Consistency proof for Peano arithmetics

Slaviša B. Prešić

Abstract. *In this paper we sintactically prove that Peano arithmetic is consistent. The proof uses so called* **sint** *algorithm.*

1. Sint-algorithm in theory Num

1. Let $L = \{0, /, +, \cdot, =\}$ be a first-order language, where $0, /, +, \cdot$ are operational symbols[1)] with $|0| = 0$ (i.e. 0 is a symbol of a constant) $|/| = 1$, $|+| = 2$, $|\cdot| = 2$; $=$ is a binary relational symbol "equality symbol". By Var is denoted the set of variables $x_1, x_2, ..., x_n, ...$. Let $\mathcal{A}$ be the alphabet consisting of the following "letters":

$$), \quad (, \quad 0, \quad /, \quad +, \quad \cdot \qquad \text{and the variables}$$

By $W(\mathcal{A})$ we denote the set of all words of the alphabet $\mathcal{A}$, including the empty word. The symbol $\equiv$ is used for the word equality relation. So, $w_1 \equiv w_2$ means that the word w_1 is equal to the word w_2.

Now we define the notion of a numeral. We shall use the following rule

$$(Step) \qquad \frac{w}{w/}$$

which is an abbrevation of the following sentence: if w is a numeral, then $w/$ is a numeral too.

By use of this rule we have the following definition for a numeral:

Definition 1. 1 . First we adopt that the ward 0 is a numeral and second we adopt the rule $(Step)$.

Now we introduce the following basic sequence

$$(Sequence\ of\ numerals) \qquad 0, \quad 0/, \quad 0//, \cdots$$

---

1)We point out that instead the symbol ′ we shall mainly use the symbol /

We point out that in this definition we use some intuition of infinite sequence.

Let n be any numeral of the form $w/$ with some numeral w, then w is the *predecessor* of the numeral n, briefly written $pred(n)$. For instance, $0//$ is the predecessor of $0///$.

Now we define *induction proof* of some formula $(\forall x)\Phi(x)$:

$(Induction\ proof)$ To prove $(\forall x)\Phi(x)$ inductive *means:* first to prove $\Phi(0)$ and, second to prove the following a sequent $\Phi(W) \vdash \Phi(W/)$.

Now we have the following definition related to the quantifier $\exists$:

$(Proof\ of\ (\exists x)\Phi(x))$ *means* to prove $\Phi(c)$, where c is a member of $(Sequence\ of\ numerals)$

Certain elements of the set $W(\mathcal{A})$ are called L-terms. They are introduced by the following inductive definition.

Definition 1.2.

(i) Symbol 0 *is an L-term.*

(ii) A variable is an L-term.

(iii) If u, v *are L-terms then the words* $u/$, $(u+v), (u \cdot v)$ *are L-terms.*

A *ground L-term* is an L-term not containing variables.

In a usual way one can prove the following

Lemma 1.1. (Uniqueness of L-term construction) *If* A, B, C, D *are any L-terms and* $*, \Delta$ *are elements of the set* $\{+, \cdot\}$ *then we have the following equivalences*

$$A/ \equiv B/ \quad \textit{iff} \quad A \equiv B$$
$$(A * B) \equiv (C \Delta D) \quad \textit{iff} \quad * \equiv \Delta,\ A \equiv C,\ B \equiv D$$

We emphasize some crucial points about the meta-theory, we are going to use. Namely, dealing with some terms, formulas we shall not use the principle of mathematical induction on the length of a term, a formula, etc. Related to this question, suppose that we have some sentence depending on a numeral n, briefly denoted by $\varphi(n)$, and suppose that we want to prove $\varphi(n)$ for every numeral n. Then, instead of using the induction on the length of the word n, according to Definition 1.1, we can perform in the following way

(1.1) *First, to prove* $\varphi(0)$
Second, to prove the sequent $\varphi(w) \vdash \varphi(w/)$, *where* w *is a new constant symbol representing numerals. In other words, supposing* $\varphi(w)$ *to prove* $\varphi(w/)$, *where* w *is an inspecified numeral.*

Such a proof we shall call *a proof by induction on numeral* n. In the sequent $\varphi(w) \vdash \varphi(w/)$ the part $\varphi(w)$ will be called *the induction hypothesis.*

Remark 1.1 *In general, if we have to prove some sentence* $\varphi(t)$ *where* t *is a term, formula or theorem (in some theory) we can in a similar way use a proof by induction on term, formula, teorem. For instance, in such a way one can prove Lemma* 1.1.

Let now Alg be any algorithm dealing with some words, for instance with ground terms. Such an algorithm Alg dealing with any ground term t reads (n is an auxiliary variable) :

(1.2) *We put* $n :\equiv 0$. *Going over* t, *letter by letter from the left to the right, whenever we 'meet' the letter* $+$ *we put* $n :\equiv n/$ *until we reach the end of the word* t.

For instance, if t is term $(0// + (0/ + (0 \cdot 0//))$ then $n(t)$ is $0//$. We can say that $n(t)$ 'sintactically counts' the letters $+$ occurring in t. This $n(t)$ is an example of the so called *numeral counter.* In general, such a numeral counter will be some determined numeral associated with a given ground term t.

Let now Alg be the following algorithm dealing with a given numeral num (n is an auxiliary variable)

(1.4) We put $n :\equiv num$.

(i) If $n \equiv 0$ the Alg stops, otherwise we put $n :\equiv pred(n)$ and go to (i).

In order to see that this Alg will stop at some step we introduce the following sentence $\varphi(num)$:

Algorithm Alg applied to num will stop at some step

Then using the criterion (1.1) one can easily prove that Alg applied to arbitrary numeral num will stop at some step.

Suppose now that certain words are taken as *stopping words*, and that Alg is *a genuine word-algorithm* in the following sense:

Alg applied to any word w produces a new word, denoted by $Alg < w >$, to which Alg should be further applied, and additionaly Alg stops whenever it reaches a stopping word.

2. Now we shall define an equational theory Num,

which will be *restricted* in the following sense: the variables in its axioms are restricted to the ground L-terms. Theory Num is defined by the following axioms

$$(1.5) \qquad \begin{array}{ll} (X+0) = X, & (X+Y/) = (X+Y)/ \\ (X \cdot 0) = 0, & (X \cdot Y/) = ((X \cdot Y) + X) \end{array}$$

where X, Y may be any ground L-terms. The general equality axiom $X = X$ and the corresponding rules (symmetry, tranzitivity, ...) are supposed.

Related to (1.5) we introduce the following word substitutions

$$(1.6) \qquad \begin{array}{ll} \text{(i)} \ (X+0) \longrightarrow X & (ii) \ (X+Y/) \longrightarrow (X+Y)/ \\ \text{(iii)} \ (X \cdot 0) \longrightarrow 0, & (iv) \ (X \cdot Y/) \longrightarrow ((X \cdot Y) + X) \end{array}$$
$$(X, Y \text{ are any ground } L\text{-terms})$$

If $A \longrightarrow B$ is any of such substitutions then B is called **the successor** of A, briefly written $succ(A)$. Now, we are going to define the substantial notion of Num, the so called [2] $sint$. This $sint$ is an algorithm by which for any given ground $L-$term t one can construct exactly one numeral $sint(t)$, such that the equality $t = sint(t)$ is a theorem of Num.

First, we define the notion of *a simple term* and the notion of *the first subterm* of a given ground L-term t. *A simple term* is a ground L-term

[2] An abbreviation of "sintactical value".

of the form $(P * Q)$, where $*$ is $+$ or $\cdot$ and P, Q are some numerals. For instance, $(0// + 0/)$, $(0/// \cdot 0//)$ are simple terms.

Let t be a ground L-term, not a numeral. *The first subterm* of t is a simple term $(P * Q)$ with the following properties

1^0 $(P * Q)$ is a subterm of t

2^0 In the word t, going from the beginning to the right, the word $(P * Q)$ is the first simple term being a subterm of [3] t.

For instance, the term $(0// + 0///)$ is the first subterm of the following term $((0 + (0 \cdot (0// + 0///))) + 0/)$.

Let now t be a ground L-term for which A is the first subterm. Denote t by $t[A]$. Then a substitution of the form

$$t[A] \longrightarrow t[succ(A)]$$

will be called *a step of the sint-algorithm.* Now we define the sint-algorithm, for which numerals are stopping words. If it stops, the final word, i.e. some numeral will be its *result*, briefly denoted by $sint[t]$:

(1.7) (j) *If t is a numeral, sint-algorithm stops and its result $sint[t]$ is t.*
(jj) *If t is not a numeral then we find the first subterm A of t and perform the step*

$$(\sigma) \qquad t[A] \longrightarrow t[succ(A)]$$

Next, we put $t :\equiv t[succ(A)]$ and go to (j)

Concerning the definition (1.7) the main problem is how to prove that sint-algorithm must stop at some step. First we give an example. The sint-algorithm applied to term $(0// + 0///)$ has the following steps

$$
\begin{array}{llll}
(*1) & (0// + 0///) & \longrightarrow (0// + 0//)/ & \text{Applying (1.6)(ii)} \\
 & & \longrightarrow (0// + 0/)// & \text{Applying (1.6)(ii)} \\
 & & \longrightarrow (0// + 0)/// & \text{Applying (1.6)(ii)}
\end{array}
$$

[3] That means that the symbol), the final letter of $(P * Q)$, is the first such symbol ossuring in t.

$\longrightarrow 0/////$ Applying (1.6)(i)

Thus, the result is $0/////$. Next, we point the following facts

(1.8) *(i) Let $(P+Q/)$ be the first subterm of the term t. Replacing $(P+Q/)$ by $(P+Q)/$ from t we obtain a new term whose first subterm is $(P+Q)$.*

(ii) Let $(P \cdot Q/)$ be the first subterm of the term t. Replacing $(P \cdot Q/)$ by $((P \cdot Q)+P)$ from t we obtain a new term whose first subterm is $(P \cdot Q)$.

Proof. (i) Term t can be viewed in the following form

$$L(P+Q/)R$$

where L, R are some words. Replacing $(P+Q/)$ by $(P+Q)/$ we obtain the following term

$$L(P+Q)/R$$

Let f be its first subterm. To prove (i) we shall prove that f is not a subword of L. Indeed, in the opposite case this f would be the first subterm of t, what contradicts with the assumption that $(P+Q/)$ is the first subterm of t. The part (ii) can be proved in a similar way.

The next lemma concerning sint-algorithm is an immediate consequence of fact (1.8):

Lemma 1.2. *Let a ground term t, with the first subterm $(P*Q)$, has a subterm T which also contain the subterm $(P*Q)$ [4]. Term t has the form*

$$LTR$$

where L, R are some words. Replacing $(P*Q)$ by its successor from terms T, t we obtain new terms T', t'. Term t' has the form

$$LT'R$$

The words L, R remain unchanged. If T' is not a numeral, then succ$((P*Q))$ is the first subterm both of t' and T'.

Now in virtue of Lemma 1.2 we have the following

Lemma 1.3 *Let a ground term t has a subterm T, so that t and T have the same first subterm. Suppose that sint-algorithm applied to T stops at some*

[4] T may be equal to $(P*Q)$

step with the result m, where m is a certain numeral. Then sint-algorithm applied to $t[T]$ reduses to sint-algorithm applied to $t[m]$.

Proof. Consider the following algorithm applied to the word $t[T]$ (in the algorithm x is an auxiliary variable)

We set $x :\equiv T$

While (x is not m) do

Begin The first subterm of x replace by its successor. From x we obtain say x'. Set $x :\equiv x'$ End

According to Lemma 1.2 this algorithm is an initial part of the sint-algorithm applied to term $t[T]$. The final result is $t[m]$. The proof is complete.

The above example (*1) illustrates how sint-algorithm works on a simple term of the form $(P + Q)$. In general, according to (1.8), part (i), sint-algorithm applied to a given simple term $(P + Q)$ can be viewed as the following algorithm, in which w, x are auxiliary variables

We set $w :\equiv (P + Q)$, $x :\equiv Q$

While (x is not 0) do

Begin Find the first subterm of w and apply (1.6) (ii).

Set : w is the obtained word, $x :\equiv pred(x)$ End;

In word w replace subword $(P + 0)$ by P.

The result will be the final value of the variable w. In fact, this algorithm is a proof of following lemma:

Lemma 1.4. *The sint-algorithm applied to a simple term of the form $(P + Q)$ will stop at some step.*

Next, combining Lemma 1.3 and Lemma 1.4 we have the following

Lemma 1.5. *Let t be any ground L-term whose the first subterm is $(P + Q)$. Suppose that $sint((P + Q)) \equiv m$, where m is a numeral. Then sint-algorithm applied to t reduces to sint-algorithm applied to $t[m]$.*

Example 1.1. Find $sint(((0// + (0/// + 0/)) + (0/ + 0////)))$.

Using Lemma 1.5 we have the following abridged steps

$((0// + (0/// + 0/)) + (0/ + 0////))$

$\longrightarrow$ $((0// + 0////) + (0/ + 0////))$ Since $(0/// + 0/)$ is the first subterm, and $sint((0/// + 0/))$ is $0////$

$\longrightarrow$ $(0////// + (0/ + 0////))$ Since $(0// + 0////)$ is the first subterm and $sint[(0// + 0////)]$ is $0//////$

$\longrightarrow$ $(0////// + 0/////)$ Since $(0/ + 0////)$ is the first subterm and $sint((0/ + 0////))$ is $0/////$

$\longrightarrow$ $0///////////$ Since $(0////// + 0/////)$ is the first subterm and $sint((0////// + 0/////))$ is $0///////////$

A ground term t is called *a plus-term* if it does not contain the symbol $\cdot$.

Lemma 1.6. *The sint-algorithm applied to a plus-term t will stop at some step.*

Proof. Let $numc(t)$ be the numeral defined by the algorithm (1.2). The proof is by induction on $numc(t)$.

If $numc(t)$ is 0, then t is a numeral and sint-algorithm stops.

If $numc(t)$ is not 0, suppose that $(P+Q)$ is the first subterm of t. According to Lemma 1.4 let a numeral m be $sint[(P+Q)]$. Replacing $(P+Q)$ by m from t we obtain a new term t'. In virtue Lemma 1.5 sint-algorithm reduces to the sint-algorithm applied to t', whose $numc$ is the predecessor of $numc(t)$. The proof is complete.

The next problem is how to find $sint[t]$ when term t contains the symbol $\cdot$. First, we give an example.

Example 1.2. Find $sint[t]$, where t is the term $(0// \cdot 0///)$.

The main idea is the following one : by sint-algorithm from term t to go to a plus-term t', and further to use Lemma 1.6. We have the following steps

$(0// \cdot 0///)$ Now we shall apply (1.6) (iv)

$\longrightarrow$ $((0// \cdot 0//) + 0//)$ Now the first subterm is $(0// \cdot 0//)$. Applying (1.6) (iv) we obtain

$\longrightarrow (((0// \cdot 0/) + 0//) + 0//)$ Applying (1.6) (iv) to the first subterm $(0// \cdot 0/)$ we obtain

$\longrightarrow ((((0// \cdot 0) + 0//) + 0//) + 0//)$ Applying (1.6) (iii) to the first subterm $(0// \cdot 0)$ we obtain

$\longrightarrow (((0 + 0//) + 0//) + 0//)$

Thus, we have obtained a plus-term, such that we can do likewise as in Example 1.1. The result is $0//////$.

In fact, this example illustrates how sint-algorithm applied to a simple term of the form $(P \cdot Q)$ produces a plus-term. For this goal we used only substitutions (1.6) (iv) and (1.6) (iii). Next, we have the following

Lemma 1.7. *The sint-algorithm applied to a simple term of the form $(P \cdot Q)$ can be reduced to sint-algorithm applied to a plus-term.*

Proof. We apply to the simple term $(P \cdot Q)$ the following algorithm, in which w, x are auxiliary variables

We set $w :\equiv (P \cdot Q)$, $x :\equiv Q$

While (x is not 0) do

Begin Find the first subterm of w and apply (1.6) (iv).

Set : w is the obtained word, $x :\equiv pred(x)$ End;

In the word w replace subword $(P \cdot a)$ by 0.

This algorithm is an initial part of the sint-algorithm applied to term $(P \cdot Q)$. Its result is the final value of the variable w, which contain none symbol $\cdot$. In other words w is plus-term. So, sint-algorithm applied to the simple term $(P \cdot Q)$ reduces to applying sint-algorithm to a determined plus-term. The proof is complete.

Lemma 1.8. *The sint-algorithm applied to a simple term of the form $(P \cdot Q)$ will stop at some step.*

Proof follows immediately by Lemma 1.7 and Lemma 1.6.

Next, combining Lemma 1.3 and Lemma 1.8 we have the following lemma

Lemma 1.9. *Let t be any ground L-term whose the first subterm is $(P \cdot Q)$. Suppose that $sint((P \cdot Q)) \equiv m$, where m is a numeral. Then sint-algorithm applied to t reduces to sint-algorithm applied to $t[m]$.*

Now we can prove the following theorem

Theorem 1.1. *Let t be any ground term. Then sint-algorithm applied to t will stop at some step.*

Proof. Let $numc(t)$ be the numeral n defined by the following algorithm (similar to (1.2)) :

> We put $n :\equiv 0$. Going over t, letter by letter from the left to the right, whenever we 'meet' one of the letters $+$, $\cdot$ we put $n :\equiv n/$ until we reach the end of the word t.

The proof is by induction on $numc(t)$. If $numc(t)$ is w, then t is a numeral and sint-algorithm stops. If $numc(t)$ is not w, suppose that $(P * Q)$ is the first subterm of t, where $*$ is $+$ or $\cdot$. According to Lemma 1.4 and Lemma 1.8 let a numeral m be $sint[(P * Q)]$. Replacing $(P * Q)$ by m from t we obtain a new term t'. In virtue Lemma 1.5 and Lemma 1.9 sint-algorithm reduces to the sint-algorithm applied to t', whose $numc$ is the predecessor of $numc(t)$. The proof is complete.

Further, we shall establish some properties of the sint-algorithm. First, according to its definition we have

$$(1.9) \qquad Num \vdash t = sint(t) \qquad (t \text{ is any ground } L\text{-term})$$

Next, we have the following

Lemma 1.10. *Let A and B be any ground L-terms. Then we have the following equalities*

(i) $\quad sint((A * B)) \equiv sint((sint(A) * sint(B))) \quad$ (*$*$ is $+$ or $\cdot$*)

(ii) $\quad sint(A/) \equiv sint(A)/$

Proof. (i) We distinguish the following cases

(a) $\quad A$ is a numeral, B is a numeral

(b) $\quad A$ is a numeral, B is not a numeral

(c) $\quad A$ is not a numeral

In the case (a) the equality (i) reduces to the true equality $sint[(A*B)] \equiv sint[(A*B)]$, since $sint[A] \equiv A$, $sint[B] \equiv B$.

In the case (b) the first subterm of $(A*B)$ is the first subterm of B. By Lemma 1.3 sint-algorithm applied to $(A*B)$ reduces to sint-algorithm applied to $(A*sint[B])$, i.e. to $(sint[A]*sint[B])$ and the proof is complete.

In the case (c) the first subterm of $(A*B)$ is the first subterm of A. By Lemma 1.3 sint-algorithm applied to $(A*B)$ reduces to sint-algorithm applied to $(sint[< A > *B]$. Now we have the case (a) or (b). So, sint-algorithm applied to $((sint[A]*B)$ reduces to sint-algorithm applied to $((sint[A]*sint[B])$ and the proof is complete.

(ii) This equality is trivial.

Next, by Lemma 1.10 we shall prove the following

Lemma 1.11. *Let P, Q be any ground L-terms. Then the following equalities hold*

$$sint((P+0)) \equiv sint[P], \quad sint((P+Q/)) \equiv sint((P+Q)/)$$

$$sint((P \cdot 0)) \equiv 0, \quad sint((P \cdot Q/)) \equiv sint(((P \cdot Q)+P))$$

Proof. For the first equality we have

$$\begin{aligned} sint[(P+0)] &\equiv sint[(sint[P]+sint[0])] \quad \text{(By Lemma 1.10)} \\ &\equiv sint[(sint[P]+0]] \\ &\equiv sint(P) \quad \text{Since by the sint-algorithm we have the equality} \end{aligned}$$

$$sint((m+0)) \equiv m, \text{ where } m \text{ is the numeral } sint(P)$$

For the second equality we have the following derivations:

$$\text{(j)} \quad \begin{aligned} sint((P+Q/)) &\equiv sint((sint(P)+sint(Q/))) && \text{(By Lemma 1.10)} \\ &\equiv sint[(sint[P]+sint[Q]/)) && \text{(By Lemma 1.10)} \end{aligned}$$

$$\text{(jj)} \quad sint[(P+Q)/] \equiv sint[P+Q]/ \quad \text{(By Lemma 1.10)}$$

$$\equiv \ sint[(sint[P] + sint[Q]]/$$

Since $sint[P]$ and $sint[Q]$ are numerals by definition of the sint-algoritm we have the equality

(jjj) $\quad sint[[sint[P] + sint[Q]/)) \ \equiv \ sint((sint(P) + sint(Q))/)$

The second equality follows immediately from (j), (jj),(jjj). The third and fourth equality can be proved in a similar way.

Now, in order to see easier the sint-algorithm in general, we introduce its abridged version, the so called the *abridged sint-algorithm.* This algorithm[5] instead of substitutions of the form (σ) in (1.7) uses the following substitutions

$$(P + Q) \longrightarrow s_1, \qquad (P \cdot Q) \longrightarrow s_2$$

where P, Q are numerals and s_1, s_2 are $sint[(P+Q)]$, $sint[(P \cdot Q)]$ respectively.

As a matter of fact, Example 1.1 illustrates the abridged sint-algorithm. Here is another example

Example 1.3. Using the denotations : A is $0//$, B is $0/$, C is $0///$, D is 0 find $sint[(((A + A) + B) + (A \cdot (C + D)))]$.

Solution. We have the following abridged steps

$(((A + A) + B) + (A \cdot (C + D)))$ $\qquad$ $(A + A)$ is the first subterm and $sint[(A + A)]$ is $0////$

$\longrightarrow \quad ((a//// + B) + (A \cdot (C + D)))$ Replacing $(A + A)$ by $0////$. Now

the first subterm is $(0//// + B)$, whose $sint$ is $0/////$

$\longrightarrow \quad (0///// + (A \cdot (C + D)))$ Replacing $(0//// + B)$ by $0/////$. Now

the first subterm is $(C + D)$, whose $sin[t]$ is $0///$

$\longrightarrow \quad (a///// + (A \cdot 0///))$ Replacing $(C + D)$ by $0///$. Now the

[5] According to Lemma 1.5 and Lemma 1.9

first subterm is $(A \cdot 0///)$, whose $sin[t]$ is $0//////$

$$\longrightarrow \quad (0///// + 0//////)$$

$$\longrightarrow \quad 0/////////// \text{ Since } sint[(0///// + 0//////)] \text{ is } 0///////////$$

The result is $0///////////$. Next, we generalize Lemma 1.3 :

Lemma 1.12. *Let a ground term t has a subterm T, not a numeral, and let m be $sint[T]$. Then the equality $sint[t < T >] \equiv sint[t < m >]$ holds.*

Proof. Term t can be viewed in this form

$$LTR \qquad (L,\ R \text{ are some words})$$

The abridged sint-algorithm at each step removes one simple subterm of the $(P * Q)$; in other words, it eliminates one pair of symbols (and). Consequently, at some step the initial term t will reduce to a new term t' such that:

T is a subterm of t', and the first subterm of T is also the first subterm of t'

On the one hand, by Lemma 1.3 we have the equality

$$(*1) \qquad sint[t'[T]] \equiv sint[t' < m >]$$

On the other hand, by applying the abridged sint-algorithm to t at some step we obtain term t', consequently t' can be obtained at some step by applying the sint-algorithm to t. Therefore we have the equality

$$(*2) \qquad sint[t < T >] \equiv sint[t' < T >]$$

From $(*1), (*2)$ it follows the equality $sint[t < T >] \equiv sint[t < m >]$, since $sint[t' < m >]$
$\equiv sint[t < m >]$, and the proof is complete.

The following lemma is an immediate consequence of Lemma 1.12.

Lemma 1.13. *Let t be a ground L-term and P its subterm. If Q is any ground L-term such that $sint[P] \equiv sint[Q]$ then the following equality is true*

$$sint[t[< P >] \equiv sint[t < Q >]$$

Now we prove the following main theorem

Theorem 1.2. *Let t_1, t_2 be any L-terms. The following equivalence*

$$Num \vdash t_1 = t_2 \qquad \textit{iff} \qquad sint[t_1] \equiv sint[t_2]$$

holds.

Proof. *If* part is expressed by (1.9). In order to prove *only-if* part suppose that $t_1 = t_2$ is a theorem of Num. Since Num is an equational theory there is a proof of the following type:

> From the ground term t_1 to the ground term t_2 there exists a finite sequence of ground terms
>
> $(\Delta) \qquad w_1, w_2, ..., w_k \qquad (w_1 \text{ is } t_1,\ w_k \text{ is } t_2)$
>
> in which any w_{i+1} becomes from w_i by replacing some subterm A of w_i by a new term B, under condition that the equality $A = B$ or the equality $B = A$ is one of the (1.5) equalities.

Now replacing each w_i by $sint(w_i)$ from (Δ) we obtain the following sequence

$$sint[w_1],\ sint[w_2],\ ...,\ sint[w_k]$$

Bearing in mind Lemma 1.11 and Lemma 1.13 we see that all members of this sequence are the same word (i.e. the same numeral). Thus, we infer the equality $sint[t_1] \equiv sint[t_2]$ and the proof is complete.

As the first corollary of this theorem we see that the equality $0/ = 0$ can not be a theorem of Num, since $0/$ and 0 are different words. In general we have:

(1.10) If m is any numeral then equality $m/ = 0$ is not a theorem of Num.

In the theory Num we also have the following meta-implication

(1.11) $\quad Num \vdash X/ = Y/ \quad \longrightarrow \quad Num \vdash X = Y$

Indeed, if $Num \vdash X/ = Y/$ then,by Theorem 1.2, the words $sint[X/]$, $sint[Y/]$

are equal, hence we conclude that the words $sint[X]$, $sint[Y]$ are equal, consequently $Num \vdash X = Y$

Notice that it turns out that Peano arithetics besided the axioms for equality has only the given definiton for numerals.

REFERENCES

[1] K. Gödel, *Über formal unentsheidbare Sätze der Principia Mathematica und verwander Systeme,* 1 Monatsh. Math. Phys. 1931, 38, S. 173-198,

[2] E. Mendelson *Introduction to Mathematical Logic, 1963*

[3] J. R. Shoenfield *Mathematical Logic, 1967*