Undecidability in Logic -Part II

K. Subramani[1]

[1]Lane Department of Computer Science and Electrical Engineering
West Virginia University

Incompleteness of Number Theory

Outline

Outline

Outline

Number-theoretic encoding of computation

Turing Machine encodings

(i) Every Turing Machine $M = (K, \Sigma, \delta, s\rangle)$ can be represented as a number in b-ary notation, where $b = |K| + |\Sigma|$.

(ii) Therefore, configurations can be encoded as *sequences* of integers in b-ary representation.

(iii) The "yields in one step" function over configurations of a Turing Machine, defines a relation $Y_M \subseteq \mathcal{N}^2$

Goal

To formulate a first-order expression $yields_M(x, y)$ in number theory, over the free variables x and y, such that

$\mathbf{N}_{x=m, y=n} \models yields_M(x, y)$ *iff* $Y_M(m, n)$.

Number-theoretic encoding of computation

Turing Machine encodings

(i) Every Turing Machine $M = (K, \Sigma, \delta, s\rangle)$ can be represented as a number in b-ary notation, where $b = |K| + |\Sigma|$.

(ii) Therefore, configurations can be encoded as *sequences* of integers in b-ary representation.

(iii) The "yields in one step" function over configurations of a Turing Machine, defines a relation $Y_M \subseteq \mathcal{N}^2$

Goal

To formulate a first-order expression $yields_M(x, y)$ in number theory, over the free variables x and y, such that

$\mathbf{N}_{x=m, y=n} \models yields_M(x, y)$ *iff* $Y_M(m, n)$.

Number-theoretic encoding of computation

Turing Machine encodings

(i) Every Turing Machine $M = (K, \Sigma, \delta, s\rangle)$ can be represented as a number in *b*-ary notation, where $b = |K| + |\Sigma|$.

(ii) Therefore, configurations can be encoded as *sequences* of integers in *b*-ary representation.

(iii) The "yields in one step" function over configurations of a Turing Machine, defines a relation $Y_M \subseteq \mathcal{N}^2$

Goal

To formulate a first-order expression $yields_M(x, y)$ in number theory, over the free variables x and y, such that

$\mathbf{N}_{x=m, y=n} \models yields_M(x, y)$ *iff* $Y_M(m, n)$.

Number-theoretic encoding of computation

Turing Machine encodings

(i) Every Turing Machine $M = (K, \Sigma, \delta, s\rangle)$ can be represented as a number in b-ary notation, where $b = |K| + |\Sigma|$.

(ii) Therefore, configurations can be encoded as *sequences* of integers in b-ary representation.

(iii) The "yields in one step" function over configurations of a Turing Machine, defines a relation $Y_M \subseteq \mathcal{N}^2$

Goal

To formulate a first-order expression $yields_M(x, y)$ in number theory, over the free variables x and y, such that

$\mathbf{N}_{x=m,y=n} \models yields_M(x, y)$ *iff* $Y_M(m, n)$.

Number-theoretic encoding of computation

Turing Machine encodings

(i) Every Turing Machine $M = (K, \Sigma, \delta, s\rangle)$ can be represented as a number in b-ary notation, where $b = |K| + |\Sigma|$.

(ii) Therefore, configurations can be encoded as *sequences* of integers in b-ary representation.

(iii) The "yields in one step" function over configurations of a Turing Machine, defines a relation $Y_M \subseteq \mathcal{N}^2$

Goal

To formulate a first-order expression $yields_M(x, y)$ in number theory, over the free variables x and y, such that

$\mathbf{N}_{x=m,y=n} \models yields_M(x, y)$ *iff* $Y_M(m, n)$.

Number-theoretic encoding of computation

Turing Machine encodings

(i) Every Turing Machine $M = (K, \Sigma, \delta, s\rangle)$ can be represented as a number in b-ary notation, where $b = |K| + |\Sigma|$.

(ii) Therefore, configurations can be encoded as *sequences* of integers in b-ary representation.

(iii) The "yields in one step" function over configurations of a Turing Machine, defines a relation $Y_M \subseteq \mathcal{N}^2$

Goal

To formulate a first-order expression $yields_M(x, y)$ in number theory, over the free variables x and y, such that

$\mathbf{N}_{x=m, y=n} \models yields_M(x, y)$ *iff* $Y_M(m, n)$.

Formulating the encoding technique

$p \in K$,	$\sigma \in \Sigma$	$\delta(p, \sigma)$
s	a	$(s, a, \rightarrow)$
s	b	$(s, b, \rightarrow)$
s	$\sqcup$	$(q, \sqcup, \leftarrow)$
s	$\triangleright$	$(q, \triangleright, \rightarrow)$
q	a	$(q, \sqcup, \leftarrow)$
q	b	$(\text{"}no\text{"}, b, -)$
q	$\triangleright$	$(\text{"}yes\text{"}, \triangleright, \rightarrow)$

Table: A Turing Machine that accepts a^*

Example

Consider the $C_1 = (q, \triangleright aa, \sqcup\sqcup)$ and the configuration that follows $C_2 = (q, \triangleright a, \sqcup \sqcup \sqcup)$. The corresponding encodings $m = 022711_8$ and 027111_8 are related under Y_M.

Observation

(i) *m and n are identical, except for the replacement of* 271_8 *in m, by* 711_8 *in n.*

(ii) *But this corresponds to Rule 5 in the table! Thus, every move is a local replacement of digits.*

Formulating the encoding technique

$p \in K$,	$\sigma \in \Sigma$	$\delta(p, \sigma)$
s	a	$(s, a, \rightarrow)$
s	b	$(s, b, \rightarrow)$
s	$\sqcup$	$(q, \sqcup, \leftarrow)$
s	$\triangleright$	$(q, \triangleright, \rightarrow)$
q	a	$(q, \sqcup, \leftarrow)$
q	b	$(\text{"}no\text{"}, b, -)$
q	$\triangleright$	$(\text{"}yes\text{"}, \triangleright, \rightarrow)$

Table: A Turing Machine that accepts a^*

Example

Consider the $C_1 = (q, \triangleright aa, \sqcup\sqcup)$ and the configuration that follows $C_2 = (q, \triangleright a, \sqcup\sqcup\sqcup)$. The corresponding encodings $m = 022711_8$ and 027111_8 are related under Y_M.

Observation

(i) *m and n are identical, except for the replacement of 271_8 in m, by 711_8 in n.*

(ii) *But this corresponds to Rule 5 in the table! Thus, every move is a local replacement of digits.*

Formulating the encoding technique

$p \in K$,	$\sigma \in \Sigma$	$\delta(p, \sigma)$
s	a	$(s, a, \rightarrow)$
s	b	$(s, b, \rightarrow)$
s	$\sqcup$	$(q, \sqcup, \leftarrow)$
s	$\triangleright$	$(q, \triangleright, \rightarrow)$
q	a	$(q, \sqcup, \leftarrow)$
q	b	$(\text{"}no\text{"}, b, -)$
q	$\triangleright$	$(\text{"}yes\text{"}, \triangleright, \rightarrow)$

Table: A Turing Machine that accepts a^*

Example

Consider the $C_1 = (q, \triangleright aa, \sqcup\sqcup)$ and the configuration that follows $C_2 = (q, \triangleright a, \sqcup \sqcup \sqcup)$. The corresponding encodings $m = 022711_8$ and 027111_8 are related under Y_M.

Observation

(i) *m and n are identical, except for the replacement of* 271_8 *in m, by* 711_8 *in n.*

(ii) *But this corresponds to Rule 5 in the table! Thus, every move is a local replacement of digits.*

Formulating the encoding technique

$p \in K,$	$\sigma \in \Sigma$	$\delta(p, \sigma)$
s	a	$(s, a, \rightarrow)$
s	b	$(s, b, \rightarrow)$
s	$\sqcup$	$(q, \sqcup, \leftarrow)$
s	$\triangleright$	$(q, \triangleright, \rightarrow)$
q	a	$(q, \sqcup, \leftarrow)$
q	b	$(\text{"}no\text{"}, b, -)$
q	$\triangleright$	$(\text{"}yes\text{"}, \triangleright, \rightarrow)$

Table: A Turing Machine that accepts a^*

Example

Consider the $C_1 = (q, \triangleright aa, \sqcup\sqcup)$ and the configuration that follows $C_2 = (q, \triangleright a, \sqcup \sqcup \sqcup)$. The corresponding encodings $m = 022711_8$ and 027111_8 are related under Y_M.

Observation

(i) *m and n are identical, except for the replacement of 271_8 in m, by 711_8 in n.*
(ii) *But this corresponds to Rule 5 in the table! Thus, every move is a local replacement of digits.*

Formulating the encoding technique

$p \in K,$	$\sigma \in \Sigma$	$\delta(p, \sigma)$
s	a	$(s, a, \rightarrow)$
s	b	$(s, b, \rightarrow)$
s	$\sqcup$	$(q, \sqcup, \leftarrow)$
s	$\triangleright$	$(q, \triangleright, \rightarrow)$
q	a	$(q, \sqcup, \leftarrow)$
q	b	$(\text{"}no\text{"}, b, -)$
q	$\triangleright$	$(\text{"}yes\text{"}, \triangleright, \rightarrow)$

Table: A Turing Machine that accepts a^*

Example

Consider the $C_1 = (q, \triangleright aa, \sqcup\sqcup)$ and the configuration that follows $C_2 = (q, \triangleright a, \sqcup \sqcup \sqcup)$. The corresponding encodings $m = 022711_8$ and 027111_8 are related under Y_M.

Observation

(i) *m and n are identical, except for the replacement of 271_8 in m, by 711_8 in n.*

(ii) *But this corresponds to Rule 5 in the table! Thus, every move is a local replacement of digits.*

Formulating the encoding technique

$p \in K,$	$\sigma \in \Sigma$	$\delta(p, \sigma)$
s	a	$(s, a, \rightarrow)$
s	b	$(s, b, \rightarrow)$
s	$\sqcup$	$(q, \sqcup, \leftarrow)$
s	$\triangleright$	$(q, \triangleright, \rightarrow)$
q	a	$(q, \sqcup, \leftarrow)$
q	b	$(\text{"}no\text{"}, b, -)$
q	$\triangleright$	$(\text{"}yes\text{"}, \triangleright, \rightarrow)$

Table: A Turing Machine that accepts a^*

Example

Consider the $C_1 = (q, \triangleright aa, \sqcup\sqcup)$ and the configuration that follows $C_2 = (q, \triangleright a, \sqcup \sqcup \sqcup)$. The corresponding encodings $m = 022711_8$ and 027111_8 are related under Y_M.

Observation

(i) *m and n are identical, except for the replacement of 271_8 in m, by 711_8 in n.*
(ii) *But this corresponds to Rule 5 in the table!* *Thus, every move is a local replacement of digits.*

Formulating the encoding technique

$p \in K,$	$\sigma \in \Sigma$	$\delta(p, \sigma)$
s	a	$(s, a, \rightarrow)$
s	b	$(s, b, \rightarrow)$
s	$\sqcup$	$(q, \sqcup, \leftarrow)$
s	$\triangleright$	$(q, \triangleright, \rightarrow)$
q	a	$(q, \sqcup, \leftarrow)$
q	b	$("no", b, -)$
q	$\triangleright$	$("yes", \triangleright, \rightarrow)$

Table: A Turing Machine that accepts a^*

Example

Consider the $C_1 = (q, \triangleright aa, \sqcup\sqcup)$ and the configuration that follows $C_2 = (q, \triangleright a, \sqcup \sqcup \sqcup)$. The corresponding encodings $m = 022711_8$ and 027111_8 are related under Y_M.

Observation

(i) *m and n are identical, except for the replacement of 271_8 in m, by 711_8 in n.*

(ii) *But this corresponds to Rule 5 in the table! Thus, every move is a local replacement of digits.*

Formulating the encoding technique (contd.)

Triplet changes

Capture each rule change as a triplet transformation!

042_8	$\longrightarrow$	024_8
043_8	$\longrightarrow$	034_8
041_8	$\longrightarrow$	014_8
242_8	$\longrightarrow$	224_8
243_8	$\longrightarrow$	234_8
241_8	$\longrightarrow$	214_8
242_8	$\longrightarrow$	224_8
242_8	$\longrightarrow$	224_8
$\vdots$		$\vdots$
371_8	$\longrightarrow$	361_8

Table: Encoding the triplet changes

$$table_M(x, y) = ((x = 042_8 \wedge y = 024_8) \vee \ldots (x = 371_8 \wedge y = 361_8))$$

Formulating the encoding technique (contd.)

Triplet changes

Capture each rule change as a triplet transformation!

042_8	$\longrightarrow$	024_8
043_8	$\longrightarrow$	034_8
041_8	$\longrightarrow$	014_8
242_8	$\longrightarrow$	224_8
243_8	$\longrightarrow$	234_8
241_8	$\longrightarrow$	214_8
242_8	$\longrightarrow$	224_8
242_8	$\longrightarrow$	224_8
$\vdots$		$\vdots$
371_8	$\longrightarrow$	361_8

Table: Encoding the triplet changes

$$table_M(x, y) = ((x = 042_8 \wedge y = 024_8) \vee \ldots (x = 371_8 \wedge y = 361_8))$$

Formulating the encoding technique (contd.)

Triplet changes

Capture each rule change as a triplet transformation!

042_8	$\longrightarrow$	024_8
043_8	$\longrightarrow$	034_8
041_8	$\longrightarrow$	014_8
242_8	$\longrightarrow$	224_8
243_8	$\longrightarrow$	234_8
241_8	$\longrightarrow$	214_8
242_8	$\longrightarrow$	224_8
242_8	$\longrightarrow$	224_8
$\vdots$		$\vdots$
371_8	$\longrightarrow$	361_8

Table: Encoding the triplet changes

$table_M(x, y) = ((x = 042_8 \wedge y = 024_8) \vee \ldots (x = 371_8 \wedge y = 361_8))$

Formulating the encoding technique (contd.)

Triplet changes

Capture each rule change as a triplet transformation!

042_8	$\longrightarrow$	024_8
043_8	$\longrightarrow$	034_8
041_8	$\longrightarrow$	014_8
242_8	$\longrightarrow$	224_8
243_8	$\longrightarrow$	234_8
241_8	$\longrightarrow$	214_8
242_8	$\longrightarrow$	224_8
242_8	$\longrightarrow$	224_8
$\vdots$		$\vdots$
371_8	$\longrightarrow$	361_8

Table: Encoding the triplet changes

$$table_M(x, y) = ((x = 042_8 \wedge y = 024_8) \vee \ldots (x = 371_8 \wedge y = 361_8))$$

Formulating the encoding technique (contd.)

Padding

The computation of M on input aa:

0422_8, 0242_8, 0224_8, 02241_8, ...

Formulating the encoding technique (contd.)

Padding

The computation of *M* on input *aa*:

0422_8, 0242_8, 0224_8, 02241_8, ...

Formulating the encoding technique (contd.)

Padding

The computation of *M* on input *aa*:

0422_8, 0242_8, 0224_8, 02241_8, ...

Formulating the encoding technique (contd.)

Padding

The computation of *M* on input *aa*:

0422_8, 0242_8, 0224_8, 02241_8, ...

Formulating the encoding technique (contd.)

Padding

The computation of M on input aa:

$0422_8, 0242_8, 0224_8, 02241_8, \ldots.$

A first-order Number-Theoretic expression for Computation

The Actual Formula

$$yields_M(x, x') = pads_M(x, x') \vee$$
$$(\exists y < x)(\exists z_1 < x)(\exists z_2 < x)(\exists z_2' < x)(\exists z_3 < x)(\exists z_3' < x)(\exists z_4 < x)$$
$$(conf_M(x) \wedge conf_M(x') \wedge$$
$$\bmod(x, b \uparrow y, z_1) \wedge \mathrm{div}(x, b \uparrow y, z_2) \wedge \quad \bmod(x', b \uparrow y, z_1) \wedge \mathrm{div}(x', b \uparrow y, z_2') \wedge$$
$$\bmod(z_2, b \uparrow 3, z_3) \wedge \mathrm{div}(z_2, b \uparrow 3, z_4) \wedge \quad \bmod(z_2', b \uparrow 3, z_3') \wedge \mathrm{div}(z_2', b \uparrow 3, z_4) \wedge$$
$$table_M(z_3, z_3'))$$

Auxiliary expressions

Similar expressions can be written for $pads_M(x, x')$ (x' is obtained from x by adding a ⊔) and $conf_M(x)$ (the b-ary representation of x correctly encodes a configuration of M).

A first-order Number-Theoretic expression for Computation

The Actual Formula

$$yields_M(x, x') = pads_M(x, x') \vee$$
$$(\exists y < x)(\exists z_1 < x)(\exists z_2 < x)(\exists z_2' < x)(\exists z_3 < x)(\exists z_3' < x)(\exists z_4 < x)$$
$$(conf_M(x) \wedge conf_M(x') \wedge$$
$$\text{mod}\ (x, b \uparrow y, z_1) \wedge \text{div}(x, b \uparrow y, z_2) \wedge \quad \text{mod}\ (x', b \uparrow y, z_1) \wedge \text{div}(x', b \uparrow y, z_2') \wedge$$
$$\text{mod}\ (z_2, b \uparrow 3, z_3) \wedge \text{div}(z_2, b \uparrow 3, z_4) \wedge \quad \text{mod}\ (z_2', b \uparrow 3, z_3') \wedge \text{div}(z_2', b \uparrow 3, z_4) \wedge$$
$$table_M(z_3, z_3')$$

Auxiliary expressions

Similar expressions can be written for $pads_M(x, x')$ (x' is obtained from x by adding a $\sqcup$) and $conf_M(x)$ (the b-ary representation of x correctly encodes a configuration of M).

A first-order Number-Theoretic expression for Computation

The Actual Formula

$$
\begin{aligned}
yields_M(x, x') = pads_M(x, x') \vee \\
(\exists y < x)(\exists z_1 < x)(\exists z_2 < x)(\exists z_2' < x)(\exists z_3 < x)(\exists z_3' < x)(\exists z_4 < x) \\
(conf_M(x) \wedge conf_M(x') \wedge \\
\text{mod}\ (x, b \uparrow y, z_1) \wedge \text{div}(x, b \uparrow y, z_2) \wedge \quad \text{mod}\ (x', b \uparrow y, z_1) \wedge \text{div}(x', b \uparrow y, z_2') \wedge \\
\text{mod}\ (z_2, b \uparrow 3, z_3) \wedge \text{div}(z_2, b \uparrow 3, z_4) \wedge \quad \text{mod}\ (z_2', b \uparrow 3, z_3') \wedge \text{div}(z_2', b \uparrow 3, z_4) \wedge \\
table_M(z_3, z_3')
\end{aligned}
$$

Auxiliary expressions

Similar expressions can be written for $pads_M(x, x')$ (x' is obtained from x by adding a ⊔) and $conf_M(x)$ (the b-ary representation of x correctly encodes a configuration of M).

A first-order Number-Theoretic expression for Computation

The Actual Formula

$$\begin{aligned} yields_M(x, x') = pads_M(x, x') \vee \\ (\exists y < x)(\exists z_1 < x)(\exists z_2 < x)(\exists z_2' < x)(\exists z_3 < x)(\exists z_3' < x)(\exists z_4 < x) \\ (conf_M(x) \wedge conf_M(x') \wedge \\ \bmod(x, b \uparrow y, z_1) \wedge \mathrm{div}(x, b \uparrow y, z_2) \wedge \quad \bmod(x', b \uparrow y, z_1) \wedge \mathrm{div}(x', b \uparrow y, z_2') \wedge \\ \bmod(z_2, b \uparrow 3, z_3) \wedge \mathrm{div}(z_2, b \uparrow 3, z_4) \wedge \quad \bmod(z_2', b \uparrow 3, z_3') \wedge \mathrm{div}(z_2', b \uparrow 3, z_4) \wedge \\ table_M(z_3, z_3')) \end{aligned}$$

Auxiliary expressions

Similar expressions can be written for $pads_M(x, x')$ (x' is obtained from x by adding a ⊔) and $conf_M(x)$ (the b-ary representation of x correctly encodes a configuration of M).

A first-order Number-Theoretic expression for Computation

The Actual Formula

$$
\begin{aligned}
yields_M(x,x') = pads_M(x,x') \vee \\
(\exists y < x)(\exists z_1 < x)(\exists z_2 < x)(\exists z_2' < x)(\exists z_3 < x)(\exists z_3' < x)(\exists z_4 < x) \\
(conf_M(x) \wedge conf_M(x') \wedge \\
\bmod (x, b \uparrow y, z_1) \wedge \mathrm{div}(x, b \uparrow y, z_2) \wedge \quad \bmod (x', b \uparrow y, z_1) \wedge \mathrm{div}(x', b \uparrow y, z_2') \wedge \\
\bmod (z_2, b \uparrow 3, z_3) \wedge \mathrm{div}(z_2, b \uparrow 3, z_4) \wedge \quad \bmod (z_2', b \uparrow 3, z_3') \wedge \mathrm{div}(z_2', b \uparrow 3, z_4) \wedge \\
table_M(z_3, z_3')
\end{aligned}
$$

Auxiliary expressions

Similar expressions can be written for $pads_M(x,x')$ (x' is obtained from x by adding a ⊔) and $conf_M(x)$ (the b-ary representation of x correctly encodes a configuration of M).

A first-order Number-Theoretic expression for Computation

The Actual Formula

$$yields_M(x, x') = pads_M(x, x') \vee$$
$$(\exists y < x)(\exists z_1 < x)(\exists z_2 < x)(\exists z_2' < x)(\exists z_3 < x)(\exists z_3' < x)(\exists z_4 < x)$$
$$(conf_M(x) \wedge conf_M(x') \wedge$$
$$\text{mod } (x, b \uparrow y, z_1) \wedge \text{div}(x, b \uparrow y, z_2) \wedge \quad \text{mod } (x', b \uparrow y, z_1) \wedge \text{div}(x', b \uparrow y, z_2') \wedge$$
$$\text{mod } (z_2, b \uparrow 3, z_3) \wedge \text{div}(z_2, b \uparrow 3, z_4) \wedge \quad \text{mod } (z_2', b \uparrow 3, z_3') \wedge \text{div}(z_2', b \uparrow 3, z_4) \wedge$$
$$table_M(z_3, z_3')$$

Auxiliary expressions

Similar expressions can be written for $pads_M(x, x')$ (x' is obtained from x by adding a ⊔) and $conf_M(x)$ (the b-ary representation of x correctly encodes a configuration of M).

A first-order Number-Theoretic expression for Computation

The Actual Formula

$$
\begin{aligned}
&yields_M(x, x') = pads_M(x, x') \vee \\
&(\exists y < x)(\exists z_1 < x)(\exists z_2 < x)(\exists z_2' < x)(\exists z_3 < x)(\exists z_3' < x)(\exists z_4 < x) \\
&(conf_M(x) \wedge conf_M(x') \wedge \\
&\text{mod } (x, b \uparrow y, z_1) \wedge \text{div}(x, b \uparrow y, z_2) \wedge \quad \text{mod } (x', b \uparrow y, z_1) \wedge \text{div}(x', b \uparrow y, z_2') \wedge \\
&\text{mod } (z_2, b \uparrow 3, z_3) \wedge \text{div}(z_2, b \uparrow 3, z_4) \wedge \quad \text{mod } (z_2', b \uparrow 3, z_3') \wedge \text{div}(z_2', b \uparrow 3, z_4) \wedge \\
&table_M(z_3, z_3')
\end{aligned}
$$

Auxiliary expressions

Similar expressions can be written for $pads_M(x, x')$ (x' is obtained from x by adding a $\sqcup$) and $conf_M(x)$ (the b-ary representation of x correctly encodes a configuration of M).

A first-order Number-Theoretic expression for Computation

The Actual Formula

$$\begin{aligned}
&yields_M(x, x') = pads_M(x, x') \lor \\
&(\exists y < x)(\exists z_1 < x)(\exists z_2 < x)(\exists z_2' < x)(\exists z_3 < x)(\exists z_3' < x)(\exists z_4 < x) \\
&(conf_M(x) \land conf_M(x') \land \\
&\bmod(x, b \uparrow y, z_1) \land \mathrm{div}(x, b \uparrow y, z_2) \land \quad \bmod(x', b \uparrow y, z_1) \land \mathrm{div}(x', b \uparrow y, z_2') \land \\
&\bmod(z_2, b \uparrow 3, z_3) \land \mathrm{div}(z_2, b \uparrow 3, z_4) \land \quad \bmod(z_2', b \uparrow 3, z_3') \land \mathrm{div}(z_2', b \uparrow 3, z_4) \land \\
&table_M(z_3, z_3')
\end{aligned}$$

Auxiliary expressions

Similar expressions can be written for $pads_M(x, x')$ (x' is obtained from x by adding a ⊔) and $conf_M(x)$ (the b-ary representation of x correctly encodes a configuration of M).

A first-order Number-Theoretic expression for Computation

The Actual Formula

$$
\begin{aligned}
yields_M(x, x') = pads_M(x, x') \vee \\
(\exists y < x)(\exists z_1 < x)(\exists z_2 < x)(\exists z_2' < x)(\exists z_3 < x)(\exists z_3' < x)(\exists z_4 < x) \\
(conf_M(x) \wedge conf_M(x') \wedge \\
\text{mod } (x, b \uparrow y, z_1) \wedge \text{div}(x, b \uparrow y, z_2) \wedge \quad \text{mod } (x', b \uparrow y, z_1) \wedge \text{div}(x', b \uparrow y, z_2') \wedge \\
\text{mod } (z_2, b \uparrow 3, z_3) \wedge \text{div}(z_2, b \uparrow 3, z_4) \wedge \quad \text{mod } (z_2', b \uparrow 3, z_3') \wedge \text{div}(z_2', b \uparrow 3, z_4) \wedge \\
table_M(z_3, z_3')
\end{aligned}
$$

Auxiliary expressions

Similar expressions can be written for $pads_M(x, x')$ (x' is obtained from x by adding a $\sqcup$) and $conf_M(x)$ (the b-ary representation of x correctly encodes a configuration of M).

A first-order Number-Theoretic expression for Computation (contd.)

Observation

Whole computations of M can be encoded!

Lemma

For each Turing machine M, we can construct a bounded expression $comp_M(x)$ in number theory such that: $\forall n \in \mathcal{N},\ \mathbf{N}_{x=n} \models comp_M(x) \leftrightarrow$ the b-ary representation of n is the juxtaposition of consecutive configurations of a halting computation of M, starting from the empty string.

A first-order Number-Theoretic expression for Computation (contd.)

Observation

Whole computations of M can be encoded!

Lemma

For each Turing machine M, we can construct a bounded expression $comp_M(x)$ in number theory such that: $\forall n \in \mathcal{N}$, $\mathbf{N}_{x=n} \models comp_M(x) \leftrightarrow$ the b-ary representation of n is the juxtaposition of consecutive configurations of a halting computation of M, starting from the empty string.

A first-order Number-Theoretic expression for Computation (contd.)

Observation

Whole computations of M can be encoded!

Lemma

For each Turing machine M, we can construct a bounded expression $comp_M(x)$ in number theory such that: $\forall n \in \mathcal{N}, \ \mathbf{N}_{x=n} \models comp_M(x) \leftrightarrow$ the b-ary representation of n is the juxtaposition of consecutive configurations of a halting computation of M, starting from the empty string.

A first-order Number-Theoretic expression for Computation (contd.)

Observation

Whole computations of M can be encoded!

Lemma

For each Turing machine M, we can construct a bounded expression $comp_M(x)$ in number theory such that: $\forall n \in \mathcal{N},\ \mathbf{N}_{x=n} \models comp_M(x) \leftrightarrow$ the b-ary representation of n is the juxtaposition of consecutive configurations of a halting computation of M, starting from the empty string.

Outline

1. Number-theoretic encoding of computation
2. Undecidability
   - Sentence Classification
   - Recursive Inseparability
3. Incompleteness
   - Gødel's Incompleteness Theorem

Sentence Classification

(i) ϕ is valid, i.e., $\models \phi$ (L_v).

(ii) ϕ is provable from **NT**, i.e., **NT** $\vdash \phi$ (L_p).

(iii) **N** is a model for ϕ, i.e., **N** $\models \phi$ (L_m).

(iv) **N** is a model for $\neg\phi$, i.e., **N** $\models \neg\phi$ (L_{nm}).

(v) $\neg\phi$ is provable from **NT**, i.e., **NT** $\vdash \neg\phi$ (L_{np}).

(vi) $\neg\phi$ is valid, i.e., $\models \neg\phi$ (L_{us}).

Sentence Classification

(i) ϕ is valid, i.e., $\models \phi$ (L_v).
(ii) ϕ is provable from **NT**, i.e., **NT** $\vdash \phi$ (L_p).
(iii) **N** is a model for ϕ, i.e., **N** $\models \phi$ (L_m).
(iv) **N** is a model for $\neg\phi$, i.e., **N** $\models \neg\phi$ (L_{nm}).
(v) $\neg\phi$ is provable from **NT**, i.e., **NT** $\vdash \neg\phi$ (L_{np}).
(vi) $\neg\phi$ is valid, i.e., $\models \neg\phi$ (L_{us}).

Sentence Classification

(i) ϕ is valid, i.e., $\models \phi$ (L_v).

(ii) ϕ is provable from **NT**, i.e., **NT** $\vdash \phi$ (L_p).

(iii) **N** is a model for ϕ, i.e., **N** $\models \phi$ (L_m).

(iv) **N** is a model for $\neg\phi$, i.e., **N** $\models \neg\phi$ (L_{nm}).

(v) $\neg\phi$ is provable from **NT**, i.e., **NT** $\vdash \neg\phi$ (L_{np}).

(vi) $\neg\phi$ is valid, i.e., $\models \neg\phi$ (L_{us}).

Sentence Classification

(i) ϕ is valid, i.e., $\models \phi$ (L_v).

(ii) ϕ is provable from **NT**, i.e., **NT** $\vdash \phi$ (L_p).

(iii) **N** is a model for ϕ, i.e., **N** $\models \phi$ (L_m).

(iv) **N** is a model for $\neg\phi$, i.e., **N** $\models \neg\phi$ (L_{nm}).

(v) $\neg\phi$ is provable from **NT**, i.e., **NT** $\vdash \neg\phi$ (L_{np}).

(vi) $\neg\phi$ is valid, i.e., $\models \neg\phi$ (L_{us}).

Sentence Classification

(i) ϕ is valid, i.e., $\models \phi$ (L_v).

(ii) ϕ is provable from **NT**, i.e., **NT** $\vdash \phi$ (L_p).

(iii) **N** is a model for ϕ, i.e., **N** $\models \phi$ (L_m).

(iv) **N** is a model for $\neg\phi$, i.e., **N** $\models \neg\phi$ (L_{nm}).

(v) $\neg\phi$ is provable from **NT**, i.e., **NT** $\vdash \neg\phi$ (L_{np}).

(vi) $\neg\phi$ is valid, i.e., $\models \neg\phi$ (L_{us}).

Sentence Classification

(i) ϕ is valid, i.e., $\models \phi$ (L_v).
(ii) ϕ is provable from **NT**, i.e., **NT** $\vdash \phi$ (L_p).
(iii) **N** is a model for ϕ, i.e., **N** $\models \phi$ (L_m).
(iv) **N** is a model for $\neg\phi$, i.e., **N** $\models \neg\phi$ (L_{nm}).
(v) $\neg\phi$ is provable from **NT**, i.e., **NT** $\vdash \neg\phi$ (L_{np}).
(vi) $\neg\phi$ is valid, i.e., $\models \neg\phi$ (L_{us}).

Sentence Classification

(i) ϕ is valid, i.e., $\models \phi$ (L_v).

(ii) ϕ is provable from **NT**, i.e., **NT** $\vdash \phi$ (L_p).

(iii) **N** is a model for ϕ, i.e., **N** $\models \phi$ (L_m).

(iv) **N** is a model for $\neg\phi$, i.e., **N** $\models \neg\phi$ (L_{nm}).

(v) $\neg\phi$ is provable from **NT**, i.e., **NT** $\vdash \neg\phi$ (L_{np}).

(vi) $\neg\phi$ is valid, i.e., $\models \neg\phi$ (L_{us}).

Outline

1. Number-theoretic encoding of computation
2. Undecidability
   - Sentence Classification
   - Recursive Inseparability
3. Incompleteness
   - Gødel's Incompleteness Theorem

Recursive Inseparability

Definition

Two languages L_1 and L_2 are said to be recursively inseparable, if there does not exist a recursive language R such that $L_1 \cap R = \emptyset$ and $L_2 \subset R$.

Theorem

Let $L_1 = \{M : M(M) = \text{"yes"}\}$ and $L_2 = \{M : M(M) = \text{"no"}\}$. L_1 and L_2 are recursively inseparable.

Corollary (Inseparability of halting on empty string)

Let $L_y = \{M : M(\epsilon) = \text{"yes"}\}$ and $L_n = \{M : M(\epsilon) = \text{"no"}\}$. L_y and L_n are recursively inseparable.

Recursive Inseparability

Definition

Two languages L_1 and L_2 are said to be recursively inseparable, if there does not exist a recursive language R such that $L_1 \cap R = \emptyset$ and $L_2 \subset R$.

Theorem

Let $L_1 = \{M : M(M) = \text{"yes"}\}$ and $L_2 = \{M : M(M) = \text{"no"}\}$. L_1 and L_2 are recursively inseparable.

Corollary (Inseparability of halting on empty string)

Let $L_y = \{M : M(\epsilon) = \text{"yes"}\}$ and $L_n = \{M : M(\epsilon) = \text{"no"}\}$. L_y and L_n are recursively inseparable.

Recursive Inseparability

Definition

Two languages L_1 and L_2 are said to be recursively inseparable, if there does not exist a recursive language R such that $L_1 \cap R = \emptyset$ and $L_2 \subset R$.

Theorem

Let $L_1 = \{M \ : \ M(M) = \text{“yes”}\}$ and $L_2 = \{M \ : \ M(M) = \text{“no”}\}$. L_1 and L_2 are recursively inseparable.

Corollary (Inseparability of halting on empty string)

Let $L_y = \{M \ : \ M(\epsilon) = \text{“yes”}\}$ and $L_n = \{M \ : \ M(\epsilon) = \text{“no”}\}$. L_y and L_n are recursively inseparable.

Recursive Inseparability of L_p and L_{us}

Theorem

L_p and L_{us} are recursively inseparable.

Proof.

Main idea: Given an Turing Machine M, we construct an expression ϕ_M such that if $M(\epsilon) =$ "*yes*", then **NT** $\vdash \phi_M$ and if $M(\epsilon) =$ "*no*", them ϕ_M is unsatisfiable.
Assume that there exists an algorithm $\mathcal{A}$ to separate L_p from L_{us}, i.e., $\mathcal{A}$ separates the true properties of integers from the unsatisfiable sentences.
But now we can separate L_y and L_n!
Given an arbitrary Turing machine M, construct ϕ_M and then provide it to $\mathcal{A}$! What is ϕ_M?
$\phi_M = \mathbf{NT} \land \psi$, where,

$$\psi = (\exists x)(comp_M(x) \land ((\forall y < x)\neg comp_M(y)) \land \mod(x, b \uparrow 2, b \cdot (|\Sigma| + 1))).$$

ψ states that there exists a smallest integer, which encodes an accepting computation of M. □

Recursive Inseparability of L_p and L_{us}

Theorem

L_p and L_{us} are recursively inseparable.

Proof.

Main idea: Given an Turing Machine M, we construct an expression ϕ_M such that if $M(\epsilon) = ``yes''$, then **NT** $\vdash \phi_M$ and if $M(\epsilon) = ``no''$, them ϕ_M is unsatisfiable.
Assume that there exists an algorithm $\mathcal{A}$ to separate L_p from L_{us}, i.e., $\mathcal{A}$ separates the true properties of integers from the unsatisfiable sentences.
But now we can separate L_y and L_n!
Given an arbitrary Turing machine M, construct ϕ_M and then provide it to $\mathcal{A}$! What is ϕ_M?
$\phi_M = \mathbf{NT} \wedge \psi$, where,

$$\psi = (\exists x)(comp_M(x) \wedge ((\forall y < x)\neg comp_M(y)) \wedge \mod(x, b \uparrow 2, b \cdot (|\Sigma| + 1))).$$

ψ states that there exists a smallest integer, which encodes an accepting computation of M. □

Recursive Inseparability of L_p and L_{us}

Theorem

L_p and L_{us} are recursively inseparable.

Proof.

Main idea: Given an Turing Machine M, we construct an expression ϕ_M such that if $M(\epsilon) = \text{“}yes''$, then **NT** $\vdash \phi_M$ and if $M(\epsilon) = \text{“}no''$, them ϕ_M is unsatisfiable.
Assume that there exists an algorithm $\mathcal{A}$ to separate L_p from L_{us}, i.e., $\mathcal{A}$ separates the true properties of integers from the unsatisfiable sentences.
But now we can separate L_y and L_n!
Given an arbitrary Turing machine M, construct ϕ_M and then provide it to $\mathcal{A}$! What is ϕ_M?
$\phi_M = \mathbf{NT} \wedge \psi$, where,

$$\psi = (\exists x)(comp_M(x) \wedge ((\forall y < x)\neg comp_M(y)) \wedge \mod(x, b \uparrow 2, b \cdot (|\Sigma| + 1))).$$

ψ states that there exists a smallest integer, which encodes an accepting computation of M. □

Recursive Inseparability of L_p and L_{us}

Theorem

L_p and L_{us} are recursively inseparable.

Proof.

Main idea: Given an Turing Machine M, we construct an expression ϕ_M such that if $M(\epsilon) = ``yes''$, then **NT** $\vdash \phi_M$ and if $M(\epsilon) = ``no''$, them ϕ_M is unsatisfiable.
Assume that there exists an algorithm $\mathcal{A}$ to separate L_p from L_{us}, i.e., $\mathcal{A}$ separates the true properties of integers from the unsatisfiable sentences.
But now we can separate L_y and L_n!
Given an arbitrary Turing machine M, construct ϕ_M and then provide it to $\mathcal{A}$! What is ϕ_M?
$\phi_M = \textbf{NT} \land \psi$, where,

$$\psi = (\exists x)(comp_M(x) \land ((\forall y < x)\neg comp_M(y)) \land \mod(x, b \uparrow 2, b \cdot (|\Sigma| + 1))).$$

ψ states that there exists a smallest integer, which encodes an accepting computation of M. □

Recursive Inseparability of L_p and L_{us}

Theorem

L_p and L_{us} are recursively inseparable.

Proof.

Main idea: Given an Turing Machine M, we construct an expression ϕ_M such that if $M(\epsilon) = \text{“}yes''$, then **NT** $\vdash \phi_M$ and if $M(\epsilon) = \text{“}no''$, them ϕ_M is unsatisfiable.
Assume that there exists an algorithm $\mathcal{A}$ to separate L_p from L_{us}, i.e., $\mathcal{A}$ separates the true properties of integers from the unsatisfiable sentences.
But now we can separate L_y and L_n!
Given an arbitrary Turing machine M, construct ϕ_M and then provide it to $\mathcal{A}$! What is ϕ_M?
$\phi_M = \textbf{NT} \wedge \psi$, where,

$$\psi = (\exists x)(comp_M(x) \wedge ((\forall y < x)\neg comp_M(y)) \wedge \mod(x, b \uparrow 2, b \cdot (|\Sigma| + 1))).$$

ψ states that there exists a smallest integer, which encodes an accepting computation of M.

Recursive Inseparability of L_p and L_{us}

Theorem

L_p and L_{us} are recursively inseparable.

Proof.

Main idea: Given an Turing Machine M, we construct an expression ϕ_M such that if $M(\epsilon) = \text{“}yes''$, then **NT** $\vdash \phi_M$ and if $M(\epsilon) = \text{“}no''$, them ϕ_M is unsatisfiable.
Assume that there exists an algorithm $\mathcal{A}$ to separate L_p from L_{us}, i.e., $\mathcal{A}$ separates the true properties of integers from the unsatisfiable sentences.
But now we can separate L_y and L_n!
Given an arbitrary Turing machine M, construct ϕ_M and then provide it to $\mathcal{A}$! What is ϕ_M?
$\phi_M = \mathbf{NT} \wedge \psi$, where,

$$\psi = (\exists x)(comp_M(x) \wedge ((\forall y < x)\neg comp_M(y)) \wedge \mod(x, b \uparrow 2, b \cdot (|\Sigma| + 1))).$$

ψ states that there exists a smallest integer, which encodes an accepting computation of M. □

Recursive Inseparability of L_p and L_{us}

Theorem

L_p and L_{us} are recursively inseparable.

Proof.

Main idea: Given an Turing Machine M, we construct an expression ϕ_M such that if $M(\epsilon) = “yes''$, then **NT** $\vdash \phi_M$ and if $M(\epsilon) = “no''$, them ϕ_M is unsatisfiable.
Assume that there exists an algorithm $\mathcal{A}$ to separate L_p from L_{us}, i.e., $\mathcal{A}$ separates the true properties of integers from the unsatisfiable sentences.
But now we can separate L_y and L_n!
Given an arbitrary Turing machine M, construct ϕ_M and then provide it to $\mathcal{A}$! What is ϕ_M?
$\phi_M = \mathbf{NT} \wedge \psi$, where,

$$\psi = (\exists x)(comp_M(x) \wedge ((\forall y < x)\neg comp_M(y)) \wedge \mod(x, b \uparrow 2, b \cdot (|\Sigma| + 1))).$$

ψ states that there exists a smallest integer, which encodes an accepting computation of M. □

Recursive Inseparability of L_p and L_{us}

Theorem

L_p and L_{us} are recursively inseparable.

Proof.

Main idea: Given an Turing Machine M, we construct an expression ϕ_M such that if $M(\epsilon) = \text{“}yes''$, then **NT** $\vdash \phi_M$ and if $M(\epsilon) = \text{“}no''$, them ϕ_M is unsatisfiable.
Assume that there exists an algorithm $\mathcal{A}$ to separate L_p from L_{us}, i.e., $\mathcal{A}$ separates the true properties of integers from the unsatisfiable sentences.
But now we can separate L_y and L_n!
Given an arbitrary Turing machine M, construct ϕ_M and then provide it to $\mathcal{A}$! What is ϕ_M?
$\phi_M = \mathbf{NT} \wedge \psi$, where,

$$\psi = (\exists x)(comp_M(x) \wedge ((\forall y < x)\neg comp_M(y)) \wedge \mod(x, b \uparrow 2, b \cdot (|\Sigma| + 1))).$$

ψ states that there exists a smallest integer, which encodes an accepting computation of M. □

Recursive Inseparability (contd.)

Proof (contd.)

(i) Assume $M(\epsilon) = ``yes''$. There exists a unique computation of $M$ that starts with $\epsilon$ and halts in the $``yes''$ state. Thus, there exists a unique integer n, such that $\mathbf{N} \models comp_M[x \leftarrow n]$. Therefore, $\mathbf{N} \models (\exists x) comp_M(x)$ and since n is unique, $\mathbf{N} \models (\exists x)(comp_M(x) \land ((\forall y < x) \neg comp_M(y)))$. Since the last two digits of the b-ary expansion of n are $|\Sigma| + 1$ and 0, we have $\mathbf{N} \models \psi$.
Observe that ψ can be written as a bounded sentence in prenex form. Thus, $\mathbf{NT} \vdash \psi$ and hence $\mathbf{NT} \vdash \phi_M$. In other words, $M(\epsilon) = ``yes''$ implies $\mathbf{NT} \vdash \phi_M$.

(ii) Assume that $M(\epsilon) = ``no''$. Using the above argument, we can show that $\mathbf{N} \models \phi'_M$, where

$$\phi'_M = (\exists x')(comp_M(x') \land ((\forall y < x) \neg comp_M(y)) \land \mod (x', b \uparrow 2, b \cdot (|\Sigma| + 2))).$$

Since ϕ'_M can be written as a bounded sentence, $\mathbf{NT} \vdash \phi'_M$. We need to show that ϕ_M and ϕ'_M are inconsistent. But this is obvious! $\square$

Recursive Inseparability (contd.)

Proof (contd.)

(i) Assume $M(\epsilon) =$ "*yes*". There exists a unique computation of M that starts with ϵ and halts in the "*yes*" state. Thus, there exists a unique integer n, such that $\mathbf{N} \models comp_M[x \leftarrow n]$.
Therefore, $\mathbf{N} \models (\exists x) comp_M(x)$ and since n is unique,
$\mathbf{N} \models (\exists x)(comp_M(x) \wedge ((\forall y < x) \neg comp_M(y)))$. Since the last two digits of the b-ary expansion of n are $|\Sigma| + 1$ and 0, we have $\mathbf{N} \models \psi$.
Observe that ψ can be written as a bounded sentence in prenex form. Thus, $\mathbf{NT} \vdash \psi$ and hence $\mathbf{NT} \vdash \phi_M$. In other words, $M(\epsilon) =$ "*yes*" implies $\mathbf{NT} \vdash \phi_M$.

(ii) Assume that $M(\epsilon) =$ "*no*". Using the above argument, we can show that $\mathbf{N} \models \phi'_M$, where

$$\phi'_M = (\exists x')(comp_M(x') \wedge ((\forall y < x) \neg comp_M(y)) \wedge \mod(x', b \uparrow 2, b \cdot (|\Sigma| + 2))).$$

Since ϕ'_M can be written as a bounded sentence, $\mathbf{NT} \vdash \phi'_M$. We need to show that ϕ_M and ϕ'_M are inconsistent. But this is obvious! □

Recursive Inseparability (contd.)

Proof (contd.)

(i) Assume $M(\epsilon) = $ "*yes*". There exists a unique computation of M that starts with ϵ and halts in the "*yes*" state. Thus, there exists a unique integer n, such that $\mathbf{N} \models comp_M[x \leftarrow n]$. Therefore, $\mathbf{N} \models (\exists x) comp_M(x)$ and since n is unique, $\mathbf{N} \models (\exists x)(comp_M(x) \land ((\forall y < x) \neg comp_M(y)))$. Since the last two digits of the b-ary expansion of n are $|\Sigma| + 1$ and 0, we have $\mathbf{N} \models \psi$.
Observe that ψ can be written as a bounded sentence in prenex form. Thus, $\mathbf{NT} \vdash \psi$ and hence $\mathbf{NT} \vdash \phi_M$. In other words, $M(\epsilon) = $ "*yes*" implies $\mathbf{NT} \vdash \phi_M$.

(ii) Assume that $M(\epsilon) = $ "*no*". Using the above argument, we can show that $\mathbf{N} \models \phi'_M$, where

$$\phi'_M = (\exists x')(comp_M(x') \land ((\forall y < x) \neg comp_M(y)) \land \mod(x', b \uparrow 2, b \cdot (|\Sigma| + 2))).$$

Since ϕ'_M can be written as a bounded sentence, $\mathbf{NT} \vdash \phi'_M$. We need to show that ϕ_M and ϕ'_M are inconsistent. But this is obvious! ☐

Recursive Inseparability (contd.)

Proof (contd.)

(i) Assume $M(\epsilon) =$ "*yes*". There exists a unique computation of M that starts with ϵ and halts in the "*yes*" state. Thus, there exists a unique integer n, such that $\mathbf{N} \models comp_M[x \leftarrow n]$.
Therefore, $\mathbf{N} \models (\exists x)comp_M(x)$ and since n is unique,
$\mathbf{N} \models (\exists x)(comp_M(x) \wedge ((\forall y < x)\neg comp_M(y)))$. Since the last two digits of the b-ary expansion of n are $|\Sigma| + 1$ and 0, we have $\mathbf{N} \models \psi$.
Observe that ψ can be written as a bounded sentence in prenex form. Thus, $\mathbf{NT} \vdash \psi$ and hence $\mathbf{NT} \vdash \phi_M$. In other words, $M(\epsilon) =$ "*yes*" implies $\mathbf{NT} \vdash \phi_M$.

(ii) Assume that $M(\epsilon) =$ "*no*". Using the above argument, we can show that $\mathbf{N} \models \phi'_M$, where

$$\phi'_M = (\exists x')(comp_M(x') \wedge ((\forall y < x)\neg comp_M(y)) \wedge \mod(x', b \uparrow 2, b \cdot (|\Sigma| + 2))).$$

Since ϕ'_M can be written as a bounded sentence, $\mathbf{NT} \vdash \phi'_M$. We need to show that ϕ_M and ϕ'_M are inconsistent. But this is obvious! □

Recursive Inseparability (contd.)

Proof (contd.)

(i) Assume $M(\epsilon) =$ "*yes*". There exists a unique computation of M that starts with ϵ and halts in the "*yes*" state. Thus, there exists a unique integer n, such that $\mathbf{N} \models comp_M[x \leftarrow n]$. Therefore, $\mathbf{N} \models (\exists x) comp_M(x)$ and since n is unique, $\mathbf{N} \models (\exists x)(comp_M(x) \land ((\forall y < x) \neg comp_M(y)))$. Since the last two digits of the b-ary expansion of n are $|\Sigma| + 1$ and 0, we have $\mathbf{N} \models \psi$.
Observe that ψ can be written as a bounded sentence in prenex form. Thus, $\mathbf{NT} \vdash \psi$ and hence $\mathbf{NT} \vdash \phi_M$. In other words, $M(\epsilon) =$ "*yes*" implies $\mathbf{NT} \vdash \phi_M$.

(ii) Assume that $M(\epsilon) =$ "*no*". Using the above argument, we can show that $\mathbf{N} \models \phi'_M$, where

$$\phi'_M = (\exists x')(comp_M(x') \land ((\forall y < x) \neg comp_M(y)) \land \mod(x', b \uparrow 2, b \cdot (|\Sigma| + 2))).$$

Since ϕ'_M can be written as a bounded sentence, $\mathbf{NT} \vdash \phi'_M$. We need to show that ϕ_M and ϕ'_M are inconsistent. But this is obvious! $\square$

Recursive Inseparability (contd.)

Proof (contd.)

(i) Assume $M(\epsilon) = $ "yes". There exists a unique computation of M that starts with ϵ and halts in the "yes" state. Thus, there exists a unique integer n, such that $\mathbf{N} \models comp_M[x \leftarrow n]$. Therefore, $\mathbf{N} \models (\exists x) comp_M(x)$ and since n is unique, $\mathbf{N} \models (\exists x)(comp_M(x) \land ((\forall y < x)\neg comp_M(y)))$. Since the last two digits of the b-ary expansion of n are $|\Sigma| + 1$ and 0, we have $\mathbf{N} \models \psi$.
Observe that ψ can be written as a bounded sentence in prenex form. Thus, $\mathbf{NT} \vdash \psi$ and hence $\mathbf{NT} \vdash \phi_M$. In other words, $M(\epsilon) = $ "yes" implies $\mathbf{NT} \vdash \phi_M$.

(ii) Assume that $M(\epsilon) = $ "no". Using the above argument, we can show that $\mathbf{N} \models \phi'_M$, where

$$\phi'_M = (\exists x')(comp_M(x') \land ((\forall y < x)\neg comp_M(y)) \land \mod (x', b \uparrow 2, b \cdot (|\Sigma| + 2))).$$

Since ϕ'_M can be written as a bounded sentence, $\mathbf{NT} \vdash \phi'_M$. We need to show that ϕ_M and ϕ'_M are inconsistent. But this is obvious! □

Recursive Inseparability (contd.)

Proof (contd.)

(i) Assume $M(\epsilon) =$ "*yes*". There exists a unique computation of M that starts with ϵ and halts in the "*yes*" state. Thus, there exists a unique integer n, such that $\mathbf{N} \models comp_M[x \leftarrow n]$. Therefore, $\mathbf{N} \models (\exists x) comp_M(x)$ and since n is unique, $\mathbf{N} \models (\exists x)(comp_M(x) \land ((\forall y < x) \neg comp_M(y)))$. Since the last two digits of the b-ary expansion of n are $|\Sigma| + 1$ and 0, we have $\mathbf{N} \models \psi$.
Observe that ψ can be written as a bounded sentence in prenex form. Thus, $\mathbf{NT} \vdash \psi$ and hence $\mathbf{NT} \vdash \phi_M$. In other words, $M(\epsilon) =$ "*yes*" implies $\mathbf{NT} \vdash \phi_M$.

(ii) Assume that $M(\epsilon) =$ "*no*". Using the above argument, we can show that $\mathbf{N} \models \phi'_M$, where

$$\phi'_M = (\exists x')(comp_M(x') \land ((\forall y < x) \neg comp_M(y)) \land \mod(x', b \uparrow 2, b \cdot (|\Sigma| + 2))).$$

Since ϕ'_M can be written as a bounded sentence, $\mathbf{NT} \vdash \phi'_M$. We need to show that ϕ_M and ϕ'_M are inconsistent. But this is obvious! □

Recursive Inseparability (contd.)

Proof (contd.)

(i) Assume $M(\epsilon) = \text{“}yes\text{''}$. There exists a unique computation of M that starts with ϵ and halts in the “yes” state. Thus, there exists a unique integer n, such that $\mathbf{N} \models comp_M[x \leftarrow n]$. Therefore, $\mathbf{N} \models (\exists x) comp_M(x)$ and since n is unique, $\mathbf{N} \models (\exists x)(comp_M(x) \wedge ((\forall y < x)\neg comp_M(y)))$. Since the last two digits of the b-ary expansion of n are $|\Sigma| + 1$ and 0, we have $\mathbf{N} \models \psi$.
Observe that ψ can be written as a bounded sentence in prenex form. Thus, $\mathbf{NT} \vdash \psi$ and hence $\mathbf{NT} \vdash \phi_M$. In other words, $M(\epsilon) = \text{“}yes\text{''}$ implies $\mathbf{NT} \vdash \phi_M$.

(ii) Assume that $M(\epsilon) = \text{“}no\text{''}$. Using the above argument, we can show that $\mathbf{N} \models \phi'_M$, where

$$\phi'_M = (\exists x')(comp_M(x') \wedge ((\forall y < x)\neg comp_M(y)) \wedge \mod(x', b \uparrow 2, b \cdot (|\Sigma| + 2))).$$

Since ϕ'_M can be written as a bounded sentence, $\mathbf{NT} \vdash \phi'_M$. We need to show that ϕ_M and ϕ'_M are inconsistent. But this is obvious! □

Recursive Inseparability (contd.)

Proof (contd.)

(i) Assume $M(\epsilon) = $ "*yes*". There exists a unique computation of M that starts with ϵ and halts in the "*yes*" state. Thus, there exists a unique integer n, such that $\mathbf{N} \models comp_M[x \leftarrow n]$. Therefore, $\mathbf{N} \models (\exists x) comp_M(x)$ and since n is unique, $\mathbf{N} \models (\exists x)(comp_M(x) \wedge ((\forall y < x)\neg comp_M(y)))$. Since the last two digits of the b-ary expansion of n are $|\Sigma| + 1$ and 0, we have $\mathbf{N} \models \psi$.
Observe that ψ can be written as a bounded sentence in prenex form. Thus, $\mathbf{NT} \vdash \psi$ and hence $\mathbf{NT} \vdash \phi_M$. In other words, $M(\epsilon) = $ "*yes*" implies $\mathbf{NT} \vdash \phi_M$.

(ii) Assume that $M(\epsilon) = $ "*no*". Using the above argument, we can show that $\mathbf{N} \models \phi'_M$, where

$$\phi'_M = (\exists x')(comp_M(x') \wedge ((\forall y < x)\neg comp_M(y)) \wedge \mod(x', b \uparrow 2, b \cdot (|\Sigma| + 2))).$$

Since ϕ'_M can be written as a bounded sentence, $\mathbf{NT} \vdash \phi'_M$. We need to show that ϕ_M and ϕ'_M are inconsistent. But this is obvious! □

Recursive Inseparability (contd.)

Proof (contd.)

(i) Assume $M(\epsilon) = \text{“}yes''$. There exists a unique computation of M that starts with ϵ and halts in the $\text{“}yes''$ state. Thus, there exists a unique integer n, such that $\mathbf{N} \models comp_M[x \leftarrow n]$. Therefore, $\mathbf{N} \models (\exists x) comp_M(x)$ and since n is unique, $\mathbf{N} \models (\exists x)(comp_M(x) \wedge ((\forall y < x) \neg comp_M(y)))$. Since the last two digits of the b-ary expansion of n are $|\Sigma| + 1$ and 0, we have $\mathbf{N} \models \psi$.
Observe that ψ can be written as a bounded sentence in prenex form. Thus, $\mathbf{NT} \vdash \psi$ and hence $\mathbf{NT} \vdash \phi_M$. In other words, $M(\epsilon) = \text{“}yes''$ implies $\mathbf{NT} \vdash \phi_M$.

(ii) Assume that $M(\epsilon) = \text{“}no''$. Using the above argument, we can show that $\mathbf{N} \models \phi'_M$, where

$$\phi'_M = (\exists x')(comp_M(x') \wedge ((\forall y < x) \neg comp_M(y)) \wedge \mod (x', b \uparrow 2, b \cdot (|\Sigma| + 2))).$$

Since ϕ'_M can be written as a bounded sentence, $\mathbf{NT} \vdash \phi'_M$. We need to show that ϕ_M and ϕ'_M are inconsistent. But this is obvious! □

Recursive Inseparability (contd.)

Proof (contd.)

(i) Assume $M(\epsilon) = $ "yes". There exists a unique computation of M that starts with ϵ and halts in the "yes" state. Thus, there exists a unique integer n, such that $\mathbf{N} \models comp_M[x \leftarrow n]$. Therefore, $\mathbf{N} \models (\exists x) comp_M(x)$ and since n is unique, $\mathbf{N} \models (\exists x)(comp_M(x) \land ((\forall y < x)\neg comp_M(y)))$. Since the last two digits of the b-ary expansion of n are $|\Sigma| + 1$ and 0, we have $\mathbf{N} \models \psi$.
Observe that ψ can be written as a bounded sentence in prenex form. Thus, $\mathbf{NT} \vdash \psi$ and hence $\mathbf{NT} \vdash \phi_M$. In other words, $M(\epsilon) = $ "yes" implies $\mathbf{NT} \vdash \phi_M$.

(ii) Assume that $M(\epsilon) = $ "no". Using the above argument, we can show that $\mathbf{N} \models \phi'_M$, where

$$\phi'_M = (\exists x')(comp_M(x') \land ((\forall y < x)\neg comp_M(y)) \land \mod(x', b \uparrow 2, b \cdot (|\Sigma| + 2))).$$

Since ϕ'_M can be written as a bounded sentence, $\mathbf{NT} \vdash \phi'_M$. We need to show that ϕ_M and ϕ'_M are inconsistent. But this is obvious! □

Recursive Inseparability (contd.)

Proof (contd.)

(i) Assume $M(\epsilon) = \text{"}yes''$. There exists a unique computation of M that starts with ϵ and halts in the $\text{"}yes''$ state. Thus, there exists a unique integer n, such that $\mathbf{N} \models comp_M[x \leftarrow n]$.
Therefore, $\mathbf{N} \models (\exists x) comp_M(x)$ and since n is unique,
$\mathbf{N} \models (\exists x)(comp_M(x) \wedge ((\forall y < x) \neg comp_M(y)))$. Since the last two digits of the b-ary expansion of n are $|\Sigma| + 1$ and 0, we have $\mathbf{N} \models \psi$.
Observe that ψ can be written as a bounded sentence in prenex form. Thus, $\mathbf{NT} \vdash \psi$ and hence $\mathbf{NT} \vdash \phi_M$. In other words, $M(\epsilon) = \text{"}yes''$ implies $\mathbf{NT} \vdash \phi_M$.

(ii) Assume that $M(\epsilon) = \text{"}no''$. Using the above argument, we can show that $\mathbf{N} \models \phi'_M$, where

$$\phi'_M = (\exists x')(comp_M(x') \wedge ((\forall y < x) \neg comp_M(y)) \wedge \mod(x', b \uparrow 2, b \cdot (|\Sigma| + 2))).$$

Since ϕ'_M can be written as a bounded sentence, $\mathbf{NT} \vdash \phi'_M$. We need to show that ϕ_M and ϕ'_M are inconsistent. But this is obvious! □

Recursive Inseparability (contd.)

Proof (contd.)

(i) Assume $M(\epsilon) =$ "yes". There exists a unique computation of M that starts with ϵ and halts in the "yes" state. Thus, there exists a unique integer n, such that $\mathbf{N} \models comp_M[x \leftarrow n]$. Therefore, $\mathbf{N} \models (\exists x) comp_M(x)$ and since n is unique, $\mathbf{N} \models (\exists x)(comp_M(x) \wedge ((\forall y < x) \neg comp_M(y)))$. Since the last two digits of the b-ary expansion of n are $|\Sigma| + 1$ and 0, we have $\mathbf{N} \models \psi$.
Observe that ψ can be written as a bounded sentence in prenex form. Thus, $\mathbf{NT} \vdash \psi$ and hence $\mathbf{NT} \vdash \phi_M$. In other words, $M(\epsilon) =$ "yes" implies $\mathbf{NT} \vdash \phi_M$.

(ii) Assume that $M(\epsilon) =$ "no". Using the above argument, we can show that $\mathbf{N} \models \phi'_M$, where

$$\phi'_M = (\exists x')(comp_M(x') \wedge ((\forall y < x) \neg comp_M(y)) \wedge \mod(x', b \uparrow 2, b \cdot (|\Sigma| + 2))).$$

Since ϕ'_M can be written as a bounded sentence, $\mathbf{NT} \vdash \phi'_M$. We need to show that ϕ_M and ϕ'_M are inconsistent. But this is obvious! □

Recursive Inseparability (contd.)

Proof (contd.)

(i) Assume $M(\epsilon) = \text{``}yes''$. There exists a unique computation of $M$ that starts with $\epsilon$ and halts in the $\text{``}yes''$ state. Thus, there exists a unique integer n, such that $\mathbf{N} \models comp_M[x \leftarrow n]$.
Therefore, $\mathbf{N} \models (\exists x) comp_M(x)$ and since n is unique,
$\mathbf{N} \models (\exists x)(comp_M(x) \wedge ((\forall y < x) \neg comp_M(y)))$. Since the last two digits of the b-ary expansion of n are $|\Sigma| + 1$ and 0, we have $\mathbf{N} \models \psi$.
Observe that ψ can be written as a bounded sentence in prenex form. Thus, $\mathbf{NT} \vdash \psi$ and hence $\mathbf{NT} \vdash \phi_M$. In other words, $M(\epsilon) = \text{``}yes''$ implies $\mathbf{NT} \vdash \phi_M$.

(ii) Assume that $M(\epsilon) = \text{``}no''$. Using the above argument, we can show that $\mathbf{N} \models \phi'_M$, where

$$\phi'_M = (\exists x')(comp_M(x') \wedge ((\forall y < x) \neg comp_M(y)) \wedge \mod(x', b \uparrow 2, b \cdot (|\Sigma| + 2))).$$

Since ϕ'_M can be written as a bounded sentence, $\mathbf{NT} \vdash \phi'_M$. We need to show that ϕ_M and ϕ'_M are inconsistent. But this is obvious! □

Recursive Inseparability (contd.)

Proof (contd.)

(i) Assume $M(\epsilon) = $ "*yes*". There exists a unique computation of M that starts with ϵ and halts in the "*yes*" state. Thus, there exists a unique integer n, such that $\mathbf{N} \models comp_M[x \leftarrow n]$. Therefore, $\mathbf{N} \models (\exists x) comp_M(x)$ and since n is unique, $\mathbf{N} \models (\exists x)(comp_M(x) \land ((\forall y < x) \neg comp_M(y)))$. Since the last two digits of the b-ary expansion of n are $|\Sigma| + 1$ and 0, we have $\mathbf{N} \models \psi$.
Observe that ψ can be written as a bounded sentence in prenex form. Thus, $\mathbf{NT} \vdash \psi$ and hence $\mathbf{NT} \vdash \phi_M$. In other words, $M(\epsilon) = $ "*yes*" implies $\mathbf{NT} \vdash \phi_M$.

(ii) Assume that $M(\epsilon) = $ "*no*". Using the above argument, we can show that $\mathbf{N} \models \phi'_M$, where

$$\phi'_M = (\exists x')(comp_M(x') \land ((\forall y < x) \neg comp_M(y)) \land \mod (x', b \uparrow 2, b \cdot (|\Sigma| + 2))).$$

Since ϕ'_M can be written as a bounded sentence, $\mathbf{NT} \vdash \phi'_M$. We need to show that ϕ_M and ϕ'_M are inconsistent. But this is obvious! □

Undecidability of some basic problems

Theorem

The following questions, regarding a given sentence ϕ, are undecidable:

(i) *Is* $\models \phi$?
(ii) *Is* $\vdash \phi$?
(iii) *Does* $\mathbf{N} \models \phi$?
(iv) *Does* $\mathbf{NT} \vdash \phi$?

Undecidability of some basic problems

Theorem

The following questions, regarding a given sentence ϕ, are undecidable:

(i) *Is* $\models \phi$?
(ii) *Is* $\vdash \phi$?
(iii) *Does* **N** $\models \phi$?
(iv) *Does* **NT** $\vdash \phi$?

Undecidability of some basic problems

Theorem

The following questions, regarding a given sentence ϕ, are undecidable:

(i) *Is* $\models \phi$?
(ii) *Is* $\vdash \phi$?
(iii) *Does* **N** $\models \phi$?
(iv) *Does* **NT** $\vdash \phi$?

Undecidability of some basic problems

Theorem

The following questions, regarding a given sentence ϕ, are undecidable:

(i) *Is* $\models \phi$?
(ii) *Is* $\vdash \phi$?
(iii) *Does* **N** $\models \phi$?
(iv) *Does* **NT** $\vdash \phi$?

Undecidability of some basic problems

Theorem

The following questions, regarding a given sentence ϕ, are undecidable:

(i) *Is* $\models \phi$*?*
(ii) *Is* $\vdash \phi$*?*
(iii) *Does* **N** $\models \phi$*?*
(iv) *Does* **NT** $\vdash \phi$*?*

Outline

Gødel's Incompleteness Theorem

Theorem

There does not exist a recursively enumerable set of axioms Ξ, such that for all sentences ϕ, $\Xi \vdash \phi$ if and only if $\mathbf{N} \models \phi$.

Proof.

Let L_{pr} denote the set of all proofs from Ξ.
Since Ξ is recursively enumerable, so is L_{pr}: For each expression in the sequence, check whether it is

(i) a logical axiom,
(ii) it follows by *modus ponens*,
(iii) it is in Ξ.

Since L_{pr} is recursively enumerable, there exists a Turing machine that enumerates it. It follows that there exists a Turing Machine that enumerates $\{\phi : \Xi \vdash \phi\}$. By the hypothesis, there exists a Turing machine that enumerates $L_e = \{\phi : \mathbf{N} \models \phi\}$. Hence, L_e is recursively enumerable. Arguing in identical fashion, the language $L_{ne} = \{\phi : \mathbf{N} \models \neg\phi\}$ is recursively enumerable. This means that L_e and L_{ne} are recursive! □

Gødel's Incompleteness Theorem

Theorem

There does not exist a recursively enumerable set of axioms Ξ*, such that for all sentences* ϕ*,* $\Xi \vdash \phi$ *if and only if* $\mathbf{N} \models \phi$*.*

Proof.

Let L_{pr} denote the set of all proofs from Ξ.
Since Ξ is recursively enumerable, so is L_{pr}: For each expression in the sequence, check whether it is

(i) a logical axiom,
(ii) it follows by *modus ponens*,
(iii) it is in Ξ.

Since L_{pr} is recursively enumerable, there exists a Turing machine that enumerates it. It follows that there exists a Turing Machine that enumerates $\{\phi : \Xi \vdash \phi\}$. By the hypothesis, there exists a Turing machine that enumerates $L_e = \{\phi : \mathbf{N} \models \phi\}$. Hence, L_e is recursively enumerable. Arguing in identical fashion, the language $L_{ne} = \{\phi : \mathbf{N} \models \neg\phi\}$ is recursively enumerable. This means that L_e and L_{ne} are recursive! □

Gødel's Incompleteness Theorem

Theorem

There does not exist a recursively enumerable set of axioms Ξ*, such that for all sentences* ϕ*,* $\Xi \vdash \phi$ *if and only if* $\mathbf{N} \models \phi$*.*

Proof.

Let L_{pr} denote the set of all proofs from Ξ.
Since Ξ is recursively enumerable, so is L_{pr}: For each expression in the sequence, check whether it is

(i) a logical axiom,
(ii) it follows by *modus ponens*,
(iii) it is in Ξ.

Since L_{pr} is recursively enumerable, there exists a Turing machine that enumerates it. It follows that there exists a Turing Machine that enumerates $\{\phi : \Xi \vdash \phi\}$. By the hypothesis, there exists a Turing machine that enumerates $L_e = \{\phi : \mathbf{N} \models \phi\}$. Hence, L_e is recursively enumerable. Arguing in identical fashion, the language $L_{ne} = \{\phi : \mathbf{N} \models \neg\phi\}$ is recursively enumerable. This means that L_e and L_{ne} are recursive! □

Gødel's Incompleteness Theorem

Theorem

There does not exist a recursively enumerable set of axioms Ξ, such that for all sentences ϕ, $\Xi \vdash \phi$ if and only if $\mathbf{N} \models \phi$.

Proof.

Let L_{pr} denote the set of all proofs from Ξ.
Since Ξ is recursively enumerable, so is L_{pr}: For each expression in the sequence, check whether it is

(i) a logical axiom,
(ii) it follows by *modus ponens*,
(iii) it is in Ξ.

Since L_{pr} is recursively enumerable, there exists a Turing machine that enumerates it. It follows that there exists a Turing Machine that enumerates $\{\phi : \Xi \vdash \phi\}$. By the hypothesis, there exists a Turing machine that enumerates $L_e = \{\phi : \mathbf{N} \models \phi\}$. Hence, L_e is recursively enumerable. Arguing in identical fashion, the language $L_{ne} = \{\phi : \mathbf{N} \models \neg\phi\}$ is recursively enumerable. This means that L_e and L_{ne} are recursive! □

Gødel's Incompleteness Theorem

Theorem

There does not exist a recursively enumerable set of axioms Ξ*, such that for all sentences* ϕ*,* $\Xi \vdash \phi$ *if and only if* $\mathbf{N} \models \phi$*.*

Proof.

Let L_{pr} denote the set of all proofs from Ξ.
Since Ξ is recursively enumerable, so is L_{pr}: For each expression in the sequence, check whether it is

(i) a logical axiom,
(ii) it follows by *modus ponens*,
(iii) it is in Ξ.

Since L_{pr} is recursively enumerable, there exists a Turing machine that enumerates it. It follows that there exists a Turing Machine that enumerates $\{\phi : \Xi \vdash \phi\}$. By the hypothesis, there exists a Turing machine that enumerates $L_e = \{\phi : \mathbf{N} \models \phi\}$. Hence, L_e is recursively enumerable. Arguing in identical fashion, the language $L_{ne} = \{\phi : \mathbf{N} \models \neg\phi\}$ is recursively enumerable. This means that L_e and L_{ne} are recursive! □

Gødel's Incompleteness Theorem

Theorem

There does not exist a recursively enumerable set of axioms Ξ, such that for all sentences ϕ, $\Xi \vdash \phi$ if and only if $\mathbf{N} \models \phi$.

Proof.

Let L_{pr} denote the set of all proofs from Ξ.
Since Ξ is recursively enumerable, so is L_{pr}: For each expression in the sequence, check whether it is

(i) a logical axiom,
(ii) it follows by *modus ponens*,
(iii) it is in Ξ.

Since L_{pr} is recursively enumerable, there exists a Turing machine that enumerates it. It follows that there exists a Turing Machine that enumerates $\{\phi : \Xi \vdash \phi\}$. By the hypothesis, there exists a Turing machine that enumerates $L_e = \{\phi : \mathbf{N} \models \phi\}$. Hence, L_e is recursively enumerable. Arguing in identical fashion, the language $L_{ne} = \{\phi : \mathbf{N} \models \neg\phi\}$ is recursively enumerable. This means that L_e and L_{ne} are recursive! □

Gødel's Incompleteness Theorem

Theorem

There does not exist a recursively enumerable set of axioms Ξ*, such that for all sentences* ϕ*,* $\Xi \vdash \phi$ *if and only if* $\mathbf{N} \models \phi$*.*

Proof.

Let L_{pr} denote the set of all proofs from Ξ.
Since Ξ is recursively enumerable, so is L_{pr}: For each expression in the sequence, check whether it is

(i) a logical axiom,
(ii) it follows by *modus ponens*,
(iii) it is in Ξ.

Since L_{pr} is recursively enumerable, there exists a Turing machine that enumerates it. It follows that there exists a Turing Machine that enumerates $\{\phi : \Xi \vdash \phi\}$. By the hypothesis, there exists a Turing machine that enumerates $L_e = \{\phi : \mathbf{N} \models \phi\}$. Hence, L_e is recursively enumerable. Arguing in identical fashion, the language $L_{ne} = \{\phi : \mathbf{N} \models \neg\phi\}$ is recursively enumerable. This means that L_e and L_{ne} are recursive! □

Gødel's Incompleteness Theorem

Theorem

There does not exist a recursively enumerable set of axioms Ξ, such that for all sentences ϕ, $\Xi \vdash \phi$ if and only if $\mathbf{N} \models \phi$.

Proof.

Let L_{pr} denote the set of all proofs from Ξ.
Since Ξ is recursively enumerable, so is L_{pr}: For each expression in the sequence, check whether it is

(i) a logical axiom,
(ii) it follows by *modus ponens*,
(iii) it is in Ξ.

Since L_{pr} is recursively enumerable, there exists a Turing machine that enumerates it. It follows that there exists a Turing Machine that enumerates $\{\phi : \Xi \vdash \phi\}$. By the hypothesis, there exists a Turing machine that enumerates $L_e = \{\phi : \mathbf{N} \models \phi\}$. Hence, L_e is recursively enumerable. Arguing in identical fashion, the language $L_{ne} = \{\phi : \mathbf{N} \models \neg\phi\}$ is recursively enumerable. This means that L_e and L_{ne} are recursive! □

Consequences of Gødel's Incompleteness Theorem

Non-existence

There **cannot** exist a recursively enumerable (much less recursive) set of axioms that captures all and only the true properties of integers. Any sound system **must** be incomplete, i.e., there must exist a true property of integers that cannot be proved by it.

Categorization

The languages $L = \{\phi : \mathbf{N} \models \phi\}$ and $L^c = \{\phi : \mathbf{N} \models \neg\phi\}$ are not recursively enumerable. Thus L and L^c are neither **RE** nor **coRE**!

Consequences of Gødel's Incompleteness Theorem

Non-existence

There **cannot** exist a recursively enumerable (much less recursive) set of axioms that captures all and only the true properties of integers. Any sound system **must** be incomplete, i.e., there must exist a true property of integers that cannot be proved by it.

Categorization

The languages $L = \{\phi : \mathbf{N} \models \phi\}$ and $L^c = \{\phi : \mathbf{N} \models \neg\phi\}$ are not recursively enumerable. Thus L and L^c are neither **RE** nor **coRE**!

Consequences of Gødel's Incompleteness Theorem

Non-existence

There **cannot** exist a recursively enumerable (much less recursive) set of axioms that captures all and only the true properties of integers. Any sound system **must** be incomplete, i.e., there must exist a true property of integers that cannot be proved by it.

Categorization

The languages $L = \{\phi : \mathbf{N} \models \phi\}$ and $L^c = \{\phi : \mathbf{N} \models \neg\phi\}$ are not recursively enumerable. Thus L and L^c are neither **RE** nor **coRE**!

Consequences of Gødel's Incompleteness Theorem

Non-existence

There **cannot** exist a recursively enumerable (much less recursive) set of axioms that captures all and only the true properties of integers. Any sound system **must** be incomplete, i.e., there must exist a true property of integers that cannot be proved by it.

Categorization

The languages $L = \{\phi : \mathbf{N} \models \phi\}$ and $L^c = \{\phi : \mathbf{N} \models \neg\phi\}$ are not recursively enumerable. Thus L and L^c are neither **RE** nor **coRE**!

Consequences of Gødel's Incompleteness Theorem

Non-existence

There **cannot** exist a recursively enumerable (much less recursive) set of axioms that captures all and only the true properties of integers. Any sound system **must** be incomplete, i.e., there must exist a true property of integers that cannot be proved by it.

Categorization

The languages $L = \{\phi : \mathbf{N} \models \phi\}$ and $L^c = \{\phi : \mathbf{N} \models \neg\phi\}$ are not recursively enumerable. Thus L and L^c are neither **RE** nor **coRE**!