### Do We Need Recursion?

#### Vítězslav Švejdar

Dept. of Logic, College of Arts, Charles University in Prague http://www.cuni.cz/~svejdar/

Logica 19, Hejnice, June 24-28, 2019

Syntactic notions without recursion



#### Recursion in various situations. Is its use necessary?

#### The expressive power of bounded conditions and formulas

Arithmetization of syntactic notions without recursion

Vítek Švejdar, Prague

The equations  $z^0 = 1$  and  $z^{x+1} = z^x \cdot z$  derive the exponential function  $[x, z] \mapsto z^x$  by primitive recursion from g and h where g(z) = 1 and  $h(v, x, z) = v \cdot z$ .

< @ →

The equations  $z^0 = 1$  and  $z^{x+1} = z^x \cdot z$  derive the exponential function  $[x, z] \mapsto z^x$  by primitive recursion from g and h where g(z) = 1 and  $h(v, x, z) = v \cdot z$ .

The equation  $f(x) = g(\mu v(g(v) \notin \{f(0), ..., f(x-1)\}))$ derives *f* from *g* by course-of-values recursion (and minimization). If Rng(*g*) is infinite, then *f* is one-to-one and Rng(*f*) = Rng(*g*).

The equations  $z^0 = 1$  and  $z^{x+1} = z^x \cdot z$  derive the exponential function  $[x, z] \mapsto z^x$  by primitive recursion from g and h where g(z) = 1 and  $h(v, x, z) = v \cdot z$ .

The equation  $f(x) = g(\mu v(g(v) \notin \{f(0), ..., f(x-1)\}))$ derives *f* from *g* by course-of-values recursion (and minimization). If Rng(*g*) is infinite, then *f* is one-to-one and Rng(*f*) = Rng(*g*).

Consider the definition: *t* is a term in the arithmetic language if *t* is the constant 0, or *t* is a variable, or *t* has one of the forms  $s(t_1), +(t_1, t_2)$  or  $(t_1, t_2)$  where  $t_1$  and  $t_2$  are terms.

The equations  $z^0 = 1$  and  $z^{x+1} = z^x \cdot z$  derive the exponential function  $[x, z] \mapsto z^x$  by primitive recursion from g and h where g(z) = 1 and  $h(v, x, z) = v \cdot z$ .

The equation  $f(x) = g(\mu v(g(v) \notin \{f(0), ..., f(x-1)\}))$ derives *f* from *g* by course-of-values recursion (and minimization). If  $\operatorname{Rng}(g)$  is infinite, then *f* is one-to-one and  $\operatorname{Rng}(f) = \operatorname{Rng}(g)$ .

Consider the definition: *t* is a term in the arithmetic language if *t* is the constant 0, or *t* is a variable, or *t* has one of the forms  $s(t_1), +(t_1, t_2)$  or  $(t_1, t_2)$  where  $t_1$  and  $t_2$  are terms.

Once *variables* are defined (say, as strings like v1011), a programmer can write a procedure that decides what is and what is not a term by making calls to itself.

The equations  $z^0 = 1$  and  $z^{x+1} = z^x \cdot z$  derive the exponential function  $[x, z] \mapsto z^x$  by primitive recursion from g and h where g(z) = 1 and  $h(v, x, z) = v \cdot z$ .

The equation  $f(x) = g(\mu v(g(v) \notin \{f(0), ..., f(x-1)\}))$ derives *f* from *g* by course-of-values recursion (and minimization). If Rng(*g*) is infinite, then *f* is one-to-one and Rng(*f*) = Rng(*g*).

Consider the definition: *t* is a term in the arithmetic language if *t* is the constant 0, or *t* is a variable, or *t* has one of the forms  $s(t_1), +(t_1, t_2)$  or  $(t_1, t_2)$  where  $t_1$  and  $t_2$  are terms.

Coding of syntactic objects: the term +(v1,0) is the number  $43 \cdot 128^6 + 40 \cdot 128^5 + 118 \cdot 128^4 + 49 \cdot 128^3 + 44 \cdot 128^2 + 48 \cdot 128 + 41$ . The codes 43, 40, 118, ... are taken from modified ascii table.

The equations  $z^0 = 1$  and  $z^{x+1} = z^x \cdot z$  derive the exponential function  $[x, z] \mapsto z^x$  by primitive recursion from g and h where g(z) = 1 and  $h(v, x, z) = v \cdot z$ .

The equation  $f(x) = g(\mu v(g(v) \notin \{f(0), ..., f(x-1)\}))$ derives *f* from *g* by course-of-values recursion (and minimization). If Rng(*g*) is infinite, then *f* is one-to-one and Rng(*f*) = Rng(*g*).

Consider the definition: *t* is a term in the arithmetic language if *t* is the constant 0, or *t* is a variable, or *t* has one of the forms  $s(t_1), +(t_1, t_2)$  or  $(t_1, t_2)$  where  $t_1$  and  $t_2$  are terms.

If terms are (identified with) natural numbers, then the above definition is an application of course-of-values recursion.

### So where do we meet recursion?

#### 1. In some nice proofs, or in programming languages like in the proof that every infinite recursively enumerable set is the range of a one-to-one recursive function.

### So where do we meet recursion?

#### 1. In some nice proofs, or in programming languages like in the proof that every infinite recursively enumerable set is the range of a one-to-one recursive function.

#### 2. In the basic definitions in computability theory:

A function is *primitive recursive* if it can be derived from three initial functions using primitive recursion and composition. A function is *partial recursive* if it can be derived from the same initial functions using primitive recursion, composition and minimization.

### So where do we meet recursion?

#### 1. In some nice proofs, or in programming languages like in the proof that every infinite recursively enumerable set is the range of a one-to-one recursive function.

#### 2. In the basic definitions in computability theory:

A function is *primitive recursive* if it can be derived from three initial functions using primitive recursion and composition. A function is *partial recursive* if it can be derived from the same initial functions using primitive recursion, composition and minimization.

#### 3. In the definitions of syntactic notions in logic:

terms, formulas, free and bound occurrences of variables, substitutability of terms, the substitution operation itself.

If *f* is derived from *g* by minimization,  $f(x) = \mu v(g(x, v) = 0)$ , then  $y = f(x) \Leftrightarrow g(x, y) = 0 \& \forall v < y(g(x, v) \neq 0)$ .

If *f* is derived from *g* by minimization,  $f(x) = \mu v(g(x, v) = 0)$ , then  $y = f(x) \Leftrightarrow g(x, y) = 0 \& \forall v < y(g(x, v) \neq 0)$ . If *f* is derived from *g* and *h* by composition,  $f = h \circ g$ , then  $y = f(x) \Leftrightarrow \exists v(g(x) = v \& h(v) = y)$ .

< @ →

If *f* is derived from *g* by minimization,  $f(x) = \mu v(g(x, v) = 0)$ , then  $y = f(x) \Leftrightarrow g(x, y) = 0 \& \forall v < y(g(x, v) \neq 0)$ . If *f* is derived from *g* and *h* by composition,  $f = h \circ g$ , then  $y = f(x) \Leftrightarrow \exists v(g(x) = v \& h(v) = y)$ . However there is nothing in the arithmetic language (in logic) that would directly correspond to recursion.

If *f* is derived from *g* by minimization,  $f(x) = \mu v(g(x, v) = 0)$ , then  $y = f(x) \Leftrightarrow g(x, y) = 0 \& \forall v < y(g(x, v) \neq 0)$ . If *f* is derived from *g* and *h* by composition,  $f = h \circ g$ , then  $y = f(x) \Leftrightarrow \exists v(g(x) = v \& h(v) = y)$ . However there is nothing in the arithmetic language (in logic) that would directly correspond to recursion.

#### One possible approach (Oddifreddi [Odi89])

If the set of initial functions (normally consisting of  $x \mapsto x + 1$ ,  $x \mapsto 0$  and  $[x_1, \ldots, x_k] \mapsto x_j$ )

If *f* is derived from *g* by minimization,  $f(x) = \mu v(g(x, v) = 0)$ , then  $y = f(x) \Leftrightarrow g(x, y) = 0 \& \forall v < y(g(x, v) \neq 0)$ . If *f* is derived from *g* and *h* by composition,  $f = h \circ g$ , then  $y = f(x) \Leftrightarrow \exists v(g(x) = v \& h(v) = y)$ . However there is nothing in the arithmetic language (in logic) that would directly correspond to recursion.

#### One possible approach (Oddifreddi [Odi89])

If the set of initial functions (normally consisting of  $x \mapsto x + 1$ ,  $x \mapsto 0$  and  $[x_1, ..., x_k] \mapsto x_j$ ) is extended by adding  $[x, y] \mapsto x + y$ ,  $[x, y] \mapsto x \cdot y$  and e where e(x, y) = 1 if x = y and e(x, y) = 0 otherwise,

If *f* is derived from *g* by minimization,  $f(x) = \mu v(g(x, v) = 0)$ , then  $y = f(x) \Leftrightarrow g(x, y) = 0 \& \forall v < y(g(x, v) \neq 0)$ . If *f* is derived from *g* and *h* by composition,  $f = h \circ g$ , then  $y = f(x) \Leftrightarrow \exists v(g(x) = v \& h(v) = y)$ . However there is nothing in the arithmetic language (in logic) that would directly correspond to recursion.

#### One possible approach (Oddifreddi [Odi89])

If the set of initial functions (normally consisting of  $x \mapsto x + 1$ ,  $x \mapsto 0$  and  $[x_1, ..., x_k] \mapsto x_j$ ) is extended by adding  $[x, y] \mapsto x + y$ ,  $[x, y] \mapsto x \cdot y$  and e where e(x, y) = 1 if x = y and e(x, y) = 0 otherwise, then primitive recursion can be dropped from the definition.

If *f* is derived from *g* by minimization,  $f(x) = \mu v(g(x, v) = 0)$ , then  $y = f(x) \Leftrightarrow g(x, y) = 0 \& \forall v < y(g(x, v) \neq 0)$ . If *f* is derived from *g* and *h* by composition,  $f = h \circ g$ , then  $y = f(x) \Leftrightarrow \exists v(g(x) = v \& h(v) = y)$ . However there is nothing in the arithmetic language (in logic) that would directly correspond to recursion.

#### One possible approach (Oddifreddi [Odi89])

If the set of initial functions (normally consisting of  $x \mapsto x + 1$ ,  $x \mapsto 0$  and  $[x_1, ..., x_k] \mapsto x_j$ ) is extended by adding  $[x, y] \mapsto x + y$ ,  $[x, y] \mapsto x \cdot y$  and e where e(x, y) = 1 if x = y and e(x, y) = 0 otherwise, then primitive recursion can be dropped from the definition.

#### Another option

Using  $\Delta_0$  conditions.

### Examples of $\Delta_0$ conditions (bounded conditions) $\exists d \leq b (d \cdot a = b)$ , can be written as $a \mid b$ ;

< 67 ►

Examples of  $\Delta_0$  conditions (bounded conditions)  $\exists d \leq b(d \cdot a = b)$ , can be written as  $a \mid b$ ;  $\forall d \leq (a+b)(d \mid a \& d \mid b \Rightarrow d = 1)$ ;

< @ >

Examples of  $\Delta_0$  conditions (bounded conditions)  $\exists d \leq b(d \cdot a = b)$ , can be written as  $a \mid b$ ;  $\forall d \leq (a+b)(d \mid a \& d \mid b \Rightarrow d = 1)$ ;  $a > 1 \& \forall d < a(d \mid a \Rightarrow d = 1)$ , can be written as Prime(a);

< 47 >

Examples of  $\Delta_0$  conditions (bounded conditions)  $\exists d \leq b(d \cdot a = b)$ , can be written as  $a \mid b$ ;  $\forall d \leq (a+b)(d \mid a \& d \mid b \Rightarrow d = 1)$ ;  $a > 1 \& \forall d < a(d \mid a \Rightarrow d = 1)$ , can be written as Prime(a);  $\exists r < b(a = b \cdot q + r) \lor (b = 0 \& q = 0)$  expresses that q = Div(a, b).

Examples of  $\Delta_0$  conditions (bounded conditions)  $\exists d \leq b(d \cdot a = b)$ , can be written as  $a \mid b$ ;  $\forall d \leq (a + b)(d \mid a \& d \mid b \Rightarrow d = 1)$ ;  $a > 1 \& \forall d < a(d \mid a \Rightarrow d = 1)$ , can be written as Prime(a);  $\exists r < b(a = b \cdot q + r) \lor (b = 0 \& q = 0)$  expresses that q = Div(a, b). Saying that r = Mod(a, b) is similar.

Examples of  $\Delta_0$  conditions (bounded conditions)  $\exists d \le b(d \cdot a = b)$ , can be written as  $a \mid b$ ;  $\forall d \le (a+b)(d \mid a \& d \mid b \Rightarrow d = 1)$ ;  $a > 1 \& \forall d < a(d \mid a \Rightarrow d = 1)$ , can be written as Prime(a);  $\exists r < b(a = b \cdot q + r) \lor (b = 0 \& q = 0)$  expresses that q = Div(a, b). Saying that r = Mod(a, b) is similar.

 $\Delta_0$  conditions: a link between computability and logic *RE* sets are exactly the projections of  $\Delta_0$  conditions.

Examples of  $\Delta_0$  conditions (bounded conditions)  $\exists d \le b(d \cdot a = b)$ , can be written as  $a \mid b$ ;  $\forall d \le (a+b)(d \mid a \& d \mid b \Rightarrow d = 1)$ ;  $a > 1 \& \forall d < a(d \mid a \Rightarrow d = 1)$ , can be written as Prime(a);  $\exists r < b(a = b \cdot q + r) \lor (b = 0 \& q = 0)$  expresses that q = Div(a, b). Saying that r = Mod(a, b) is similar.

#### $\Delta_0$ conditions: a link between computability and logic

RE sets are exactly the projections of  $\Delta_0$  conditions. If  $\Delta_0$ -formulas are introduced (defined as a subclass of all arithmetic formulas),

Examples of  $\Delta_0$  conditions (bounded conditions)  $\exists d \le b(d \cdot a = b)$ , can be written as  $a \mid b$ ;  $\forall d \le (a+b)(d \mid a \& d \mid b \Rightarrow d = 1)$ ;  $a > 1 \& \forall d < a(d \mid a \Rightarrow d = 1)$ , can be written as Prime(a);  $\exists r < b(a = b \cdot q + r) \lor (b = 0 \& q = 0)$  expresses that q = Div(a, b). Saying that r = Mod(a, b) is similar.

### $\varDelta_0$ conditions: a link between computability and logic

*RE* sets are exactly the projections of  $\Delta_0$  conditions. If  $\Delta_0$ -formulas are introduced (defined as a subclass of all arithmetic formulas), then  $\Delta_0 = \Delta_0^{\mathbb{N}}$ .

Examples of  $\Delta_0$  conditions (bounded conditions)  $\exists d \le b(d \cdot a = b)$ , can be written as  $a \mid b$ ;  $\forall d \le (a+b)(d \mid a \& d \mid b \Rightarrow d = 1)$ ;  $a > 1 \& \forall d < a(d \mid a \Rightarrow d = 1)$ , can be written as Prime(a);  $\exists r < b(a = b \cdot q + r) \lor (b = 0 \& q = 0)$  expresses that q = Div(a, b). Saying that r = Mod(a, b) is similar.

#### $\Delta_0$ conditions: a link between computability and logic

*RE* sets are exactly the projections of  $\Delta_0$  conditions. If  $\Delta_0$ -formulas are introduced (defined as a subclass of all arithmetic formulas), then  $\Delta_0 = \Delta_0^{\mathbb{N}}$ .

#### The expressive power of $\Delta_0$ -formulas Is the condition $y = z^x$ bounded?

Examples of  $\Delta_0$  conditions (bounded conditions)  $\exists d \le b(d \cdot a = b)$ , can be written as  $a \mid b$ ;  $\forall d \le (a+b)(d \mid a \& d \mid b \Rightarrow d = 1)$ ;  $a > 1 \& \forall d < a(d \mid a \Rightarrow d = 1)$ , can be written as Prime(a);  $\exists r < b(a = b \cdot q + r) \lor (b = 0 \& q = 0)$  expresses that q = Div(a, b). Saying that r = Mod(a, b) is similar.

#### $\Delta_0$ conditions: a link between computability and logic

*RE* sets are exactly the projections of  $\Delta_0$  conditions. If  $\Delta_0$ -formulas are introduced (defined as a subclass of all arithmetic formulas), then  $\Delta_0 = \Delta_0^{\mathbb{N}}$ .

#### The expressive power of $\Delta_0$ -formulas

Is the condition  $y = z^x$  bounded? Is the set { y ;  $\exists x(y = 2^x)$  } bounded?

Examples of  $\Delta_0$  conditions (bounded conditions)  $\exists d \le b(d \cdot a = b)$ , can be written as  $a \mid b$ ;  $\forall d \le (a+b)(d \mid a \& d \mid b \Rightarrow d = 1)$ ;  $a > 1 \& \forall d < a(d \mid a \Rightarrow d = 1)$ , can be written as Prime(a);  $\exists r < b(a = b \cdot q + r) \lor (b = 0 \& q = 0)$  expresses that q = Div(a, b). Saying that r = Mod(a, b) is similar.

#### $\Delta_0$ conditions: a link between computability and logic

*RE* sets are exactly the projections of  $\Delta_0$  conditions. If  $\Delta_0$ -formulas are introduced (defined as a subclass of all arithmetic formulas), then  $\Delta_0 = \Delta_0^{\mathbb{N}}$ .

#### The expressive power of $\Delta_0$ -formulas

Is the condition  $y = z^x$  bounded? Is the set { y ;  $\exists x(y = 2^x)$  } bounded? Answer:  $\exists x(y = 2^x)$  is equivalent to

Examples of  $\Delta_0$  conditions (bounded conditions)  $\exists d \le b(d \cdot a = b)$ , can be written as  $a \mid b$ ;  $\forall d \le (a+b)(d \mid a \& d \mid b \Rightarrow d = 1)$ ;  $a > 1 \& \forall d < a(d \mid a \Rightarrow d = 1)$ , can be written as Prime(a);  $\exists r < b(a = b \cdot q + r) \lor (b = 0 \& q = 0)$  expresses that q = Div(a, b). Saying that r = Mod(a, b) is similar.

#### $\Delta_0$ conditions: a link between computability and logic

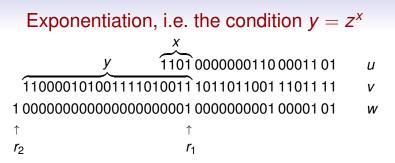
*RE* sets are exactly the projections of  $\Delta_0$  conditions. If  $\Delta_0$ -formulas are introduced (defined as a subclass of all arithmetic formulas), then  $\Delta_0 = \Delta_0^{\mathbb{N}}$ .

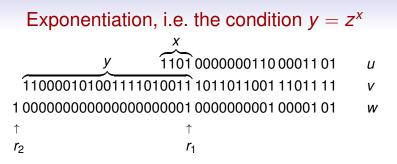
#### The expressive power of $\Delta_0$ -formulas

Is the condition  $y = z^x$  bounded? Is the set { y ;  $\exists x(y = 2^x)$  } bounded? Answer:  $\exists x(y = 2^x)$  is equivalent to  $\forall v \le y(v \mid y \rightarrow (v = 1 \lor 2 \mid v))$ .

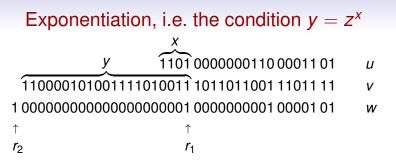
### Exponentiation, i.e. the condition $y = z^x$

#### 





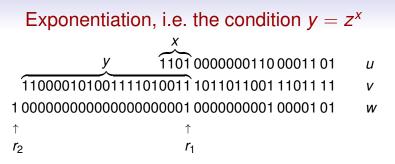
Let  $\operatorname{ExpW}(y, x, z, u, v, w)$  be a formula (which obviously is  $\Delta_0$ ) that describes this date structure. It says that if an item in *u* is *t* and the corresponding item in *v* is *s*, then either the next items are 2*t* and *s*<sup>2</sup>, or they are 2*t* + 1 and *s*<sup>2</sup> · *z*, etc.



Let  $\operatorname{ExpW}(y, x, z, u, v, w)$  be a formula (which obviously is  $\Delta_0$ ) that describes this date structure. It says that if an item in *u* is *t* and the corresponding item in *v* is *s*, then either the next items are 2*t* and *s*<sup>2</sup>, or they are 2*t* + 1 and *s*<sup>2</sup> · *z*, etc. Then the formula

$$\exists u \exists v \exists w \mathsf{ExpW}(y, x, z, u, v, w) \lor \\ \lor (x = 0 \& y = 1) \lor (x \neq 0 \& z < 2 \& y = z)$$

expresses that  $y = z^x$ .



Let  $\operatorname{ExpW}(y, x, z, u, v, w)$  be a formula (which obviously is  $\Delta_0$ ) that describes this date structure. It says that if an item in *u* is *t* and the corresponding item in *v* is *s*, then either the next items are 2t and  $s^2$ , or they are 2t + 1 and  $s^2 \cdot z$ , etc. Then the formula

$$\exists u \exists v \exists w \mathsf{ExpW}(y, x, z, u, v, w) \lor \\ \lor (x = 0 \& y = 1) \lor (x \neq 0 \& z < 2 \& y = z)$$

expresses that  $y = z^x$ . The number w does not exceed  $y^3$ .

### More $\Delta_0$ -formulas, strings

NPB(x) = y, the number of positive bits in the binary expansion of x is y (Appendix).

< @ >

NPB(x) = y, the number of positive bits in the binary expansion of x is y (Appendix).

Being a string: no digit in the 128-ary notation is zero.

NPB(x) = y, the number of positive bits in the binary expansion of x is y (Appendix).

Being a string: no digit in the 128-ary notation is zero.

Length of a string w: the least z such that  $w < 128^z$ .

NPB(x) = y, the number of positive bits in the binary expansion of x is y (Appendix).

Being a string: no digit in the 128-ary notation is zero.

Length of a string w: the least z such that  $w < 128^z$ .

Concatenation of two strings:  $w_1 \cdot 128^{Lh(w_1)} + w_2$ .

NPB(x) = y, the number of positive bits in the binary expansion of x is y (Appendix).

Being a string: no digit in the 128-ary notation is zero.

Length of a string w: the least z such that  $w < 128^z$ .

Concatenation of two strings:  $w_1 \cdot 128^{Lh(w_1)} + w_2$ .

Number of occurrences of a character: if *w* is the number  $83 \cdot 128^6 + 40 \cdot 128^5 + 83 \cdot 128^4 + 40 \cdot 128^3 + 48 \cdot 128^2 + 41 \cdot 128 + 41$ , then NOcc(48, *w*) = 1 and NOcc(41, *w*) = 2.

NPB(x) = y, the number of positive bits in the binary expansion of x is y (Appendix).

Being a string: no digit in the 128-ary notation is zero.

Length of a string w: the least z such that  $w < 128^z$ .

Concatenation of two strings:  $w_1 \cdot 128^{Lh(w_1)} + w_2$ .

Number of occurrences of a character: if *w* is the number  $83 \cdot 128^6 + 40 \cdot 128^5 + 83 \cdot 128^4 + 40 \cdot 128^3 + 48 \cdot 128^2 + 41 \cdot 128 + 41$ , then NOcc(48, *w*) = 1 and NOcc(41, *w*) = 2. We can also write NOcc(0, S(S(0))) = 1 and NOcc(), S(S(0))) = 2.

NPB(x) = y, the number of positive bits in the binary expansion of x is y (Appendix).

Being a string: no digit in the 128-ary notation is zero.

Length of a string w: the least z such that  $w < 128^z$ .

Concatenation of two strings:  $w_1 \cdot 128^{Lh(w_1)} + w_2 = w_1 w_2$ .

Number of occurrences of a character: if *w* is the number  $83 \cdot 128^6 + 40 \cdot 128^5 + 83 \cdot 128^4 + 40 \cdot 128^3 + 48 \cdot 128^2 + 41 \cdot 128 + 41$ , then NOcc(48, *w*) = 1 and NOcc(41, *w*) = 2. We can also write NOcc(0, S(S(0))) = 1 and NOcc(), S(S(0))) = 2.

A string *w* is balanced if  $Lh(w) \ge 2$ , NOcc((, w) = NOcc(), w), and NOcc((, u) > NOcc(), u) for any proper initial segment *u* of *w*. Example: (()()). Non-examples: v1011 and ()().

A string *w* is balanced if  $Lh(w) \ge 2$ , NOcc((, w) = NOcc(), w), and NOcc((, u) > NOcc(), u) for any proper initial segment *u* of *w*. Example: (()()). Non-examples: v1011 and ()().

Quasiterm is any variable, the single-letter string 0, or any string of the form S(w), +(w) or  $\cdot(w)$  where (w) is a balanced string. Examples: +((0)) and S(()()()).

A string *w* is balanced if  $Lh(w) \ge 2$ , NOcc((, w) = NOcc(), w), and NOcc((, u) > NOcc(), u) for any proper initial segment *u* of *w*. Example: (()()). Non-examples: v1011 and ()().

Quasiterm is any variable, the single-letter string 0, or any string of the form S(w), +(w) or  $\cdot$  (w) where (w) is a balanced string. Examples: +((0)) and S(()()).

A quasiterm *t* is a *term* (abbreviated Term(*t*)) if every balanced substing (*w*) of *t* is either immediately preceded by the letter s and *w* is a quasiterm, or it is immediately preceded by + or  $\cdot$  and *w* has the form *u*, *v* where *u* and *v* are quasiterms.

A string *w* is balanced if  $Lh(w) \ge 2$ , NOcc((, w) = NOcc(), w), and NOcc((, u) > NOcc(), u) for any proper initial segment *u* of *w*. Example: (()()). Non-examples: v1011 and ()().

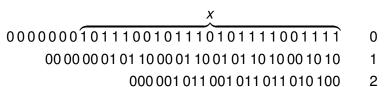
Quasiterm is any variable, the single-letter string 0, or any string of the form S(w), +(w) or  $\cdot$  (w) where (w) is a balanced string. Examples: +((0)) and S(()()).

A quasiterm *t* is a *term* (abbreviated Term(*t*)) if every balanced substing (*w*) of *t* is either immediately preceded by the letter s and *w* is a quasiterm, or it is immediately preceded by + or  $\cdot$  and *w* has the form *u*, *v* where *u* and *v* are quasiterms.

Properties of terms provable in PA: Any variable and the string 0 are terms. If  $t_1$  and  $t_2$  are terms, then  $s(t_1)$ ,  $+(t_1, t_2)$  and  $\cdot(t_1, t_2)$  are terms. Any term has one the forms  $s(t_1)$ ,  $+(t_1, t_2)$  or  $\cdot(t_1, t_2)$  unless it is a variable or the string 0.

# Appendix: the number of positive bits

Work with a *summation tree w* for a number *x*:



- 0001010001100110 3
  - 0010101100 4
    - 10001 5

where the bits (of the single number *w*) are split to several lines for better readability. It can be checked that y = NPB(x) is a  $\Delta_0$ -formula.

In the above example, the summation tree witnesses the fact that the number of positive bits in the number 24 308 687 is 17.

#### References

- J. H. Bennet. *On Spectra*. Dissertation, Princeton University, Princeton, NJ, 1962.
- S. Feferman. Arithmetization of metamathematics in a general setting. *Fundamenta Mathematicae*, 49:35–92, 1960.
- P. Hájek and P. Pudlák. Metamathematics of First Order Arithmetic. Springer, 1993.
- P. Odifreddi. Classical Recursion Theory. North-Holland, 1989.
- P. Pudlák. A definition of exponentiation by a bounded arithmetical formula. *Comm. Math. Univ. Carolinae*, 24(4):667–671, 1983.