Primitive recursion

Definition

Given a binary function $\beta(m, n)$ and any $a \in \mathbb{N}$ define the function $\varphi(n)$ as follows;

$$\varphi(0) := a \text{ and } \varphi(n+1) := \beta(n, \varphi(n)).$$

Primitive recursion

Definition

Given a binary function $\beta(m, n)$ and any $a \in \mathbb{N}$ define the function $\varphi(n)$ as follows;

$$\varphi(0) := a \text{ and } \varphi(n+1) := \beta(n, \varphi(n)).$$

- Note how the successor function $\sigma_0(n) = n + 1$ is built directly into the definition of primitive recursion.

Primitive recursion

Definition

Given a binary function $\beta(m, n)$ and any $a \in \mathbb{N}$ define the function $\varphi(n)$ as follows;

$$\varphi(0) := a \text{ and } \varphi(n+1) := \beta(n, \varphi(n)).$$

- Note how the successor function $\sigma_0(n) = n + 1$ is built directly into the definition of primitive recursion.
- The class PR is built up from σ_0 the constant functions and the projections $\pi_i(a_1, ..., a_i, ..., a_n) = a_i$ by

Primitive recursion

Definition

Given a binary function $\beta(m, n)$ and any $a \in \mathbb{N}$ define the function $\varphi(n)$ as follows;

$$\varphi(0) := a \text{ and } \varphi(n+1) := \beta(n, \varphi(n)).$$

- Note how the successor function $\sigma_0(n) = n + 1$ is built directly into the definition of primitive recursion.
- The class PR is built up from σ_0 the constant functions and the projections $\pi_i(a_1, ..., a_i, ..., a_n) = a_i$ by substitution

Primitive recursion

Definition

Given a binary function $\beta(m, n)$ and any $a \in \mathbb{N}$ define the function $\varphi(n)$ as follows;

$$\varphi(0) := a \text{ and } \varphi(n+1) := \beta(n, \varphi(n)).$$

- Note how the successor function $\sigma_0(n) = n + 1$ is built directly into the definition of primitive recursion.
- The class PR is built up from σ_0 the constant functions and the projections $\pi_i(a_1, ..., a_i, ..., a_n) = a_i$ by substitution and primitive recursion.

Two surprisingly useful functions

We define by primitive recursion sg and $\bar{\mathrm{sg}}$.

- $\mathrm{sg}(0) = 0$ with $\mathrm{sg}(\mathrm{n} + 1) = 1$.
- $\bar{\mathrm{sg}}(0) = 1$ with $\bar{\mathrm{sg}}(n + 1) = 0$.

Two surprisingly useful functions

We define by primitive recursion sg and $\bar{\mathrm{sg}}$.

- $\mathrm{sg}(0) = 0$ with $\mathrm{sg}(\mathrm{n} + 1) = 1$.
- $\bar{\mathrm{sg}}(0) = 1$ with $\bar{\mathrm{sg}}(n + 1) = 0$.
- Exponent notation is used e.g. $11^{\mathrm{sg}} = 1$ and $2^{\bar{\mathrm{sg}}} = 0$.

Examples

Well known primitive recursive functions. Takes some work!

1. The n-th prime p_n

Examples

Well known primitive recursive functions. Takes some work!

1. The n-th prime p_n
2. The exponent $\exp_l(n)$ of the l-th prime in the prime factorization of $n \in \mathbb{N}$.

Examples

Well known primitive recursive functions. Takes some work!

1. The n-th prime p_n
2. The exponent $\exp_l(n)$ of the l-th prime in the prime factorization of $n \in \mathbb{N}$.
3. The residue $\mathrm{res(a, n)}$ of a modulo n.

Examples

Well known primitive recursive functions. Takes some work!

1. The n-th prime p_n
2. The exponent $\exp_l(n)$ of the l-th prime in the prime factorization of $n \in \mathbb{N}$.
3. The residue $\mathrm{res(a, n)}$ of a modulo n.
4. Restricted subtraction $a \dot{-} b$.

Examples

- If we let

$$S(n) = \sum_{i=1}^{n} \bar{\mathrm{sg}}(\mathrm{res(n, i)})$$

Examples

- If we let

$$S(n) = \sum_{i=1}^{n} \bar{\mathrm{sg}}(\mathrm{res(n,i)})$$

then $S(n)$ yields the number of divisors of n.

Examples

- If we let
$$S(n) = \sum_{i=1}^{n} \bar{\mathrm{sg}}(\mathrm{res(n, i)})$$
then $S(n)$ yields the number of divisors of n.
- If we let
$$\pi(n) = \sum_{i=2}^{n} \bar{\mathrm{sg}}(S(n) \dot{-} 2)$$

Examples

- If we let
$$S(n) = \sum_{i=1}^{n} \bar{\mathrm{sg}}(\mathrm{res(n,i)})$$
then $S(n)$ yields the number of divisors of n.
- If we let
$$\pi(n) = \sum_{i=2}^{n} \bar{\mathrm{sg}}(S(n) \dot{-} 2)$$
then $\pi(n)$ yields the number of primes among $2, ..., n$.

Examples

- If we let
$$S(n) = \sum_{i=1}^{n} \bar{\mathrm{sg}}(\mathrm{res(n, i)})$$
then $S(n)$ yields the number of divisors of n.
- If we let
$$\pi(n) = \sum_{i=2}^{n} \bar{\mathrm{sg}}(S(n) \dot{-} 2)$$
then $\pi(n)$ yields the number of primes among $2, \ldots, n$.

Then the nth prime is the least j such that $\pi(j) = n + 1$.

Bounded search

Suppose a function $\tau(a_1, ..., a_r, n)$ is given which satisfies the following formula.

$$(\forall a_1 \ldots \forall a_r \exists n)(\tau(\vec{a}, n) = 0)$$

Bounded search

Suppose a function $\tau(a_1, ..., a_r, n)$ is given which satisfies the following formula.

$$(\forall a_1 \ldots \forall a_r \exists n)(\tau(\vec{a}, n) = 0)$$

Then the smallest n for which $\tau(a_1, ..., a_r, n) = 0$ is given by the expression below.

$$\sum_{i=0}^{B} \prod_{j=0}^{i} \tau(\vec{a}, j)^{\mathrm{sg}}$$

Bounded search

Suppose a function $\tau(a_1, ..., a_r, n)$ is given which satisfies the following formula.

$$(\forall a_1 \ldots \forall a_r \exists n)(\tau(\vec{a}, n) = 0)$$

Then the smallest n for which $\tau(a_1, ..., a_r, n) = 0$ is given by the expression below.

$$\sum_{i=0}^{B} \prod_{j=0}^{i} \tau(\vec{a}, j)^{\mathrm{sg}}$$

$$= \tau(\vec{a}, 0)^{\mathrm{sg}} + \tau(\vec{a}, 0)^{\mathrm{sg}}\tau(\vec{a}, 1)^{\mathrm{sg}} + \tau(\vec{a}, 0)^{\mathrm{sg}}\tau(\vec{a}, 1)^{\mathrm{sg}}\tau(\vec{a}, 2)^{\mathrm{sg}} + \ldots$$

$$+ \tau(\vec{a}, 0)^{\mathrm{sg}}\tau(\vec{a}, 1)^{\mathrm{sg}} \ldots \tau(\vec{a}, B)^{\mathrm{sg}}$$

Bounded search

The following notation is fairly standard.

$$\mu_j[\tau(\vec{a}, j) = 0] = \sum_{i=0}^{B} \prod_{j=0}^{i} \tau(\vec{a}, j)^{\mathrm{sg}}$$

Bounded search

The following notation is fairly standard.

$$\mu_j[\tau(\vec{a}, j) = 0] = \sum_{i=0}^{B} \prod_{j=0}^{i} \tau(\vec{a}, j)^{\mathrm{sg}}$$

- The bound B is specific to the function τ.

Bounded search

The following notation is fairly standard.

$$\mu_j[\tau(\vec{a}, j) = 0] = \sum_{i=0}^{B} \prod_{j=0}^{i} \tau(\vec{a}, j)^{\mathrm{sg}}$$

- The bound B is specific to the function τ.
- For example the nth prime p_n satisfies $p_n < 2^{2^n}$.

Bounded search

The following notation is fairly standard.

$$\mu_j[\tau(\vec{a}, j) = 0] = \sum_{i=0}^{B} \prod_{j=0}^{i} \tau(\vec{a}, j)^{\mathrm{sg}}$$

- The bound B is specific to the function τ.
- For example the nth prime p_n satisfies $p_n < 2^{2^n}$.
- This provides the bound $B = 2^{2^n}$.

The length of $n \in \mathbb{N}$

Consider the problem of finding the index l of the largest prime dividing $n \in \mathbb{N}$ and suppose $n > 1$.

The length of $n \in \mathbb{N}$

Consider the problem of finding the index l of the largest prime dividing $n \in \mathbb{N}$ and suppose $n > 1$.

- Take the sign of the sum

$$\mathrm{sg}(\mathsf{exp}_{\mathrm{l}}(\mathrm{n}) + \mathsf{exp}_{2}(\mathrm{n}) + \ldots + \mathsf{exp}_{\mathrm{n}}(\mathrm{n})).$$

The length of $n \in \mathbb{N}$

Consider the problem of finding the index l of the largest prime dividing $n \in \mathbb{N}$ and suppose $n > 1$.

- Take the sign of the sum

$$\mathrm{sg}(\mathsf{exp}_{\mathrm{l}}(\mathrm{n}) + \mathsf{exp}_{2}(\mathrm{n}) + \ldots + \mathsf{exp}_{\mathrm{n}}(\mathrm{n})).$$

- What we seek is the smallest j such that

$$\mathrm{sg}(\mathsf{exp}_{\mathrm{j+1}}(\mathrm{n}) + \mathsf{exp}_{\mathrm{j+2}}(\mathrm{n}) + \ldots + \mathsf{exp}_{\mathrm{n}}(\mathrm{n})) = 0.$$

The length of $n \in \mathbb{N}$

The following primitive recursive function yields the index of the largest prime divisor of the natural number n.

$$\mathrm{long}(\mathrm{n}) = \sum_{\mathrm{k}=0}^{\mathrm{n}} \prod_{\mathrm{j}=0}^{\mathrm{k}} \mathrm{sg}\left(\sum_{\mathrm{l}=\mathrm{j}+1}^{\mathrm{n}} \mathsf{exp}_{\mathrm{l}}(\mathrm{n}) \right)$$

The length of $n \in \mathbb{N}$

The following primitive recursive function yields the index of the largest prime divisor of the natural number n.

$$\mathrm{long(n)} = \sum_{\mathrm{k=0}}^{\mathrm{n}} \prod_{\mathrm{j=0}}^{\mathrm{k}} \mathrm{sg} \left(\sum_{\mathrm{l=j+1}}^{\mathrm{n}} \mathsf{exp}_{\mathrm{l}}(\mathrm{n}) \right)$$

- The bound n is sufficient since $n < p_n$.

The length of $n \in \mathbb{N}$

The following primitive recursive function yields the index of the largest prime divisor of the natural number n.

$$\mathrm{long(n)} = \sum_{\mathrm{k=0}}^{\mathrm{n}} \prod_{\mathrm{j=0}}^{\mathrm{k}} \mathrm{sg}\left(\sum_{\mathrm{l=j+1}}^{\mathrm{n}} \mathsf{exp}_{\mathrm{l}}(\mathrm{n}) \right)$$

- The bound n is sufficient since $n < p_n$.
- This yields the smallest j such that $\mathrm{exp}_l(n) = 0$ if $l > j$.

Other forms of recursion

Course-of-values recursion

Definition

Given a binary function β and any $a \in \mathbb{N}$ define φ as follows;

$$\varphi(0) = a \text{ and } \varphi(n+1) = \beta\left(\prod_{i=0}^{n} p_i^{\varphi(i)},\ n\right).$$

We can prove that such a definition yields a primitive recursive function.

Course-of-values recursion

Definition of φ

$$\varphi(0) = a \text{ and } \varphi(n+1) = \beta\left(\prod_{i=0}^{n} p_i^{\varphi(i)},\ n\right)$$

Course-of-values recursion

Definition of φ

$$\varphi(0) = a \text{ and } \varphi(n+1) = \beta\left(\prod_{i=0}^{n} p_i^{\varphi(i)},\ n \right)$$

Define a function $\psi(n) = \prod_{i=0}^{n} p_i^{\varphi(n)}$. Then $\psi(0) = 2^a$ and

Course-of-values recursion

Definition of φ

$$\varphi(0) = a \text{ and } \varphi(n+1) = \beta\left(\prod_{i=0}^{n} p_i^{\varphi(i)},\ n \right)$$

Define a function $\psi(n) = \prod_{i=0}^{n} p_i^{\varphi(n)}$. Then $\psi(0) = 2^a$ and

$$\begin{aligned} \psi(n+1) &= \psi(n) \cdot p_{n+1}^{\varphi(n+1)} \\ &= \psi(n) \cdot p_{n+1}^{\beta(\psi(n),n)} \end{aligned}$$

Course-of-values recursion

Definition of φ

$$\varphi(0) = a \text{ and } \varphi(n+1) = \beta\left(\prod_{i=0}^{n} p_i^{\varphi(i)},\ n\right)$$

Define a function $\psi(n) = \prod_{i=0}^{n} p_i^{\varphi(n)}$. Then $\psi(0) = 2^a$ and

$$\begin{aligned} \psi(n+1) &= \psi(n) \cdot p_{n+1}^{\varphi(n+1)} \\ &= \psi(n) \cdot p_{n+1}^{\beta(\psi(n),n)} \end{aligned}$$

Thus ψ is primitive recursive and $\varphi(n) = \exp_n \psi(n)$ hence φ is also primitive recursive.

Another form of recursion

Simultaneous recursion

Definition

Given $a, b \in \mathbb{N}$ and binary functions β_1 and β_2 define σ_1 and σ_2; $\sigma_1(0) = a,\ \sigma_2(0) = b$ with

$$\begin{aligned} \sigma_1(n+1) &= \beta_1(\sigma_1(n), \sigma_2(n)) \\ \sigma_2(n+1) &= \beta_2(\sigma_1(n), \sigma_2(n)). \end{aligned}$$

Another form of recursion

Simultaneous recursion

Definition

Given $a, b \in \mathbb{N}$ and binary functions β_1 and β_2 define σ_1 and σ_2; $\sigma_1(0) = a,\ \sigma_2(0) = b$ with

$$\begin{aligned} \sigma_1(n+1) &= \beta_1(\sigma_1(n), \sigma_2(n)) \\ \sigma_2(n+1) &= \beta_2(\sigma_1(n), \sigma_2(n)). \end{aligned}$$

We can prove that such a definition yields primitive recursive functins σ_1 and σ_2.

Simultaneous recursion

$\sigma_1(0) = a$ and $\sigma_2(0) = b$

$$\begin{aligned} \sigma_1(n+1) &= \beta_1(\sigma_1(n), \sigma_2(n)) \\ \sigma_2(n+1) &= \beta_2(\sigma_1(n), \sigma_2(n)). \end{aligned}$$

Simultaneous recursion

$\sigma_1(0) = a$ and $\sigma_2(0) = b$

$$\begin{aligned} \sigma_1(n+1) &= \beta_1(\sigma_1(n), \sigma_2(n)) \\ \sigma_2(n+1) &= \beta_2(\sigma_1(n), \sigma_2(n)). \end{aligned}$$

Define a function $\psi(n) = p_1^{\sigma_1(n)} \cdot p_2^{\sigma_2(n)}$. So then $\psi(0) = p_1^a \cdot p_2^b$ and

Simultaneous recursion

$\sigma_1(0) = a$ and $\sigma_2(0) = b$

$$\begin{aligned}\sigma_1(n+1) &= \beta_1(\sigma_1(n), \sigma_2(n)) \\ \sigma_2(n+1) &= \beta_2(\sigma_1(n), \sigma_2(n)).\end{aligned}$$

Define a function $\psi(n) = p_1^{\sigma_1(n)} \cdot p_2^{\sigma_2(n)}$. So then $\psi(0) = p_1^a \cdot p_2^b$ and

$$\begin{aligned}\psi(n+1) &= p_1^{\sigma_1(n+1)} \cdot p_2^{\sigma_2(n+1)} \\ &= p_1^{\beta_1(\sigma_1(n),\sigma_2(n))} \cdot p_2^{\beta_2(\sigma_1(n),\sigma_2(n)} \\ &= p_1^{\beta_1(\exp_1 \psi(n), \exp_2 \psi(n))} \cdot p_2^{\beta_2(\exp_1 \psi(n), \exp_2 \psi(n))}\end{aligned}$$

Simultaneous recursion

$\sigma_1(0) = a$ and $\sigma_2(0) = b$

$$\begin{aligned} \sigma_1(n+1) &= \beta_1(\sigma_1(n), \sigma_2(n)) \\ \sigma_2(n+1) &= \beta_2(\sigma_1(n), \sigma_2(n)). \end{aligned}$$

Define a function $\psi(n) = p_1^{\sigma_1(n)} \cdot p_2^{\sigma_2(n)}$. So then $\psi(0) = p_1^a \cdot p_2^b$ and

$$\begin{aligned} \psi(n+1) &= p_1^{\sigma_1(n+1)} \cdot p_2^{\sigma_2(n+1)} \\ &= p_1^{\beta_1(\sigma_1(n),\sigma_2(n))} \cdot p_2^{\beta_2(\sigma_1(n),\sigma_2(n)} \\ &= p_1^{\beta_1(\exp_1 \psi(n),\exp_2 \psi(n))} \cdot p_2^{\beta_2(\exp_1 \psi(n),\exp_2 \psi(n))} \end{aligned}$$

Hence ψ is primitive recursive $\Rightarrow$ σ_1 and σ_2 are primitive recursive.

Nested recursion

Example

Given known functions α, β and γ define $\varphi(0, a) = \alpha(a)$ with

$$\varphi(n+1, a) = \beta(n, \varphi(n, \gamma(n, a, \varphi(n, a)))).$$

Nested recursion

Example

Given known functions α, β and γ define $\varphi(0, a) = \alpha(a)$ with

$$\varphi(n + 1, a) = \beta(n, \varphi(n, \gamma(n, a, \varphi(n, a)))).$$

- This function is indeed primitive recursive.

Nested recursion

Example

Given known functions α, β and γ define $\varphi(0, a) = \alpha(a)$ with

$$\varphi(n+1, a) = \beta(n, \varphi(n, \gamma(n, a, \varphi(n, a)))).$$

- This function is indeed primitive recursive.
- But in such a definition if you have induction on multiple variables then your function may no longer be PR.

General recursive functions

General recursive functions are defined in terms of a system of equations.

General recursive functions

General recursive functions are defined in terms of a system of equations.

- The class of general recursive functions coincides with the class of all computable functions.

General recursive functions

General recursive functions are defined in terms of a system of equations.

- The class of general recursive functions coincides with the class of all computable functions.
- Hence functions of the λ-calculus, Turing computable functions, and so on are all general recursive functions.

General recursive functions

General recursive functions are defined in terms of a system of equations.

- The class of general recursive functions coincides with the class of all computable functions.
- Hence functions of the λ-calculus, Turing computable functions, and so on are all general recursive functions.
- Given a defining system of equations you may not be able to tell if you have a working definition.

General recursive functions

General recursive functions are defined in terms of a system of equations.

- The class of general recursive functions coincides with the class of all computable functions.
- Hence functions of the λ-calculus, Turing computable functions, and so on are all general recursive functions.
- Given a defining system of equations you may not be able to tell if you have a working definition.
- In general the problem is undecidable.

General recursive functions

Given a defining system of equations there are two numbers which are noteworthy.

General recursive functions

Given a defining system of equations there are two numbers which are noteworthy.

The index of the function being defined σ_s and the number A of axioms.

General recursive functions

Given a defining system of equations there are two numbers which are noteworthy.

The index of the function being defined σ_s and the number A of axioms.

Example

1. $\sigma_1(0, x_2) = x_2$
2. $\sigma_1(\sigma_0(x_1), x_2) = \sigma_0(\sigma_1(x_1, x_2))$
3. $\sigma_2(0, x_2) = 0$
4. $\sigma_2(\sigma_0(x_1), x_2) = \sigma_1(\sigma_2(x_1, x_2), x_2)$
5. $\sigma_3(x_1) = \sigma_2(x_1, x_1)$

General recursive functions

Given a defining system of equations there are two numbers which are noteworthy.

The index of the function being defined σ_s and the number A of axioms.

Example

1. $\sigma_1(0, x_2) = x_2$
2. $\sigma_1(\sigma_0(x_1), x_2) = \sigma_0(\sigma_1(x_1, x_2))$
3. $\sigma_2(0, x_2) = 0$
4. $\sigma_2(\sigma_0(x_1), x_2) = \sigma_1(\sigma_2(x_1, x_2), x_2)$
5. $\sigma_3(x_1) = \sigma_2(x_1, x_1)$

- For this example $s = 3$, $A = 5$ and σ_3 is the square function.

General recursive functions

Example

1. $\sigma_1(0) = 0$
2. $\sigma_1(\sigma_0(x_1)) = x_1$
3. $\sigma_2(0, x_2) = x_2$
4. $\sigma_2(\sigma_0(x_1), x_2) = \sigma_0(\sigma_2(x_1, x_2))$
5. $\sigma_3(0) = 1$
6. $\sigma_3(\sigma_0(x_1)) = \sigma_2(\varphi(x_1), \varphi(\sigma_1(x_1)))$

General recursive functions

Example

1. $\sigma_1(0) = 0$
2. $\sigma_1(\sigma_0(x_1)) = x_1$
3. $\sigma_2(0, x_2) = x_2$
4. $\sigma_2(\sigma_0(x_1), x_2) = \sigma_0(\sigma_2(x_1, x_2))$
5. $\sigma_3(0) = 1$
6. $\sigma_3(\sigma_0(x_1)) = \sigma_2(\varphi(x_1), \varphi(\sigma_1(x_1)))$

- For this example $s = 3$, $A = 6$. Can you identify the function σ_3?

General recursive functions

Example

1. $\sigma_1(0) = 0$
2. $\sigma_1(\sigma_0(x_1)) = x_1$
3. $\sigma_2(x_1, 0) = x_1$
4. $\sigma_2(x_1, \sigma_0(x_2)) = \sigma_1(\sigma_2(x_1, x_2))$

General recursive functions

Example

1. $\sigma_1(0) = 0$
2. $\sigma_1(\sigma_0(x_1)) = x_1$
3. $\sigma_2(x_1, 0) = x_1$
4. $\sigma_2(x_1, \sigma_0(x_2)) = \sigma_1(\sigma_2(x_1, x_2))$

- For this example $s = 2$ and $A = 4$.
- $\sigma_2(x_1, x_2) = x_1 \dot{-} x_2$.

Computations as deductions

$$
\textit{Axioms} \begin{cases} \sigma_1(0, x_2) = x_2 & (1) \\ \sigma_1(\sigma_0(x_1), x_2) = \sigma_0(\sigma_1(x_1, x_2)) & (2) \end{cases}
$$

$$
\textit{Deductions} \begin{cases} \\ \\ \\ \end{cases}
$$

Computations as deductions

$$\text{Axioms} \begin{cases} \sigma_1(0, x_2) = x_2 & (1) \\ \sigma_1(\sigma_0(x_1), x_2) = \sigma_0(\sigma_1(x_1, x_2)) & (2) \end{cases}$$

$$\text{Deductions} \begin{cases} n(1,2) \qquad \sigma_1(0, \sigma_0(x_2)) = \sigma_0(x_2) & (3) \\ \\ \\ \end{cases}$$

Computations as deductions

$$\textit{Axioms} \begin{cases} \sigma_1(0, x_2) = x_2 & (1) \\ \sigma_1(\sigma_0(x_1), x_2) = \sigma_0(\sigma_1(x_1, x_2)) & (2) \end{cases}$$

$$\textit{Deductions} \begin{cases} n(1,2) & \sigma_1(0, \sigma_0(x_2)) = \sigma_0(x_2) & (3) \\ z(3,2) & \sigma_1(0, \sigma_0(0)) = \sigma_0(0) & (4) \\ & & \\ & & \end{cases}$$

Computations as deductions

$$\textit{Axioms} \begin{cases} \sigma_1(0, x_2) = x_2 & (1) \\ \sigma_1(\sigma_0(x_1), x_2) = \sigma_0(\sigma_1(x_1, x_2)) & (2) \end{cases}$$

$$\textit{Deductions} \begin{cases} n(1,2) & \sigma_1(0, \sigma_0(x_2)) = \sigma_0(x_2) & (3) \\ z(3,2) & \sigma_1(0, \sigma_0(0)) = \sigma_0(0) & (4) \\ n(2,2) & \sigma_1(\sigma_0(x_1), \sigma_0(x_2)) = \sigma_0(\sigma_1(x_1, \sigma_0(x_2))) & (5) \\ & & \\ & & \end{cases}$$

Computations as deductions

$$\textit{Axioms} \begin{cases} \sigma_1(0, x_2) = x_2 & (1) \\ \sigma_1(\sigma_0(x_1), x_2) = \sigma_0(\sigma_1(x_1, x_2)) & (2) \end{cases}$$

$$\textit{Deductions} \begin{cases} n(1,2) & \sigma_1(0, \sigma_0(x_2)) = \sigma_0(x_2) & (3) \\ z(3,2) & \sigma_1(0, \sigma_0(0)) = \sigma_0(0) & (4) \\ n(2,2) & \sigma_1(\sigma_0(x_1), \sigma_0(x_2)) = \sigma_0(\sigma_1(x_1, \sigma_0(x_2))) & (5) \\ z(5,i) & \sigma_1(\sigma_0(0), \sigma_0(0)) = \sigma_0(\sigma_1(0, \sigma_0(0))) & (6) \end{cases}$$

Computations as deductions

$$\text{Axioms} \begin{cases} \sigma_1(0, x_2) = x_2 & (1) \\ \sigma_1(\sigma_0(x_1), x_2) = \sigma_0(\sigma_1(x_1, x_2)) & (2) \end{cases}$$

$$\text{Deductions} \begin{cases} n(1,2) & \sigma_1(0, \sigma_0(x_2)) = \sigma_0(x_2) & (3) \\ z(3,2) & \sigma_1(0, \sigma_0(0)) = \sigma_0(0) & (4) \\ n(2,2) & \sigma_1(\sigma_0(x_1), \sigma_0(x_2)) = \sigma_0(\sigma_1(x_1, \sigma_0(x_2))) & (5) \\ z(5,i) & \sigma_1(\sigma_0(0), \sigma_0(0)) = \sigma_0(\sigma_1(0, \sigma_0(0))) & (6) \\ sub(4,6,15) & \sigma_1(\sigma_0(0), \sigma_0(0)) = \sigma_0(\sigma_0(0)) & (7) \end{cases}$$

Computations as deductions

$$
\text{Axioms}\begin{cases}
\sigma_1(0, x_2) = x_2 & (1)\\
\sigma_1(\sigma_0(x_1), x_2) = \sigma_0(\sigma_1(x_1, x_2)) & (2)
\end{cases}
$$

$$
\text{Deductions}\begin{cases}
n(1,2) & \sigma_1(0, \sigma_0(x_2)) = \sigma_0(x_2) & (3)\\
z(3,2) & \sigma_1(0, \sigma_0(0)) = \sigma_0(0) & (4)\\
n(2,2) & \sigma_1(\sigma_0(x_1), \sigma_0(x_2)) = \sigma_0(\sigma_1(x_1, \sigma_0(x_2))) & (5)\\
z(5,i) & \sigma_1(\sigma_0(0), \sigma_0(0)) = \sigma_0(\sigma_1(0, \sigma_0(0))) & (6)\\
sub(4,6,15) & \sigma_1(\sigma_0(0), \sigma_0(0)) = \sigma_0(\sigma_0(0)) & (7)
\end{cases}
$$

This illustrates the three admissable steps in any computation; *n*, *z* and *sub*.

The admissable steps

- $z(i, j)$ denotes plugging zero into the j variable of equation i.

The admissable steps

- $z(i,j)$ denotes plugging zero into the j variable of equation i.
- $n(i,j)$ denotes non-zero evaluation; We are plugging $\sigma_0(x_j)$ into equation i in the variable j.

The admissable steps

- $z(i,j)$ denotes plugging zero into the j variable of equation i.
- $n(i,j)$ denotes non-zero evaluation; We are plugging $\sigma_0(x_j)$ into equation i in the variable j.
- $sub(i,j,k)$ denotes substitution of RHS of equation i into equation j at the kth position.

Computations as deductions

$$
\textit{Axioms}\begin{cases}
\sigma_1(0, x_2) = x_2 & (1)\\
\sigma_1(\sigma_0(x_1), x_2) = \sigma_0(\sigma_1(x_1, x_2)) & (2)\\
\sigma_2(0, x_2) = 0 & (3)\\
\sigma_2(\sigma_0(x_1), x_2) = \sigma_1(\sigma_2(x_1, x_2), x_2) & (4)
\end{cases}
$$

Computations as deductions

$$
Axioms \begin{cases} \sigma_1(0, x_2) = x_2 & (1) \\ \sigma_1(\sigma_0(x_1), x_2) = \sigma_0(\sigma_1(x_1, x_2)) & (2) \\ \sigma_2(0, x_2) = 0 & (3) \\ \sigma_2(\sigma_0(x_1), x_2) = \sigma_1(\sigma_2(x_1, x_2), x_2) & (4) \end{cases}
$$

$$
\begin{cases} n(1,2) \qquad \sigma_1(0, \sigma_0(x_2)) = \sigma_0(x_2) & (5) \\ \\ \\ \\ \end{cases}
$$

Computations as deductions

$$Axioms \begin{cases} \sigma_1(0, x_2) = x_2 & (1) \\ \sigma_1(\sigma_0(x_1), x_2) = \sigma_0(\sigma_1(x_1, x_2)) & (2) \\ \sigma_2(0, x_2) = 0 & (3) \\ \sigma_2(\sigma_0(x_1), x_2) = \sigma_1(\sigma_2(x_1, x_2), x_2) & (4) \end{cases}$$

$$\begin{cases} n(1,2) \quad \sigma_1(0, \sigma_0(x_2)) = \sigma_0(x_2) & (5) \\ z(5,2) \quad \sigma_1(0, \sigma_0(0)) = \sigma_0(0) & (6) \end{cases}$$

Computations as deductions

$$Axioms \begin{cases} \sigma_1(0, x_2) = x_2 & (1) \\ \sigma_1(\sigma_0(x_1), x_2) = \sigma_0(\sigma_1(x_1, x_2)) & (2) \\ \sigma_2(0, x_2) = 0 & (3) \\ \sigma_2(\sigma_0(x_1), x_2) = \sigma_1(\sigma_2(x_1, x_2), x_2) & (4) \end{cases}$$

$$\begin{cases} n(1,2) \quad \sigma_1(0, \sigma_0(x_2)) = \sigma_0(x_2) & (5) \\ z(5,2) \quad \sigma_1(0, \sigma_0(0)) = \sigma_0(0) & (6) \\ n(4,2) \quad \sigma_2(\sigma_0(x_1), \sigma_0(x_2)) = \sigma_1(\sigma_2(x_1, \sigma_0(x_2)), \sigma_0(x_2)) & (7) \end{cases}$$

Computations as deductions

$$Axioms \begin{cases} \sigma_1(0, x_2) = x_2 & (1) \\ \sigma_1(\sigma_0(x_1), x_2) = \sigma_0(\sigma_1(x_1, x_2)) & (2) \\ \sigma_2(0, x_2) = 0 & (3) \\ \sigma_2(\sigma_0(x_1), x_2) = \sigma_1(\sigma_2(x_1, x_2), x_2) & (4) \end{cases}$$

$$\begin{cases} n(1,2) \quad \sigma_1(0, \sigma_0(x_2)) = \sigma_0(x_2) & (5) \\ z(5,2) \quad \sigma_1(0, \sigma_0(0)) = \sigma_0(0) & (6) \\ n(4,2) \quad \sigma_2(\sigma_0(x_1), \sigma_0(x_2)) = \sigma_1(\sigma_2(x_1, \sigma_0(x_2)), \sigma_0(x_2)) & (7) \\ z(7,i) \quad \sigma_2(\sigma_0(0), \sigma_0(0)) = \sigma_1(\sigma_2(0, \sigma_0(0)), \sigma_0(0)) & (8) \end{cases}$$

Computations as deductions

$$
Axioms \begin{cases} \sigma_1(0, x_2) = x_2 & (1) \\ \sigma_1(\sigma_0(x_1), x_2) = \sigma_0(\sigma_1(x_1, x_2)) & (2) \\ \sigma_2(0, x_2) = 0 & (3) \\ \sigma_2(\sigma_0(x_1), x_2) = \sigma_1(\sigma_2(x_1, x_2), x_2) & (4) \end{cases}
$$

$$
\begin{cases} n(1,2) & \sigma_1(0, \sigma_0(x_2)) = \sigma_0(x_2) & (5) \\ z(5,2) & \sigma_1(0, \sigma_0(0)) = \sigma_0(0) & (6) \\ n(4,2) & \sigma_2(\sigma_0(x_1), \sigma_0(x_2)) = \sigma_1(\sigma_2(x_1, \sigma_0(x_2)), \sigma_0(x_2)) & (7) \\ z(7,i) & \sigma_2(\sigma_0(0), \sigma_0(0)) = \sigma_1(\sigma_2(0, \sigma_0(0)), \sigma_0(0)) & (8) \\ n(3,2) & \sigma_2(0, \sigma_0(x_2)) = 0 & (9) \end{cases}
$$

Computations as deductions

$$\textit{Axioms}\left\{\begin{array}{ll} \sigma_1(0, x_2) = x_2 & (1) \\ \sigma_1(\sigma_0(x_1), x_2) = \sigma_0(\sigma_1(x_1, x_2)) & (2) \\ \sigma_2(0, x_2) = 0 & (3) \\ \sigma_2(\sigma_0(x_1), x_2) = \sigma_1(\sigma_2(x_1, x_2), x_2) & (4) \end{array}\right.$$

$$\left\{\begin{array}{lll} n(1,2) & \sigma_1(0, \sigma_0(x_2)) = \sigma_0(x_2) & (5) \\ z(5,2) & \sigma_1(0, \sigma_0(0)) = \sigma_0(0) & (6) \\ n(4,2) & \sigma_2(\sigma_0(x_1), \sigma_0(x_2)) = \sigma_1(\sigma_2(x_1, \sigma_0(x_2)), \sigma_0(x_2)) & (7) \\ z(7,i) & \sigma_2(\sigma_0(0), \sigma_0(0)) = \sigma_1(\sigma_2(0, \sigma_0(0)), \sigma_0(0)) & (8) \\ n(3,2) & \sigma_2(0, \sigma_0(x_2)) = 0 & (9) \\ z(9,2) & \sigma_2(0, \sigma_0(0)) = 0 & (10) \end{array}\right.$$

Computations as deductions

$$Axioms \begin{cases} \sigma_1(0, x_2) = x_2 & (1) \\ \sigma_1(\sigma_0(x_1), x_2) = \sigma_0(\sigma_1(x_1, x_2)) & (2) \\ \sigma_2(0, x_2) = 0 & (3) \\ \sigma_2(\sigma_0(x_1), x_2) = \sigma_1(\sigma_2(x_1, x_2), x_2) & (4) \end{cases}$$

$$\begin{cases} n(1,2) & \sigma_1(0, \sigma_0(x_2)) = \sigma_0(x_2) & (5) \\ z(5,2) & \sigma_1(0, \sigma_0(0)) = \sigma_0(0) & (6) \\ n(4,2) & \sigma_2(\sigma_0(x_1), \sigma_0(x_2)) = \sigma_1(\sigma_2(x_1, \sigma_0(x_2)), \sigma_0(x_2)) & (7) \\ z(7,i) & \sigma_2(\sigma_0(0), \sigma_0(0)) = \sigma_1(\sigma_2(0, \sigma_0(0)), \sigma_0(0)) & (8) \\ n(3,2) & \sigma_2(0, \sigma_0(x_2)) = 0 & (9) \\ z(9,2) & \sigma_2(0, \sigma_0(0)) = 0 & (10) \\ sub(10,8) & \sigma_2(\sigma_0(0), \sigma_0(0))) = \sigma_1(0, \sigma_0(0)) & (11) \end{cases}$$

Computations as deductions

$$
\textit{Axioms}\left\{\begin{array}{lr}
\sigma_1(0, x_2) = x_2 & (1)\\
\sigma_1(\sigma_0(x_1), x_2) = \sigma_0(\sigma_1(x_1, x_2)) & (2)\\
\sigma_2(0, x_2) = 0 & (3)\\
\sigma_2(\sigma_0(x_1), x_2) = \sigma_1(\sigma_2(x_1, x_2), x_2) & (4)
\end{array}\right.
$$

$$
\left\{\begin{array}{llr}
n(1,2) & \sigma_1(0, \sigma_0(x_2)) = \sigma_0(x_2) & (5)\\
z(5,2) & \sigma_1(0, \sigma_0(0)) = \sigma_0(0) & (6)\\
n(4,2) & \sigma_2(\sigma_0(x_1), \sigma_0(x_2)) = \sigma_1(\sigma_2(x_1, \sigma_0(x_2)), \sigma_0(x_2)) & (7)\\
z(7,i) & \sigma_2(\sigma_0(0), \sigma_0(0)) = \sigma_1(\sigma_2(0, \sigma_0(0)), \sigma_0(0)) & (8)\\
n(3,2) & \sigma_2(0, \sigma_0(x_2)) = 0 & (9)\\
z(9,2) & \sigma_2(0, \sigma_0(0)) = 0 & (10)\\
sub(10,8) & \sigma_2(\sigma_0(0), \sigma_0(0))) = \sigma_1(0, \sigma_0(0)) & (11)\\
sub(6,11) & \sigma_2(\sigma_0(0), \sigma_0(0))) = \sigma_0(0) & (12)
\end{array}\right.
$$

Key component of the Kleene normal form

The Kleene operator

Given a function $\tau(a_1, ..., a_r, n)$ in which for every $\vec{a}$ there exists an n such that $\tau(\vec{a}, n) = 0$ we can define a new function $\varphi(\vec{a})$.

Key component of the Kleene normal form

The Kleene operator

Given a function $\tau(a_1, ..., a_r, n)$ in which for every $\vec{a}$ there exists an n such that $\tau(\vec{a}, n) = 0$ we can define a new function $\varphi(\vec{a})$.

$$\varphi(a_1, ..., a_r) = \mu_j[\tau(a_1, ..., a_r) = 0]$$

Key component of the Kleene normal form

The Kleene operator

Given a function $\tau(a_1, ..., a_r, n)$ in which for every $\vec{a}$ there exists an n such that $\tau(\vec{a}, n) = 0$ we can define a new function $\varphi(\vec{a})$.

$$\varphi(a_1, ..., a_r) = \mu_j[\tau(a_1, ..., a_r) = 0]$$

This is the **unbounded** search.

$$\lim_{B \to \infty} \sum_{i=0}^{B} \prod_{j=0}^{i} \tau(\vec{a}, j)^{\mathrm{sg}}$$

Key component of the Kleene normal form

The Kleene operator

Given a function $\tau(a_1, ..., a_r, n)$ in which for every $\vec{a}$ there exists an n such that $\tau(\vec{a}, n) = 0$ we can define a new function $\varphi(\vec{a})$.

$$\varphi(a_1, ..., a_r) = \mu_j[\tau(a_1, ..., a_r) = 0]$$

This is the **unbounded** search.

$$\lim_{B \to \infty} \sum_{i=0}^{B} \prod_{j=0}^{i} \tau(\vec{a}, j)^{\mathrm{sg}}$$

We will use this **Kleene operator** later to search through computations.

The Kleene operator as a general recursive function

Suppose a function $\tau(a_1, ..., a_r, n)$ is already given. We define a new function σ.

- $\sigma(j, a_1, ..., a_r, 0) = j$
- $\sigma(j, a_1, ..., a_r, n+1) = \sigma(j+1, a_1, ..., a_r, \tau(a_1, ..., a_r, j+1))$

The Kleene operator as a general recursive function

Suppose a function $\tau(a_1, ..., a_r, n)$ is already given. We define a new function σ.

- $\sigma(j, a_1, ..., a_r, 0) = j$
- $\sigma(j, a_1, ..., a_r, n+1) = \sigma(j+1, a_1, ..., a_r, \tau(a_1, ..., a_r, j+1))$

Then $\mu_j[\tau(a_1, ..., a_r, j) = 0] = \sigma(0, a_1, ..., a_r, \tau(a_1, ..., a_r, 0))$.

Kleene's theorem states that any general recursive function φ can be put in the form

$$\varphi = \psi(\mu[\tau])$$

where μ is the Kleene operator and ψ and τ are primitive recursive.

Kleene's theorem states that any general recursive function φ can be put in the form

$$\varphi = \psi(\mu[\tau])$$

where μ is the Kleene operator and ψ and τ are primitive recursive.

- This is the normal form we wish to prove exists.

Kleene's theorem states that any general recursive function φ can be put in the form

$$\varphi = \psi(\mu[\tau])$$

where μ is the Kleene operator and ψ and τ are primitive recursive.

- This is the normal form we wish to prove exists.
- This proves the Kleene operator cannot be primitive recursive.

Kleene's theorem states that any general recursive function φ can be put in the form

$$\varphi = \psi(\mu[\tau])$$

where μ is the Kleene operator and ψ and τ are primitive recursive.

- This is the normal form we wish to prove exists.
- This proves the Kleene operator cannot be primitive recursive.
- This also proves that if we augment the class PR with the Kleene operator this yields the class of all computable functions.

Arithmetization of computations

We begin with the basic Godel numbering of symbols.

Individual symbol	**Godel number**
σ_n	$2n+2,\ n=0,1,...$
0	1
$=$	3
(	5
)	7
,	9
x_n	$2n+9,\ n=1,2,...$

Arithmetization of computations

We begin with the basic Godel numbering of symbols.

Individual symbol	**Godel number**
σ_n	$2n+2,\ n=0,1,...$
0	1
$=$	3
(	5
)	7
,	9
x_n	$2n+9,\ n=1,2,...$

By Godel numbering the symbols used in computations we can employ standard arithmetic to sort through sets of computations.

Arithmetization of computations

We begin with the basic Godel numbering of symbols.

Individual symbol	Godel number
σ_n	$2n+2,\ n=0,1,\ldots$
0	1
$=$	3
(	5
)	7
,	9
x_n	$2n+9,\ n=1,2,\ldots$

By Godel numbering the symbols used in computations we can employ standard arithmetic to sort through sets of computations. Furthermore we can hone in on those computations which are actually correct.

More on Godel numbers

Some examples of symbol sequences and their Godel numbers.

Example

1. $\sigma_0(0)$ has Godel number $p_1^2 p_2^5 p_3^1 p_4^7$

More on Godel numbers

Some examples of symbol sequences and their Godel numbers.

Example

1. $\sigma_0(0)$ has Godel number $p_1^2 p_2^5 p_3^1 p_4^7$
2. $\sigma_0(\sigma_0(0))$ has Godel number $p_1^2 p_2^5 p_3^2 p_4^5 p_5^1 p_6^7 p_7^7$

More on Godel numbers

Some examples of symbol sequences and their Godel numbers.

Example

1. $\sigma_0(0)$ has Godel number $p_1^2 p_2^5 p_3^1 p_4^7$
2. $\sigma_0(\sigma_0(0))$ has Godel number $p_1^2 p_2^5 p_3^2 p_4^5 p_5^1 p_6^7 p_7^7$
3. $\sigma_2(x_1) = x_1$ has Godel number $p_1^4 p_2^5 p_3^{11} p_4^7 p_5^3 p_6^{11}$

More on Godel numbers

Some examples of symbol sequences and their Godel numbers.

Example

1. $\sigma_0(0)$ has Godel number $p_1^2 p_2^5 p_3^1 p_4^7$
2. $\sigma_0(\sigma_0(0))$ has Godel number $p_1^2 p_2^5 p_3^2 p_4^5 p_5^1 p_6^7 p_7^7$
3. $\sigma_2(x_1) = x_1$ has Godel number $p_1^4 p_2^5 p_3^{11} p_4^7 p_5^3 p_6^{11}$
4. $\sigma_0(x_i)$ has Godel number $p_1^2 p_2^5 p_3^{2i+9} p_4^7$

Godel numbering a computation

This is the first example computation.

$$Axioms \begin{cases} q_1 := p_1^4 p_2^5 p_3^1 p_4^9 p_5^{13} p_6^7 p_7^3 p_8^{13} \\ \ldots \end{cases}$$

$$Deductions \begin{cases} q_3 := p_1^4 p_2^5 \ldots p_{13}^{13} p_{14}^7 \\ \vdots \\ q_7 := p_1^4 p_2^5 \ldots p_{19}^7 p_{20}^7 \end{cases}$$

The entire computation is encoded in the Godel number $p_1^{q_1} p_2^{q_2} \ldots p_7^{q_7}$.

Basic strategy

Basic strategy

- Every computation is assigned a Godel number $p_1^{e_1} p_2^{e_2} \ldots p_l^{e_l}$.

Basic strategy

- Every computation is assigned a Godel number $p_1^{e_1} p_2^{e_2} \ldots p_l^{e_l}$.
- Each exponent e_i is the Godel number of an equation.

Basic strategy

- Every computation is assigned a Godel number $p_1^{e_1} p_2^{e_2} \dots p_l^{e_l}$.
- Each exponent e_i is the Godel number of an equation.
- The first A equations $e_1,\dots,e_A$ are the axioms.

Basic strategy

- Every computation is assigned a Godel number $p_1^{e_1} p_2^{e_2} \dots p_l^{e_l}$.
- Each exponent e_i is the Godel number of an equation.
- The first A equations e_1,...,e_A are the axioms.
- The rest e_{A+1},...,e_l are expected to be deductions from previous equations.

Basic strategy

- Every computation is assigned a Godel number $p_1^{e_1} p_2^{e_2} \dots p_l^{e_l}$.
- Each exponent e_i is the Godel number of an equation.
- The first A equations $e_1,\ldots,e_A$ are the axioms.
- The rest $e_{A+1},\ldots,e_l$ are expected to be deductions from previous equations.
- We require functions which can detect such deductions.

Basic strategy

- We will construct a primitive recursive function τ_1 such that $\tau_1(\omega, i) = 0$ if and only if $e_i = z(j, k)$ for $j < i$ in the computation ω.

Basic strategy

- We will construct a primitive recursive function τ_1 such that $\tau_1(\omega, i) = 0$ if and only if $e_i = z(j,k)$ for $j < i$ in the computation ω.
-
$$(\exists j \exists k)[\ (j < i)\ \wedge\ (k < \omega)\ \wedge (e_i = z(j,k))\ ]$$

Basic strategy

- We will construct a primitive recursive function τ_1 such that $\tau_1(\omega, i) = 0$ if and only if $e_i = z(j, k)$ for $j < i$ in the computation ω.
- $$(\exists j \exists k)[\ (j < i)\ \wedge\ (k < \omega)\ \wedge (e_i = z(j, k))\ ]$$
- We will construct similar primitive recursive functions τ_2 and τ_3 for the other deduction steps *n* and *sub*.

Basic strategy

- We will construct a primitive recursive function τ_1 such that $\tau_1(\omega, i) = 0$ if and only if $e_i = z(j, k)$ for $j < i$ in the computation ω.
- $$(\exists j \exists k)[\ (j < i)\ \wedge\ (k < \omega)\ \wedge (e_i = z(j, k))\ ]$$
- We will construct similar primitive recursive functions τ_2 and τ_3 for the other deduction steps *n* and *sub*.
- Then $\tau_1(\omega, i)\tau_2(\omega, i)\tau_3(\omega, i)$ will equal zero if and only if the ith equation of ω is some proper deduction.

Admissable deduction step z

Substituting zero for the variable x_i in the equation e altars the Godel number as follows.

Admissable deduction step z

Substituting zero for the variable x_i in the equation e altars the Godel number as follows.

$$z(e,i) = \prod_{j=1}^{\text{long(e)}} p_j^{\left\{ \begin{array}{lll} 1 & \text{if} & \exp_j(e) = 2i + 9 \\ \exp_j(e) & \text{else} & \end{array} \right\}}$$

Admissable deduction step z

Substituting zero for the variable x_i in the equation e altars the Godel number as follows.

$$z(e, i) = \prod_{j=1}^{\text{long(e)}} p_j^{\left\{ \begin{array}{lll} 1 & \text{if} & \exp_j(e) = 2i + 9 \\ \exp_j(e) & \text{else} & \end{array} \right\}}$$

This function is primitive recursive.

Admissable deduction step n

Substituting a non-zero value $\sigma_0(x_i)$ in the variable x_i of equation e altars the Godel number as follows.

Admissable deduction step n

Substituting a non-zero value $\sigma_0(x_i)$ in the variable x_i of equation e altars the Godel number as follows.

$$n(e,i) = \bigotimes_{j=1}^{\mathrm{long(e)}} \left\{ \begin{array}{lll} q & \text{if} & \mathrm{exp_j(e) = 2i + 9} \\ p_1^{\mathrm{exp_j(e)}} & \text{else} & \end{array} \right\}$$

Admissable deduction step n

Substituting a non-zero value $\sigma_0(x_i)$ in the variable x_i of equation e altars the Godel number as follows.

$$n(e, i) = \bigotimes_{j=1}^{\mathrm{long(e)}} \left\{ \begin{array}{lll} q & \text{if} & \mathrm{exp_j(e) = 2i + 9} \\ p_1^{\mathrm{exp_j(e)}} & \text{else} & \end{array} \right\}$$

This function is primitive recursive.

Admissable deduction step n

Substituting a non-zero value $\sigma_0(x_i)$ in the variable x_i of equation e altars the Godel number as follows.

$$n(e,i) = \bigotimes_{j=1}^{\mathrm{long(e)}} \left\{ \begin{array}{lll} q & \text{if} & \mathrm{exp_j(e) = 2i + 9} \\ p_1^{\mathrm{exp_j(e)}} & \text{else} & \end{array} \right\}$$

This function is primitive recursive. What is q?

Useful formulas

If m is the Godel number of an equation then $\mathrm{eq(m)}$ locates the equality sign.

$$\mathrm{eq(m)} = \mu_{\mathrm{j}}[\mathrm{j} \leq \mathrm{long(m)} \ \wedge \ \mathsf{exp}_{\mathrm{j}}(\mathrm{m}) = 3]$$

Bounded search $\Longrightarrow$ PR.

Useful formulas

If m is the Godel number of an equation then $\mathrm{eq}(\mathrm{m})$ locates the equality sign.

$$\mathrm{eq}(\mathrm{m}) = \mu_{\mathrm{j}}[\mathrm{j} \leq \mathrm{long}(\mathrm{m}) \;\wedge\; \mathsf{exp}_{\mathrm{j}}(\mathrm{m}) = 3]$$

Bounded search $\Longrightarrow$ PR.

$$\mathrm{eq}(\mathrm{m}) = 1 + \sum_{\mathrm{i}=1}^{\mathrm{long}(\mathrm{m})} \prod_{\mathrm{j}=1}^{\mathrm{i}} \mathrm{sg}\left(\mathrm{sg}(\mathsf{exp}_{\mathrm{j}}(\mathrm{m}) \dot{-} 3) + \mathrm{sg}(3 \dot{-} \mathsf{exp}_{\mathrm{j}}(\mathrm{m}))\right)$$

Useful formulas

If m is the Godel number of an equation then $\beta(m)$ denotes the Godel number of the left-hand side of the equation.

$$\beta(m) = \prod_{j=1}^{\mathrm{eq(m)} \dot{-} 1} p_j^{\mathsf{exp}_j(m)}$$

Useful formulas

Then the right-hand side of equation m which we denote by $\gamma(m)$ is the following.

$$\gamma(m) = \prod_{j=1}^{\mathrm{long(m)} \dot{-} \mathrm{eq(m)}} p_j^{\mathsf{exp}_{j+\mathrm{eq(m)}}(m)}$$

Does a given equation contain some term?

Here we are testing for a copy of $\beta(m)$ in the expression n beginning at location $j+1$. Denote this predicate $\beta(m,n,j)$.

$$\sum_{i=1}^{\mathrm{long}(\beta(\mathrm{m}))} \mathrm{sg}(\ \mathsf{exp}_{\mathrm{i+j}}(\mathrm{n}) \dot{-} \mathsf{exp}_{\mathrm{i}}\,\beta(\mathrm{m})\ ) + \mathrm{sg}(\ \mathsf{exp}_{\mathrm{i}}\,\beta(\mathrm{m}) \dot{-} \mathsf{exp}_{\mathrm{i+j}}(\mathrm{n})\ )$$

Does a given equation contain some term?

Here we are testing for a copy of $\beta(m)$ in the expression n beginning at location $j+1$. Denote this predicate $\beta(m,n,j)$.

$$\sum_{i=1}^{\mathrm{long}(\beta(\mathrm{m}))} \mathrm{sg}(\ \mathsf{exp}_{\mathrm{i+j}}(\mathrm{n}) \dot{-} \mathsf{exp}_{\mathrm{i}}\,\beta(\mathrm{m})\ ) + \mathrm{sg}(\ \mathsf{exp}_{\mathrm{i}}\,\beta(\mathrm{m}) \dot{-} \mathsf{exp}_{\mathrm{i+j}}(\mathrm{n})\ )$$

- Hence $\beta(m,n,j) = 0$ iff $\mathsf{exp}_i(\beta(m)) = \mathsf{exp}_{i+j}(n)$ for every $1 \leq i \leq \mathrm{long}(\beta(\mathrm{m}))$.

Does a given equation contain some term?

Here we are testing for a copy of $\beta(m)$ in the expression n beginning at location $j+1$. Denote this predicate $\beta(m,n,j)$.

$$\sum_{i=1}^{\mathrm{long}(\beta(\mathrm{m}))} \mathrm{sg}(\ \mathsf{exp}_{\mathrm{i+j}}(\mathrm{n}) \dot{-} \mathsf{exp}_{\mathrm{i}}\,\beta(\mathrm{m})\ ) + \mathrm{sg}(\ \mathsf{exp}_{\mathrm{i}}\,\beta(\mathrm{m}) \dot{-} \mathsf{exp}_{\mathrm{i+j}}(\mathrm{n})\ )$$

- Hence $\beta(m,n,j) = 0$ iff $\mathsf{exp}_i(\beta(m)) = \mathsf{exp}_{i+j}(n)$ for every $1 \leq i \leq \mathrm{long}(\beta(\mathrm{m}))$.
- So $\beta(m,n,j) = 0$ iff the expression for n contains the expression for $\beta(m)$ precisely after the jth symbol.

Does a given equation contain some term?

Here we are testing for a copy of $\beta(m)$ in the expression n beginning at location $j+1$. Denote this predicate $\beta(m,n,j)$.

$$\sum_{i=1}^{\mathrm{long}(\beta(\mathrm{m}))} \mathrm{sg}(\ \mathsf{exp}_{\mathrm{i+j}}(\mathrm{n}) \dot{-} \mathsf{exp}_{\mathrm{i}}\,\beta(\mathrm{m})\ ) + \mathrm{sg}(\ \mathsf{exp}_{\mathrm{i}}\,\beta(\mathrm{m}) \dot{-} \mathsf{exp}_{\mathrm{i+j}}(\mathrm{n})\ )$$

- Hence $\beta(m,n,j) = 0$ iff $\mathsf{exp}_i(\beta(m)) = \mathsf{exp}_{i+j}(n)$ for every $1 \le i \le \mathrm{long}(\beta(\mathrm{m}))$.
- So $\beta(m,n,j) = 0$ iff the expression for n contains the expression for $\beta(m)$ precisely after the jth symbol.
- Note that $\beta(m,n,j) \neq 0$ if $\mathrm{long}(\mathrm{n}) - \mathrm{j} < \mathrm{long}(\beta(\mathrm{m}))$.

The binary function $\mathrm{eq}(\mathrm{m}, \mathrm{n})$

We will use eq also to denote the binary function given below.

$$\mathrm{eq}(\mathrm{m}, \mathrm{n}) = \mathrm{sg}(\mathrm{m} \dot{-} \mathrm{n}) + \mathrm{sg}(\mathrm{n} \dot{-} \mathrm{m})$$

The binary function $\mathrm{eq}(\mathrm{m}, \mathrm{n})$

We will use eq also to denote the binary function given below.

$$\mathrm{eq}(\mathrm{m}, \mathrm{n}) = \mathrm{sg}(\mathrm{m} \dot{-} \mathrm{n}) + \mathrm{sg}(\mathrm{n} \dot{-} \mathrm{m})$$

Hence $\mathrm{eq}(\mathrm{m}, \mathrm{n}) = 0$ if and only if $m = n$.

The binary function $\mathrm{eq}(\mathrm{m}, \mathrm{n})$

We will use eq also to denote the binary function given below.

$$\mathrm{eq}(\mathrm{m}, \mathrm{n}) = \mathrm{sg}(\mathrm{m} \dot{-} \mathrm{n}) + \mathrm{sg}(\mathrm{n} \dot{-} \mathrm{m})$$

Hence $\mathrm{eq}(\mathrm{m}, \mathrm{n}) = 0$ if and only if $m = n$. Then

$$\sum_{i=1}^{\mathrm{long}(\beta(\mathrm{m}))} \mathrm{sg}(\ \mathsf{exp}_{\mathrm{i+j}}(\mathrm{n}) \dot{-} \mathsf{exp}_{\mathrm{i}}\, \beta(\mathrm{m})\ ) + \mathrm{sg}(\ \mathsf{exp}_{\mathrm{i}}\, \beta(\mathrm{m}) \dot{-} \mathsf{exp}_{\mathrm{i+j}}(\mathrm{n})\ )$$

The binary function $\mathrm{eq}(\mathrm{m}, \mathrm{n})$

We will use eq also to denote the binary function given below.

$$\mathrm{eq}(\mathrm{m}, \mathrm{n}) = \mathrm{sg}(\mathrm{m} \dot{-} \mathrm{n}) + \mathrm{sg}(\mathrm{n} \dot{-} \mathrm{m})$$

Hence $\mathrm{eq}(\mathrm{m}, \mathrm{n}) = 0$ if and only if $m = n$. Then

$$\sum_{i=1}^{\mathrm{long}(\beta(\mathrm{m}))} \mathrm{sg}(\ \mathsf{exp}_{\mathrm{i+j}}(\mathrm{n}) \dot{-} \mathsf{exp}_{\mathrm{i}}\, \beta(\mathrm{m})\ ) + \mathrm{sg}(\ \mathsf{exp}_{\mathrm{i}}\, \beta(\mathrm{m}) \dot{-} \mathsf{exp}_{\mathrm{i+j}}(\mathrm{n})\ )$$

simplifies to

$$\beta(m, n, j) = \sum_{i=1}^{\mathrm{long}(\beta(\mathrm{m}))} \mathrm{eq}(\mathsf{exp}_{\mathrm{i+j}}(\mathrm{n}), \mathsf{exp}_{\mathrm{i}}\, \beta(\mathrm{m})).$$

Admissable deduction step *sub*

Substituting $\gamma(m)$ for $\beta(m)$ in equation n altars it's Godel number as follows.

If $\beta(m, n, j) = 0$ then

$$sub(m, n, j) = \prod_{i=1}^{j} p_i^{\mathrm{exp}_i(n)} * \gamma(m) * \prod_{i=1}^{\mathrm{long(n)} \dot{-} (\mathrm{j} + \mathrm{long}\beta(\mathrm{m}))} p_i^{\mathrm{exp}_{j+\mathrm{long}\beta(\mathrm{m})+\mathrm{i}}(n)}$$

otherwise $sub(m, n, j) = n$.

Admissable deduction step *sub*

Substituting $\gamma(m)$ for $\beta(m)$ in equation n altars it's Godel number as follows.

If $\beta(m,n,j) = 0$ then

$$sub(m,n,j) = \prod_{i=1}^{j} p_i^{\exp_i(n)} * \gamma(m) * \prod_{i=1}^{\mathrm{long(n)} \dot{-} (\mathrm{j} + \mathrm{long}\beta(\mathrm{m}))} p_i^{\exp_{j+\mathrm{long}\beta(\mathrm{m})+\mathrm{i}}(n)}$$

otherwise $sub(m,n,j) = n$.

This is a primitive recursive function.

Valid deduction

The natural number ω is the Godel number of a valid deduction if for every $i \leq \mathrm{long}(\omega)$ one of the following is true;

- $\exp_i(\omega)$ is the Godel number of one of the defining equations.

Valid deduction

The natural number ω is the Godel number of a valid deduction if for every $i \leq \mathrm{long}(\omega)$ one of the following is true;

- $\mathsf{exp}_i(\omega)$ is the Godel number of one of the defining equations.
- There exists $j < i$ and some k such that $\mathsf{exp}_i(\omega) = z(\mathsf{exp}_j(\omega), k)$.

Valid deduction

The natural number ω is the Godel number of a valid deduction if for every $i \leq \mathrm{long}(\omega)$ one of the following is true;

- $\exp_i(\omega)$ is the Godel number of one of the defining equations.
- There exists $j < i$ and some k such that $\exp_i(\omega) = z(\exp_j(\omega), k)$.
- There exists $j < i$ and some k such that $\exp_i(\omega) = n(\exp_j(\omega), k))$.

Valid deduction

The natural number ω is the Godel number of a valid deduction if for every $i \leq \mathrm{long}(\omega)$ one of the following is true;

- $\mathrm{exp}_i(\omega)$ is the Godel number of one of the defining equations.
- There exists $j < i$ and some k such that $\mathrm{exp}_i(\omega) = z(\mathrm{exp}_j(\omega), k)$.
- There exists $j < i$ and some k such that $\mathrm{exp}_i(\omega) = n(\mathrm{exp}_j(\omega), k))$.
- $\exists m, n < i$ and j where $\mathrm{exp}_i(\omega) = sub(\mathrm{exp}_m(\omega), \mathrm{exp}_n(\omega), j))$.

Valid deduction

Let $\tau_1(\omega, i)$ denote the predicate

$$(\exists j \exists k)[\ (j < i)\ \wedge\ (k \leq \omega)\ \wedge\ \mathrm{eq}(\mathsf{exp}_{\mathrm{i}}\, \omega, \mathrm{z}(\mathsf{exp}_{\mathrm{j}}\, \omega, \mathrm{k}))\ ]$$

Valid deduction

Let $\tau_1(\omega, i)$ denote the predicate

$$(\exists j \exists k)[\ (j < i)\ \wedge\ (k \leq \omega)\ \wedge\ \mathrm{eq}(\mathsf{exp}_{\mathrm{i}}\, \omega, \mathrm{z}(\mathsf{exp}_{\mathrm{j}}\, \omega, \mathrm{k}))\ ]$$

- Thus $\tau_1(\omega, i)$ is primitive recursive and, as formulated below, equals zero iff the property holds true.

$$\tau_1(\omega, i) = \prod_{j=1}^{i \dot{-} 1} \prod_{k=1}^{\omega} \mathrm{eq}(\ \mathsf{exp}_{\mathrm{i}}\, \omega, \mathrm{z}(\mathsf{exp}_{\mathrm{j}}\, \omega, \mathrm{k})\ )$$

Valid deduction

Let $\tau_2(\omega, i)$ denote the predicate

$$(\exists j \exists k)[\ (j < i)\ \wedge\ (k \leq \omega)\ \wedge\ \mathrm{eq}(\mathsf{exp}_{\mathrm{i}}\,\omega, \mathrm{n}(\mathsf{exp}_{\mathrm{j}}\,\omega, \mathrm{k}))\ ]$$

Valid deduction

Let $\tau_2(\omega, i)$ denote the predicate

$$(\exists j \exists k)[\ (j < i)\ \wedge\ (k \leq \omega)\ \wedge\ \mathrm{eq}(\mathsf{exp}_{\mathrm{i}}\,\omega, \mathrm{n}(\mathsf{exp}_{\mathrm{j}}\,\omega, \mathrm{k}))\ ]$$

- Thus $\tau_2(\omega, i)$ is primitive recursive and, as formulated below, equals zero iff the property holds true.

$$\tau_2(\omega, i) = \prod_{j=1}^{i \dot{-} 1} \prod_{k=1}^{\omega} \mathrm{eq}(\ \mathsf{exp}_{\mathrm{i}}\,\omega, \mathrm{n}(\mathsf{exp}_{\mathrm{j}}\,\omega, \mathrm{k})\ )$$

Valid deduction

Let $\tau_3(\omega, i)$ denote the predicate

$$(\exists m)(\exists n)(\exists j)[\,(m, n < i) \wedge (j \leq \mathrm{long(n)}) \wedge \mathsf{exp}_{\mathrm{i}}\, \omega = \mathrm{sub}(\mathsf{exp}_{\mathrm{m}}\, \omega, \mathsf{exp}_{\mathrm{n}}\, \omega, \mathrm{j})\,]$$

Valid deduction

Let $\tau_3(\omega, i)$ denote the predicate

$$(\exists m)(\exists n)(\exists j)[\,(m, n < i) \wedge (j \leq \mathrm{long(n)}) \wedge \mathsf{exp}_\mathrm{i}\,\omega = \mathrm{sub}(\mathsf{exp}_\mathrm{m}\,\omega, \mathsf{exp}_\mathrm{n}\,\omega, \mathrm{j})\,]$$

- Thus $\tau_3(\omega, i)$ is primitive recursive and, as formulated below, equals zero iff the property holds true.

$$\tau_3(\omega, i) = \prod_{m=1}^{i \dot{-} 1} \prod_{n=1}^{i \dot{-} 1} \prod_{j=1}^{\mathrm{long(n)}} \mathrm{eq}(\mathsf{exp}_\mathrm{i}\,\omega, \mathrm{sub}(\mathsf{exp}_\mathrm{m}\,\omega, \mathsf{exp}_\mathrm{n}\,\omega, \mathrm{j}))$$

Useful formula

The symbol sequence $\sigma_0(0)$ represents the natural number one and in general if $\sigma_0^n(0)$ then $\sigma_0(\sigma_0^n(0))$ represents $n + 1$.

Useful formula

The symbol sequence $\sigma_0(0)$ represents the natural number one and in general if $\sigma_0^n(0)$ then $\sigma_0(\sigma_0^n(0))$ represents $n + 1$.

The Godel numbers of these expressions have a primitive recursive formula.

Useful formula

The symbol sequence $\sigma_0(0)$ represents the natural number one and in general if $\sigma_0^n(0)$ then $\sigma_0(\sigma_0^n(0))$ represents $n+1$.

The Godel numbers of these expressions have a primitive recursive formula.

$$\zeta(0) = p_1^1 \text{ and } \zeta(n+1) = p_1^2 p_2^5 * \zeta(n) * p_1^7$$

The valid deduction of $\varphi(a_1, ..., a_r)$

Let $\tau_4(a_1, ..., a_r, \omega)$ denote the predicate

$$(\exists a)[\ (a < \gamma(\mathsf{exp}_{\mathrm{long}\omega}(\omega))\ \wedge\ \mathrm{eq}(\mathsf{exp}_{\mathrm{long}\omega}(\omega), \Phi(\mathrm{a}))\ ]$$

The valid deduction of $\varphi(a_1, ..., a_r)$

Let $\tau_4(a_1, ..., a_r, \omega)$ denote the predicate

$$(\exists a)[\ (a < \gamma(\mathsf{exp}_{\mathrm{long}\omega}(\omega))\ \wedge\ \mathrm{eq}(\mathsf{exp}_{\mathrm{long}\omega}(\omega), \Phi(\mathrm{a}))\ ]$$

- Where $\Phi(a) := p_1^{2s+9} p_2^5 * \zeta(a_1) * \ldots * \zeta(a_r) * p_1^7 p_2^3 * \zeta(a))$

$$\tau_4(a_1, ..., a_r, \omega) = \prod_{a=0}^{\gamma(\mathsf{exp}_{\mathrm{long}\omega}(\omega))} \mathrm{eq}(\mathsf{exp}_{\mathrm{long}\omega}(\omega), \Phi(\mathrm{a}))$$

PR detection of a valid computation

Let $\tau(a_1, ..., a_r, \omega)$ denote the following formula

$$\sum_{i=1+A}^{\mathrm{long}(\omega)} \tau_1(\omega, i)\tau_2(\omega, i)\tau_3(\omega, i) + \tau_4(a_1, ..., a_r, \omega)$$

PR detection of a valid computation

Let $\tau(a_1, ..., a_r, \omega)$ denote the following formula

$$\sum_{i=1+A}^{\mathrm{long}(\omega)} \tau_1(\omega, i)\tau_2(\omega, i)\tau_3(\omega, i) + \tau_4(a_1, ..., a_r, \omega)$$

- Thus ω is the Godel number of a valid computation of $\varphi(\vec{a})$ iff $\tau(a_1, ..., a_r, \omega) = 0$.

PR detection of a valid computation

Let $\tau(a_1, ..., a_r, \omega)$ denote the following formula

$$\sum_{i=1+A}^{\text{long}(\omega)} \tau_1(\omega, i)\tau_2(\omega, i)\tau_3(\omega, i) + \tau_4(a_1, ..., a_r, \omega)$$

- Thus ω is the Godel number of a valid computation of $\varphi(\vec{a})$ iff $\tau(a_1, ..., a_r, \omega) = 0$.
- The function $\mu_\omega[\tau(a_1, ..., a_r, \omega) = 0]$ denotes the unbounded search for a valid computation of $\varphi(\vec{a})$.

Kleene normal form

Let $\psi(\omega)$ denote the function

$$\mu_j[j < \gamma(\mathsf{exp}_{\mathrm{long}\omega}(\omega)) \ \wedge \ \zeta(j) = \gamma(\mathsf{exp}_{\mathrm{long}(\omega)}(\omega)) \ ]$$

Since $j < \gamma(\mathsf{exp}_{\mathrm{long}\omega}(\omega))$ the function ψ is primitive recursive and finally we have the normal form below.

$$\varphi = \psi(\mu_\omega[\tau(a_1, ..., a_r, \omega) = 0])$$

Thus every GR function can be written as above where ψ is a PR function and τ is a PR predicate on which the Kleene operator acts.