wedge \dots \wedge \phi_n \rightarrow \psi \text{ is provable}\end{aligned}$$

Corollary (Undecidability of entailment relations \models and \vdash)

The relations \models and \vdash in predicate logic are **undecidable**.

There cannot be a program that, given formulas $\phi_1, \dots, \phi_n, \psi$ decides whether or not $\phi_1, \dots, \phi_n \models \psi$ (or $\phi_1, \dots, \phi_n \vdash \psi$).

Kurt Gödel



Kurt Gödel with Albert Einstein (Princeton, around 1950)

Incompleteness Theorem

We consider the sets of function and predicate symbols:

$$\mathcal{F} = \{ 0, \mathbf{S}, +, \cdot \} \quad \mathcal{P} = \{ < \}$$

with as intended model **number theory** \mathcal{N} :

- ▶ domain of \mathcal{N} is \mathbb{N} , the natural numbers (with 0)
- ▶ $0^{\mathcal{N}} = 0$
- ▶ $\mathbf{S}^{\mathcal{N}}(n) = n + 1$
- ▶ $+^{\mathcal{N}}(n, m) = n + m$
- ▶ $\cdot^{\mathcal{N}}(n, m) = n \cdot m$
- ▶ $<^{\mathcal{N}} = \{ \langle n, m \rangle \mid n, m \in \mathbb{N} \text{ such that } n < m \}$

Example formula: $\forall n. \forall m. \exists o. (n < m + o)$

One would like to have a

complete theory (deduction system) \vdash for \mathcal{N}
that allows to derive all formulas that are true in \mathcal{N} .

First Incompleteness Theorem

Desirable: an **axiomatizable first-order theory** \vdash for \mathcal{N} :

- ▶ an effective list \mathcal{A} of axioms (pred. logic $\langle \mathcal{F}, \mathcal{P} \rangle$ -formulas)
- ▶ finitely many rules \mathcal{R}
- ▶ $\vdash \phi$ means: there is a derivation of ϕ from axioms \mathcal{A} that only used rules from \mathcal{R}

that is sound and complete for \mathcal{N} :

$$\mathcal{N} \models \phi \iff \vdash \phi \quad (\text{for all } \langle \mathcal{F}, \mathcal{P} \rangle\text{-formulas } \phi)$$

Yet it turned out that this is impossible.

First incompleteness theorem (Gödel, 1931)

Every axiomatizable and sound theory \vdash of first-order logic for number theory with language $\langle \mathcal{F}, \mathcal{P} \rangle$ is **incomplete**. That is, it contains sentences ϕ that are **true** in \mathcal{N} , **but unprovable** in \vdash :

$$\mathcal{N} \models \phi, \text{ yet } \not\vdash \phi$$

Second Incompleteness Theorem

Theorem (Gödel, von Neumann, 1930-31)

For every axiomatizable theory \vdash of first-order logic for number theory with language $\langle \mathcal{F}, \mathcal{P} \rangle$

- ▶ that is rich enough to express its own consistency by a sentence ϕ_{\vdash}

it holds that either:

- ▶ $\vdash \perp$ (\vdash is inconsistent) , or
- ▶ $\not\vdash \phi_{\vdash}$ (hence \vdash is incomplete)

\implies First-order theories (based on predicate logic) of number theory are not able to prove their own consistency.

Timeline: From Logic to Computability

- 1900** Hilbert's 23 Problems in mathematics
- 1921** Schönfinkel: Combinatory logic
- 1928** Hilbert/Ackermann: formulate completeness/decision problems for the predicate calculus (the latter called 'Entscheidungsproblem')
- 1929** Presburger: completeness/decidability of theory of addition on \mathbb{Z}
- 1930** Gödel: completeness theorem of predicate calculus
- 1931** Gödel: incompleteness theorems for first-order arithmetic
- 1932** Church: λ -calculus
- 1933/34** Herbrand/Gödel: general recursive functions
- 1936** Church/Kleene: λ -definable \sim general recursive
Church Thesis: 'effectively calculable' be defined as either
Church shows: the 'Entscheidungsproblem' is unsolvable
- 1937** Post: machine model; Church's thesis as 'working hypothesis'
Turing: convincing analysis of a 'human computer'
leading to the 'Turing machine'