

Chapter 2

Unary Algebras

Abstract We introduce the definition of unary algebra as well as subclasses of it called k -valued, strongly- k -valued and strongly- k -generated. Then we proceed with the simplification algorithm that transforms each system of equations into a more regular one at the expense of adding some definable constraints. Finally we give computational complexity characterization of SysTermSat over three-element unary algebras that depends on width of a special preorder constructed from given algebra.

Definition 2.1 We call an algebra \mathbf{A} *unary* if all of its basic operations are unary or constants.

Example 2.2 The algebra $\mathbf{A} = (\{0, 1, 2\}, f, g, h)$ with the following operations is unary.

x	f	g	h
0	1	0	0
1	0	1	0
2	0	0	1

We have announced in Sect. 1.1 that considering terms, instead of polynomials, is a more general approach. Since one can switch between terms and polynomials, when necessary, we are going to use terms during the presentation in the rest of the text.

The definition of $T(V)$ over \mathbf{A} is much simpler, when applied to unary algebras. Any term over a unary algebra $\mathbf{A} = (A, F)$:

- is either a variable x from V ,
- or has the form $f_1(f_2(\dots f_n(x) \dots))$ for some variable x from V and some sequence (f_1, \dots, f_n) of basic operations from F_1 ,
- or is an element c from F_0 ,
- or has the form $f_1(f_2(\dots f_n(c) \dots))$ for some element c from F_0 and some sequence (f_1, \dots, f_n) of basic operations from F_1 .

Since we will be solving term equations, we are interested how values of terms depend on valuation of variables. However, value of a term is defined by its term operation, thus we are going to switch from terms to term operations. A term over

\mathbf{A} has one or no variable, but for convenience we will treat all of them as unary term operations from $Cl_1(\mathbf{A})$.

Let t be a term over \mathbf{A} . If t has one variable x then we replace t by $t^{\mathbf{A}}(x)$, where $t^{\mathbf{A}} \in Cl_1(\mathbf{A})$. If t has no variable then we need a small discussion about replacement that we make. The value of t is constant, say $c \in A$. However, we can treat t as a term $t(x)$ with some variable x . This trick gives us that unary term operation that takes only value c belongs to $Cl_1(\mathbf{A})$. Since we are going to use only operations from $Cl_1(\mathbf{A})$, we denote by c the above unary term operation and call it a constant. Then we finally replace t with $c \in Cl_1(\mathbf{A})$, but without specifying a variable.

Given a term $t(x)$ we can compute its term operation $t^{\mathbf{A}}(x)$ in time $O(|t||A|)$. Every unary term operation can be described by its table of size $O(|A|)$. To compute $t^{\mathbf{A}}$ we start with the identity operation and compose basic operations that form sequence (f_1, \dots, f_n) in t . Dealing with t without a variable is even simpler.

We are going to show, that in problems that have terms on their input we can replace them by unary term operations without essential change of the complexity. If F_1 is nonempty then the number of terms over \mathbf{A} is infinite. However, if we consider unary term operations, the situation is different. Every unary term operation is a unary function over A , thus $Cl_1(\mathbf{A})$ is finite. This happens because many terms define the same term operations.

We compute the set $Cl_1(\mathbf{A})$ in time $O(|A|^{2|A|})$. Each unary term operation $p(x)$ is generated by a term over \mathbf{A} . Consider a shortest one, say $t_p(x) = f_1(f_2(\dots f_n(x) \dots))$. As $t_p(x)$ is the shortest, terms of the form $f_i(\dots f_n(x) \dots)$, for $i \in \{1 \dots n\}$ generate different term operations. As there are at most $|A|^{|A|}$ term operations, $n \leq |A|^{|A|}$. We generate all such sequences of length at most $|A|^{|A|}$ and find $t_p(x)$ for given $p(x)$. Similarly we check all sequences of the form $f_1(f_2(\dots f_n(c) \dots))$, for $c \in F_0$ and $n \leq |A|$. As we never consider \mathbf{A} as a part of the input, the complexity of the above process is constant for us. A careful implementation of the idea described above gives algorithm working in $O(|A|^{2|A|})$ time. Observe that $Pol_1(\mathbf{A})$ is $Cl_1(\mathbf{A})$ expanded by all missing constant operations. On the other hand $Cl_1(\mathbf{A})$ does not need to have all constant operations.

Example 2.3 For the algebra from the Example 2.2 the clone of unary term operations $Cl_1(\mathbf{A})$ contains 9 elements. The identity operation, corresponding to a variable, is denoted by id .

x	id	f	g	h	hh	fhh	ff	fg	fh
0	0	1	0	0	0	1	0	1	1
1	1	0	1	0	0	1	1	0	1
2	2	0	0	1	0	1	1	1	0

Observe that $Cl_1(\mathbf{A})$ contains two constant operations, hh and fhh . The clone of polynomial operations $Pol_1(\mathbf{A})$ contains also missing third constant operation equal to 2. The names of operations are not unique, as for example $hh = hf$ but it does not matter to us. For each particular element of $Cl_1(\mathbf{A})$ we can assign new symbol or choose unique term string, that defines it.

Lemma 2.4 *For a fixed unary algebra \mathbf{A} the problems of the form $\text{TERMSAT}(\mathbf{A})$, $\text{SYSTEMSAT}(\mathbf{A})$, $\text{CSYSTEMSAT}(\mathbf{A})$ are polynomially equivalent¹ to their counterparts in which on input the terms are replaced by term operations. The same happens with $\text{POLSAT}(\mathbf{A})$, $\text{SYSPOLSAT}(\mathbf{A})$ and $\text{CSYSPOLSAT}(\mathbf{A})$.*

Proof Since \mathbf{A} is not a part of the input there is a transformation working in linear time. Simply note that passing from a term t to $t^{\mathbf{A}}$ takes $O(|t|)$ time and passing from a term operation p to t_p takes constant time $O(|A|^{2|A|})$. \square

Lemma 2.4 allows us to switch between input representation by terms and term operations in each of the problem we consider. Therefore, when dealing with terms over unary algebras we will often denote by t both a term and its corresponding term operation. The same applies to polynomials and polynomial operations.

In view of all the remarks above, during the preprocessing of term equations given on input, we replace terms by their term operations:

$$x \rightarrow id(x),$$

$$f_1(f_2(\dots f_n(x)\dots)) \rightarrow (f_1^{\mathbf{A}} \dots f_n^{\mathbf{A}})(x),$$

where now $(f_1^{\mathbf{A}} \dots f_n^{\mathbf{A}})$ is an element of $\text{Clo}_1(\mathbf{A})$ and therefore has length 1. Similarly for a term without a variable we also remove unnecessary compositions and replace:

$$f_1(f_2(\dots f_n(c)\dots)) \rightarrow (f_1^{\mathbf{A}} \dots f_n^{\mathbf{A}})(c) \in \text{Clo}_1(\mathbf{A}).$$

Summarizing, every member of $T(V)$ over unary algebra \mathbf{A} can be replaced by $t(x)$, where $x \in V$ and $t \in \text{Clo}_1(\mathbf{A})$ or by a constant $c \in \text{Clo}_1(\mathbf{A})$. Remember all constants are expressible by polynomials, but not necessarily all of them by terms.

Definition 2.5 Given a unary term operation $t(x)$ we define: $\text{Var}(t(x)) = \{x\}$, while given a constant unary term operation c with unspecified variable, we put $\text{Var}(c) = \emptyset$. Moreover, for an equation $t \approx s$ we put $\text{Var}(t \approx s) = \text{Var}(t) \cup \text{Var}(s)$. Finally, for a set of equations S , we put $\text{Var}(S) = \bigcup_{e \in S} \text{Var}(e)$.

After all the preparations we have just made, we are ready for the following description of possible shapes of term equations.

Observation 2.6 With respect to left-right symmetry, each term equation has one of the following forms:

- no-variable equation
 - the equation has the form $c_1 \approx c_2$ for two constants $c_1, c_2 \in \text{Clo}_1(\mathbf{A})$,
- one-sided one-variable equation

¹We use here (and in the rest of the text) polynomial-time many-one reductions also known as polynomial transformations.

- the equation has the form $f(x) \approx c$ for some $x \in V$, $f \in Clo_1(\mathbf{A})$ and constant $c \in Clo_1(\mathbf{A})$,
- two-sided one-variable equation
 - the equation has the form $f(x) \approx g(x)$ for some $x \in V$ and $f, g \in Clo_1(\mathbf{A})$,
- two-sided two-variables equation
 - the equation has the form $f(x) \approx g(y)$ for some $x, y \in V$, $x \neq y$ and $f, g \in Clo_1(\mathbf{A})$.

At this moment we can easily see that solving systems of equations over unary algebra is interesting only if the number of equations is unbounded. Since every equation has at most two variables, we can solve every single equation in a quadratic time of the size of an algebra. Even solving k equations, where k is not part of the input, is polynomial in size of $|A|$. Therefore in the rest of the text we will only consider systems of equations.

Let I_1, I_2 be instances of either of the problems $SYSTEMSAT(\mathbf{A})$, $CSYSTEMSAT(\mathbf{A})$, $SYSPOLSAT(\mathbf{A})$ and $CSYSPOLSAT(\mathbf{A})$. We say I_1 and I_2 are *equivalent* if I_1 has a solution if and only if I_2 does. Since the size of all equations is uniformly bounded, we define the *size* of an instance I , denoted $|I|$, as the number of equations.

For a finite set A and a permutation $p : A \rightarrow A$ there is k such that $p^k(a) = a$ for all $a \in A$, indeed e.g. $k = |A|!$ works. Now if $p \in Clo_1(\mathbf{A})$, where \mathbf{A} is an algebra, p^{k-1} is a unary term operation and obviously p^{k-1} is the inverse of p . Thus $p \in Clo_1(\mathbf{A})$ implies $p^{-1} \in Clo_1(\mathbf{A})$.

Definition 2.7 A unary term operation t over a unary algebra $\mathbf{A} = (A, F)$ is:

- *k-valued*, if $|t(A)| \leq k$,
- *generic*, if $1 < |t(A)| < |A|$, so that t is neither a constant nor a permutation.

Definition 2.8 A set T of unary term operations is *k-valued* if every generic term operation in T is *k-valued*.

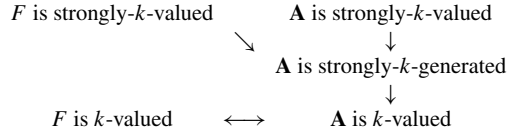
Definition 2.9 A set T of unary term operations over a unary algebra $\mathbf{A} = (A, F)$ is *strongly-k-valued* if there is $D \subseteq A$ with $|D| \leq k$ such that $t(A) \subseteq D$ for every generic $t \in T$.

Definition 2.10 Let (A, F) be a unary algebra. A set $C \subseteq Clo_1(A, F)$ is *strongly-k-generated* if $C = Clo_1(A, G)$ for some *strongly-k-valued* set $G \subseteq C$.

Definition 2.11 A unary algebra \mathbf{A} is:

- *k-valued* if $Clo_1(\mathbf{A})$ is *k-valued*.
- *strongly-k-valued* if $Clo_1(\mathbf{A})$ is *strongly-k-valued*.
- *strongly-k-generated* if $Clo_1(\mathbf{A})$ is *strongly-k-generated*.

Observation 2.12 Let $\mathbf{A} = (A, F)$ be a unary algebra. The following diagram shows straightforward dependencies between properties defined in Definitions 2.8–2.11.



With the next two examples we see that the implications in the right column can not be reversed.

Example 2.13 Let the algebra $\mathbf{A} = (\{0, 1, 2, 3\}, F)$, with $F = \{p, f\}$ be defined by:

x	p	f	pf
0	1	1	0
1	0	1	0
2	2	2	2
3	3	2	2

Then the set F is strongly-2-valued. Therefore the set $Clo_1(\mathbf{A}) = \{id, p, f, pf\}$ is strongly-2-generated. Thus \mathbf{A} is strongly-2-generated but not strongly-2-valued, as $|f(A) \cup pf(A)| = 3$.

Example 2.14 Let the algebra $\mathbf{A} = (\{0, 1, 2, 3\}, F)$, with $F = \{f, g\}$ be defined by:

x	f	g	fg
0	0	0	0
1	0	0	0
2	0	2	0
3	3	0	0

Then the set F is 2-valued and $Clo_1(\mathbf{A}) = \{id, f, g, fg\}$ is 2-valued. Thus \mathbf{A} is 2-valued but not strongly-2-generated.

2.1 Simplification Algorithm

Definition 2.15 Let $\mathbf{A} = (A, F)$ be a unary algebra. The set of *definable constraints* over algebra \mathbf{A} is the smallest set $\mathcal{C}(\mathbf{A}) \subseteq \mathcal{P}(A)$ such that:

- if $t, s \in Clo_1(\mathbf{A})$ then $\{a \in A : t(a) = s(a)\} \in \mathcal{C}(\mathbf{A})$,
- if $C_1, C_2 \in \mathcal{C}(\mathbf{A})$ then $C_1 \cap C_2 \in \mathcal{C}(\mathbf{A})$,
- if $C \in \mathcal{C}(\mathbf{A})$ and $t \in Clo_1(\mathbf{A})$ then $t(C), t^{-1}(C) \in \mathcal{C}(\mathbf{A})$.

Note that, for a unary algebra \mathbf{A} the set $\mathcal{C}(\mathbf{A})$ is computable in $O(|A|^{2|A|})$ time, because there are at most $|A|^{2|A|}$ possibilities in each case of Definition 2.15.

Our next Lemma shows that each definable constraint is essentially expressible by a set of equations.

Lemma 2.16 *Let $\mathbf{A} = (A, F)$ be a unary algebra, $C \in \mathcal{C}(\mathbf{A})$ be a definable constraint and x be a variable. Then there is a set $S_C(x)$ of term equations over \mathbf{A} , with $V = \text{Var}(S_C(x))$ and $x \in V$ such that:*

- if $v : V \rightarrow A$ is a solution of $S_C(x)$ then $v(x) \in C$,
- if $a \in C$ then there exists a solution $v : V \rightarrow A$ of $S_C(x)$ such that $v(x) = a$.

Proof We induct on the complexity of how C is built according to the definition of $\mathcal{C}(\mathbf{A})$. If $C = \{a \in A : t(a) = s(a)\}$ then the set $S_C(x) = \{t(x) \approx s(x)\}$ satisfies the Lemma.

If $C = C_1 \cap C_2$ then first we rename all variables in $\text{Var}(S_{C_2}(x)) \setminus \{x\}$ so that $\text{Var}(S_{C_1}(x)) \cap \text{Var}(S_{C_2}(x)) = \{x\}$ and then we put $S_C(x) = S_{C_1}(x) \cup S_{C_2}(x)$. The required properties of solutions of $S_C(x)$ are obvious.

Finally, let $C = t^{-1}(s(C'))$ for $t, s \in Clo_1(\mathbf{A})$. Since $id \in Clo_1(\mathbf{A})$ this covers the last possibility of building C . First note that the set C' can be realized by a set $S_{C'}(y)$ of equations in which x does not appear. Now put $S_C(x) = S_{C'}(y) \cup \{t(x) \approx s(y)\}$. For short put $V' = \text{Var}(S_{C'}(y))$ and $V = \text{Var}(S_C(x))$ and note that $V = V' \cup \{x\}$. First suppose that $v : V \rightarrow A$ is a solution of $S_C(x)$. Then $v|_{V'}$ is a solution of $S_{C'}(y)$ and by the induction hypothesis we have $v(y) \in C'$. Since v satisfies also the new equation we get $t(v(x)) = s(v(y))$, which implies $v(x) \in t^{-1}(s(C')) = C$. Now assume $a \in t^{-1}(s(C'))$ so that $t(a) = s(a')$ for some $a' \in C'$. Again by the induction hypothesis we get a solution $v' : V' \rightarrow A$ of $S_{C'}(y)$ such that $v'(y) = a'$. Then the valuation $v : V \rightarrow A$ defined by:

$$v(z) = \begin{cases} v'(z), & \text{if } z \in V', \\ a, & \text{if } z = x, \end{cases}$$

satisfies all equations in $S_C(x)$ and moreover $v(x) = a$, as required. \square

We are going to present a procedure that transforms each system of equations into a more regular one at the expense of adding some definable constraints. More formally, this `Simplify` procedure takes an instance I of `SYSTEMSAT`(\mathbf{A}) to remove equations during an iterative process. Finally it produces an instance I' of `CSYSTEMSAT`(\mathbf{A}) where a constraint function C is allowed and I' is supposed to satisfy:

- (1) I' is equivalent to I ,
- (2) all equations in I' are of the form $q(x) = r(y)$, where $q, r \in Clo_1(\mathbf{A})$ and x, y are different variables. Both q and r are generic and $|q(C(x)) \cap r(C(y))| > 1$,
- (3) $C(x) \in \mathcal{C}(\mathbf{A})$ and $|C(x)| > 1$ for all $x \in \text{Var}(I')$.

The procedure also returns a boolean value, with `False` meaning that the instance I has no solutions at all. The `True` value does not say however that I has a solution but allows to replace I by its equivalent modification I' .

```

1 Simplify( I )
2   V := Var(I)
3   C(x) := A for all x ∈ V
4   I' := (I, C)
5
6   repeat
7     if exists ( x ∈ V, C(x) = ∅ ) then return False
8     for each e ∈ I' do
9       if e is c ≈ d and c, d are different constants then
10        return False
11       if e is c ≈ c and c is a constant then
12        remove e from I'
13       if e is t(x) ≈ c then
14        remove e from I'
15        C(x) := C(x) ∩ t-1(c)
16       if e is t(x) ≈ s(x) then
17        remove e from I'
18        C(x) := C(x) ∩ {a ∈ A : t(a) = s(a)}
19       if e is s(x) ≈ p(y) and x, y are different variables
20         and p is a permutation then
21        remove e from I'
22        t := p-1s
23        replace all occurrences of y in I' with t(x)
24        C(x) := C(x) ∩ t-1(C(y))
25       if e is t(x) ≈ s(y) and x, y are different variables
26         and t(C(x)) ∩ s(C(y)) = ∅ then
27        return False
28       if e is t(x) ≈ s(y) and x, y are different variables
29         and t(C(x)) ∩ s(C(y)) = {c} for some c ∈ A then
30        remove e from I'
31        C(x) := C(x) ∩ t-1(c)
32        C(y) := C(y) ∩ s-1(c)
33   until not changed( I' )
34
35   return True

```

Looking at what is done in the repeat loop we easily see that each time the instance I' is transformed to its equivalent form. Only a few words are needed to comment lines 19–24. The equation $s(x) \approx p(y)$ is equivalent to $p^{-1}s(x) \approx y$. Therefore introducing $t = p^{-1}s$ and replacing all occurrences of y by $t(x)$ together with updating the constraint for x by requiring $t(x) \in C(y)$ transforms I' to an equivalent form. Since I' is equivalent to I we get the answer for I by considering I' . Actually one can transform any solution of I' to a solution of I by remembering the replacements for variables from the set $V \setminus \text{Var}(I')$.

To see that (2) is satisfied observe that repeat loop removes all no-variable and one-variable equations. Suppose $q(x) \approx r(y)$ is a two-variables equation with

an operation, say q , that is not generic. If q is a permutation then the equation $q(x) \approx r(y)$ is removed in line 21. Otherwise q is a constant. However then $|q(C(x)) \cap r(C(y))| \leq |q(C(x))| = 1$ thus either **False** is returned in line 27 or the equation is removed in line 30. Finally, all two-variables equation that are left in I' satisfy $|q(C(x)) \cap r(C(y))| > 1$ thanks to condition from line 29.

If I' does not satisfy (2) then each run of the **repeat** loop either returns **False** or removes at least one equation. Therefore there are at most $|I|$ runs of the loop. On the other hand each single run can be done in linear time so that **Simplify** works in quadratic time.

To prove that $|C(x)| > 1$, first pick a variable $x \in \text{Var}(I')$. Then observe that line 7 gives $|C(x)| \neq 0$. Suppose $|C(x)| = 1$ and take a two-variables equation from I' that contains x . We immediately get $|q(C(x)) \cap r(C(y))| \leq |q(C(x))| \leq |C(x)| = 1$ which is a contradiction to the property (2).

The rest of the last property (3) is covered by the following Claim.

Claim 2.17 *Whenever **Simplify** sets a value of the constraint function C for a variable x we have $C(x) \in \mathcal{C}(\mathbf{A})$. Moreover, if **Simplify** procedure returns **False** then $\emptyset \in \mathcal{C}(\mathbf{A})$.*

Proof We initialize the constraint function C in line 3 by putting the value A . Obviously $A = \{a \in A : id(a) = id(a)\} \in \mathcal{C}(\mathbf{A})$. During the main loop of **Simplify** we change values of the constraint function C in five cases. We are going to show that after every change $C(x) \in \mathcal{C}(\mathbf{A})$. First time we make a change in line 15 by putting $C(x) := C(x) \cap t^{-1}(c)$ so that we need to argue that $t^{-1}(c) \in \mathcal{C}(\mathbf{A})$. However $\{c\}$ is the image of A by the constant term operation $c \in Clo_1(\mathbf{A})$ and then $t^{-1}(c)$ is definable. Similar argument applies to line 18 with $C(x) := C(x) \cap \{a \in A : t(a) = s(a)\}$ and to line 24 with $C(x) := C(x) \cap t^{-1}(C(y))$. The situation in lines 31–32 is very similar to the one from line 15. Again we have to argue that $\{c\} \in \mathcal{C}(\mathbf{A})$. This time however, we do not know that c is a constant expressible by a term operation from $Clo_1(\mathbf{A})$. But we have $\{c\} = t(C(x)) \cap s(C(y))$, thus $\{c\} \in \mathcal{C}(\mathbf{A})$, as required.

The **Simplify** procedure returns **False** in three cases. First time it happens in line 7, after detecting x with $C(x) = \emptyset$. Since $\mathcal{C}(\mathbf{A})$ contains all sets of the form $C(x)$ we have $\emptyset \in \mathcal{C}(\mathbf{A})$, as required. Next, in line 9 there is an equation $c \approx d$ with $c \neq d$. However $\emptyset = \{a \in A : c(a) = d(a)\} \in \mathcal{C}(\mathbf{A})$. Finally **False** is returned in line 27, after detecting that $\emptyset = t(C(x)) \cap s(C(y))$ is the intersection of images of definable constraints. Thus $\emptyset \in \mathcal{C}(\mathbf{A})$. \square

2.2 Three-Element Algebras

The first example of a finite algebra \mathbf{A} such that $\text{SYSPOLSAT}(\mathbf{A})$ is **NP**-complete goes back to Cook, when he had shown that **SATISFIABILITY** problem is **NP**-complete. This simply means that the boolean algebra $(2, \vee, \wedge, \neg)$ gives rise to **NP**-complete **TERMSAT**, **POLSAT**, **SYSTERMSAT** and **SYSPOLSAT**. Many other such examples can be derived from theorems of Chap. 1.

Our first lemma shows that **SYSTEMSAT** remains **NP**-complete even for some unary algebras (already with three elements).

Lemma 2.18 *There are finite unary algebras for which the problem **SYSTEMSAT** (**A**) is **NP**-complete.*

Proof A similar argument can be found in [FMS04]. Let $\mathbf{A} = (\{0, 1, 2\}, f, g, h)$ be an algebra with the following operations:

x	f	g	h
0	1	0	0
1	0	1	0
2	0	0	1

We have already mentioned in Chap. 1 that **SYSTEMSAT**(**A**) belongs to **NP**. To show that it is **NP**-complete we need the following version of the **SATISFIABILITY** problem:

POSITIVE-1-IN-3-SAT is a problem taking on its input a formula $F = C_1 \wedge \dots \wedge C_n$, in which each clause C_i is of the form $(x \vee y \vee z)$, where x, y, z are (non-negated) variables, and answering the question if there is a boolean valuation such that in each clause *exactly one* variable takes value 1. For example for a formula $(x \vee y \vee z) \wedge (x \vee t \vee v) \wedge (v \vee t \vee z)$ the positive answer can be witnessed by $(0 \vee \mathbf{1} \vee 0) \wedge (0 \vee 0 \vee \mathbf{1}) \wedge (\mathbf{1} \vee 0 \vee 0)$. It is easy to check by Schaefer [Sch78] result² (see also Garey and Johnson [GJ79]) that **POSITIVE-1-IN-3-SAT** is **NP**-complete.

Next we reduce **POSITIVE-1-IN-3-SAT** to systems of equations over the algebra **A**. A formula $F = C_1 \wedge \dots \wedge C_n$ is transformed into $3n$ equations as follows:

$$C_i = x \vee y \vee z \quad \rightsquigarrow \quad \begin{cases} f(v_i) \approx x \\ g(v_i) \approx y \\ h(v_i) \approx z, \end{cases}$$

where v_1, \dots, v_n are new variables not occurring in F .

It is easy to check that a solution of the equations exists if and only if the formula F is satisfiable according to **POSITIVE-1-IN-3-SAT** rules. If v_i takes the value 0 (1 or 2) then only x (y or z respectively) takes boolean value 1 to make C_i true. \square

The following Lemma shows that the essence of the complexity of solving equations is hidden in generic operations:

Lemma 2.19 *For a unary algebra $\mathbf{A} = (A, F)$ with no generic operations in F the problem **SYSTEMSAT**(**A**) is in **P**.*

² The relation defined by **POSITIVE-1-IN-3-SAT** is $\{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}$ and it is not closed under any of the operations listed in Fact 1.15.

Proof We apply `Simplify` procedure to an instance I . If we get `False` then there is no solution. Otherwise, because no operation in F , and thus in $Clo_1(\mathbf{A})$, is generic, the instance I' computed by `Simplify` has empty set of equations. Thus I' has a solution. \square

Since two-element unary algebras have no generic operations we immediately get from Lemma 2.19 that `SYSTEMSAT` over two-element unary algebras is in \mathbf{P} .

For our main Theorem of this section we need to define the following:

Definition 2.20 We call a unary algebra $\mathbf{A} = (A, F)$ *proper* if there is no $a \in A$ with $f(a) = a$ for all $f \in Clo_1(\mathbf{A})$.

Consider `SYSTEMSAT`(\mathbf{A}) over a unary algebra $\mathbf{A} = (A, F)$ that is not proper, with $a \in A$ being a witness described in Definition 2.20. The valuation putting a for all variables obviously satisfies all term equations over \mathbf{A} . Thus the problem `SYSTEMSAT`(\mathbf{A}) can be solved in a constant time, with positive answer for all instances.

For a unary algebra $\mathbf{A} = (A, F)$ and $f \in Clo_1(\mathbf{A})$ by $Ker(f) = \{(x, y) \in A^2 : f(x) = f(y)\}$ we denote the *kernel* of f .

Definition 2.21 For a unary algebra $\mathbf{A} = (A, F)$ we define a *preorder* \leq on $Clo_1(\mathbf{A})$ by putting $f \leq g$ if and only if $Ker(f) \subseteq Ker(g)$. We also put $\mathbf{P}(\mathbf{A}) = (Clo_1(\mathbf{A}), \leq)$.

Example 2.22 In the algebra $\mathbf{A} = (\{0, 1, 2\}, f, g, h)$, with the operations:

x	f	g	h
0	0	0	0
1	0	1	0
2	0	0	1

we have $g \leq f, h \leq f$ while g, h are incomparable.

By the width of an ordered set (or more generally, a preordered set) \mathbf{P} we mean the largest number of pairwise incomparable elements of \mathbf{P} .

For our next theorem we need two lemmas:

Lemma 2.23 *For a proper unary algebra \mathbf{A} with three elements the problem `SYSTEMSAT`(\mathbf{A}) is NP-complete if $\text{width}(\mathbf{P}(\mathbf{A})) = 3$.*

Proof To witness width 3 in the preorder $\mathbf{P}(\mathbf{A})$ the algebra \mathbf{A} must have 3 term operations f_0, f_1, f_2 with pairwise incomparable kernels. Without loss of generality we may assume that f_0, f_1, f_2 act as follows ($a_i \neq b_i$):

x	f_0	f_1	f_2
0	b_0	a_1	a_2
1	a_0	b_1	a_2
2	a_0	a_1	b_2

We are going to show that there are $f, g, h \in Clo_1(\mathbf{A})$ and $\perp \neq \top$ in A such that either $P1$ or $P2$ holds:

P1				P2			
x	f	g	h	x	f	g	h
0	\top	\perp	\perp	0	\perp	\perp	\perp
1	\perp	\top	\perp	1	\top	\top	\perp
2	\perp	\perp	\top	2	\top	\perp	\top

- Case 1: There exists $i \in \{0, 1, 2\}$ such that $i \neq b_i$.

Without loss of generality we may assume that $i = 0$ and $b_0 = 1$. Then for $f_3 := f_1 f_0 \in Clo_1(\mathbf{A})$ we have:

x	f_0	f_1	f_2	f_3
0	1	a_1	a_2	b_1
1	a_0	b_1	a_2	a_1
2	a_0	a_1	b_2	a_1

- Subcase 1.1: $\{a_1, b_1\} \neq \{0, 1\}$. Then $f_2 f_3, f_2 f_1, f_2$ satisfy:

x	$f_2 f_3$	$f_2 f_1$	f_2		x	$f_2 f_3$	$f_2 f_1$	f_2
0	b_2	a_2	a_2	or	0	a_2	b_2	a_2
1	a_2	b_2	a_2		1	b_2	a_2	a_2
2	a_2	a_2	b_2		2	b_2	b_2	b_2

i.e., $P1$ or $P2$ (with 0 and 2 interchanged) holds.

- Subcase 1.2: $\{a_1, b_1\} = \{0, 1\}$. Since we are not going to use f_0 any more, without loss of generality we may assume³ that $a_1 = 0, b_1 = 1$. Observe that if $\{a_2, b_2\} = \{0, 1\}$ then f_3, f_1, f_2 satisfy either $P1$ or $P2$ (with 0 and 2 interchanged). Let $\{a_2, b_2\} \neq \{0, 1\}$ and put:

$$f_4 = \begin{cases} f_1 f_2, & \text{if } b_2 = 2 \text{ and } a_2 = 1, \\ f_3 f_2, & \text{if } b_2 = 2 \text{ and } a_2 = 0, \end{cases}$$

$$f_5 = \begin{cases} f_1 f_2, & \text{if } a_2 = 2 \text{ and } b_2 = 1, \\ f_3 f_2, & \text{if } a_2 = 2 \text{ and } b_2 = 0, \end{cases}$$

³We will use only f_1, f_2 and f_3 for which the situation $a_1 = 1, b_1 = 0$ is symmetric.

to get:

x	f_1	f_3	f_2	f_4	f_5
0	0	1	a_2	1	0
1	1	0	a_2	1	0
2	0	0	b_2	0	1

Thus f_3, f_1, f_4 satisfy $P2$ (with 0 and 2 interchanged) or f_3, f_1, f_5 satisfy $P1$.

- Case 2: For each $i \in \{0, 1, 2\}$ we have $i = b_i$.
Without loss of generality we may assume that $a_0 = 1$, so that:

x	f_0	f_1	f_2
0	0	a_1	a_2
1	1	1	a_2
2	1	a_1	2

If $a_2 = 0$ then replacing f_2 by f_0f_2 puts us into Case 1. If $a_2 = 1$ and $a_1 = 2$ then the term operations f_1f_0, f_1, f_2 put us into $P2$ situation (with 0 and 1 interchanged). Finally, if $a_2 = 1$ and $a_1 = 0$ the term operations f_0, f_1, f_1f_2 again put us into $P2$ situation (with 0 and 1 interchanged).

Now we know that \mathbf{A} has 3 term operations f, g, h satisfying either $P1$ or $P2$.

Being in situation $P1$ we use the reduction of POSITIVE-1-IN-3-SAT presented in the proof of Lemma 2.18 to conclude that $\text{SYSTEMSAT}(\mathbf{A})$ is NP-complete.

The reduction of POSITIVE-1-IN-3-SAT in situation $P2$ is only a bit harder. First we are going to show that if we are in $P2$ and not in $P1$ then a constant term operation \top belongs to $\text{Clo}_1(\mathbf{A})$. If $\{\perp, \top\} = \{1, 2\}$ then $ff = \top \in \text{Clo}_1(\mathbf{A})$. Assume $\{\perp, \top\} \neq \{1, 2\}$.

- Case 1: $\top = 0$. In this situation ff, g, h satisfy $P1$.

x	f	g	h	ff
0	\perp	\perp	\perp	0
1	0	0	\perp	\perp
2	0	\perp	0	\perp

- Case 2: $\perp = 0$. Since \mathbf{A} is proper, then there exists a term operation d such that $d(0) = d_0 \neq 0$. We analyze below the term operation fdf :

x	f	g	h	d	df	fdf
0	0	0	0	d_0	d_0	\top
1	\top	\top	0	d_1	d_3	d_4
2	\top	0	\top	d_2	d_3	d_4

If $d_4 = \top$ then $\top = fdf \in Clo_1(\mathbf{A})$. Otherwise $d_4 = 0$, thus fdf, g, h satisfy *P1*.

A formula $F = C_1 \wedge \dots \wedge C_n$ is transformed into equations as follows. For each variable x occurring in F we need additional two variables v_x and x' and 3 equations:

$$x \rightsquigarrow \begin{cases} f(v_x) \approx \top \\ g(v_x) \approx x \\ h(v_x) \approx x' \end{cases}$$

Next, for each clause C_i we need a variable v_i and then C_i is transformed into 3 equations as follows:

$$C_i = x \vee y \vee z \rightsquigarrow \begin{cases} f(v_i) \approx x' \\ g(v_i) \approx y \\ h(v_i) \approx z \end{cases}$$

The equations for variable x force x' to simulate the negation of x . Indeed, because of the equation $f(v_x) \approx \top$ the variable v_x cannot be valuated to 0. If $v_x = 1$ then $x = \top$ and $x' = \perp$, while for $v_x = 2$ we have $x = \perp$ and $x' = \top$. The equations for the clause C_i work as in the proof of Lemma 2.18. Indeed, if $v_i = 0$ then x', y, z take value \perp and thus x take value \top . If v_i takes value 1 or 2 again exactly one of the variables x, y, z takes value \top .

Lemma 2.24 *For a unary algebra \mathbf{A} with three elements the problem SYSTEMSAT(\mathbf{A}) is in \mathbf{P} (in fact it is $O(n^2)$) if $\text{width}(\mathbf{P}(\mathbf{A})) \leq 2$.*

Proof Given an instance I of SYSTEMSAT(\mathbf{A}) we first apply `Simplify` procedure. As a result we get `False`, meaning that there is no solution of I , or an equivalent instance I' satisfying:

- (1) All equations are of the form $f(x) \approx g(y)$, where $x \neq y$, $|f(A)| = 2$ and $f(A) = g(A)$,
- (2) $|C(x)| > 1$ for each $x \in \text{Var}(I')$.

Now we present the algorithm solving such simplified instance I' . We do a reduction into 2-SAT which is known to be polynomial (in fact $O(n^2)$, see e.g. Papadimitriou [Pap94]). We put $V' = \text{Var}(I')$ and T to be the set of generic operations from $Clo_1(\mathbf{A})$. We define set of variables of 2-SAT by $V^* = \{X_t^x : x \in V' \text{ and } t \in T\}$.

Since $|t(A)| = 2$ for $t \in T$, so there is exactly one $c_t \in A$ with $|t^{-1}(t(c_t))| = 1$. This means that for every $a \in A$ we have:

$$t(a) = t(c_t) \Leftrightarrow a = c_t. \quad (\star)$$

With the use of c_t our intended interpretation of 2-SAT variables can be described by:

$$X_t^x \text{ is valuated by } 1 \Leftrightarrow x \text{ is valuated by } c_t.$$

We start our construction of the instance I^* of 2-SAT by transforming each equation of the form $f(x) \approx g(y)$ into two 2-SAT clauses⁴:

$$\begin{aligned} X_f^x &\Leftrightarrow X_g^y, & \text{if } f(c_f) = g(c_g), \\ \text{or} & & \\ X_f^x &\Leftrightarrow \neg X_g^y, & \text{otherwise.} \end{aligned} \quad (\star\star)$$

For term operations $f, g \in T$ and a variable x with $C(x) = A$ we add clauses which code the interaction between 2-SAT variables X_f^x and X_g^x :

$$\begin{aligned} X_f^x &\Rightarrow X_g^x, & \text{if } c_f = c_g, \\ \text{or} & & \\ X_f^x &\Rightarrow \neg X_g^x, & \text{if } c_f \neq c_g. \end{aligned} \quad (\star\star\star)$$

If $C(x) \neq A$ then from (2) we know that $|C(x)| = 2$. We finish our construction of I^* by adding the following clauses for each such variable x and $f, g \in T$:

$$\begin{aligned} &\neg X_f^x, & \text{whenever } c_f \notin C(x), \\ X_f^x &\Leftrightarrow X_g^x, & \text{if } c_f = c_g \text{ and } \{c_f, c_g\} \subseteq C(x), \\ X_f^x &\Leftrightarrow \neg X_g^x, & \text{if } c_f \neq c_g \text{ and } \{c_f, c_g\} \subseteq C(x). \end{aligned} \quad (\star\star\star\star)$$

Take a solution $v : V' \rightarrow A$ of I' . We follow our intended interpretation to define a valuation $s : V^* \rightarrow 2$ by:

$$s(X_t^x) = \begin{cases} 1, & \text{if } v(x) = c_t, \\ 0, & \text{otherwise.} \end{cases}$$

To see that s is a solution of I^* first take a clause $C \in I^*$ (in fact a pair of 2-SAT clauses) generated by $(\star\star)$ for an equation of the form $f(x) \approx g(y)$. We have $f(v(x)) = g(v(y))$ and two cases to consider:

- Case 1: $f(c_f) = g(c_g)$, so that $C = (X_f^x \Leftrightarrow X_g^y)$. Thanks to (\star) the following equivalences hold:

$$\begin{aligned} s(X_f^x) = 1 &\Leftrightarrow v(x) = c_f \Leftrightarrow f(v(x)) = f(c_f) \Leftrightarrow f(v(x)) = g(c_g) \Leftrightarrow \\ &g(v(y)) = g(c_g) \Leftrightarrow v(y) = c_g \Leftrightarrow s(X_g^y) = 1, \end{aligned}$$

as required.

- Case 2: $f(c_f) \neq g(c_g)$, so that $C = (X_f^x \Leftrightarrow \neg X_g^y)$. The proof is very similar to Case 1. Remember that $f(A) = g(A)$ and $|f(A)| = 2$. This time the following equivalences hold:

⁴Obviously $X \Leftrightarrow Y$ is a pair of 2-SAT clauses: $\neg X \vee Y$ and $X \vee \neg Y$.

$$s(X_f^x) = 1 \Leftrightarrow v(x) = c_f \Leftrightarrow f(v(x)) = f(c_f) \Leftrightarrow f(v(x)) \neq g(c_g) \Leftrightarrow \\ g(v(y)) \neq g(c_g) \Leftrightarrow v(y) \neq c_g \Leftrightarrow s(X_g^y) = 0,$$

as required.

The other clauses in I^* were generated either by (★★) or by (★★★★). First choose a variable $x \in V'$ with $C(x) = A$ and take a clause C generated by (★★) for $f, g \in T$ to get:

- Case 1: $c_f = c_g$, so that $C = (X_f^x \Rightarrow X_g^x)$. We have:

$$s(X_f^x) = 1 \Leftrightarrow v(x) = c_f \Rightarrow v(x) = c_g \Leftrightarrow s(X_g^x) = 1,$$

as required.

- Case 2: $c_f \neq c_g$, so that $C = (X_f^x \Rightarrow \neg X_g^x)$. This time we have:

$$s(X_f^x) = 1 \Leftrightarrow v(x) = c_f \Rightarrow v(x) \neq c_g \Leftrightarrow s(X_g^x) = 0,$$

as required.

Finally choose a variable $x \in V'$ with $|C(x)| = 2$. If $c_f \notin C(x)$ for some $f \in T$ then $v(x) \neq c_f$ and thus $s(X_f^x) = 0$ as determined by the first type of clauses from (★★★★). Suppose $\{c_f, c_g\} \subseteq C(x)$ to get:

- Case 1: $c_f = c_g$, so that $X_f^x \Leftrightarrow X_g^x \in I^*$. Similarly to the situation $C(x) = A$ the following equivalences hold:

$$s(X_f^x) = 1 \Leftrightarrow v(x) = c_f \Leftrightarrow v(x) = c_g \Leftrightarrow s(X_g^x) = 1,$$

as required.

- Case 2: $c_f \neq c_g$, so that $X_f^x \Leftrightarrow \neg X_g^x \in I^*$. This time thanks to $|C(x)| = 2$ we have $v(x) = c_f \Leftrightarrow v(x) \neq c_g$ thus:

$$s(X_f^x) = 1 \Leftrightarrow v(x) = c_f \Leftrightarrow v(x) \neq c_g \Leftrightarrow s(X_g^x) = 0,$$

as required.

Conversely, for a boolean valuation $s : V^* \rightarrow 2$ satisfying all clauses from I^* we define a valuation $v : V' \rightarrow A$ by:

$$v(x) = \begin{cases} c_t, & \text{for some } t \in T \text{ such that } s(X_t^x) = 1, & \text{if such } t \text{ exists,} \\ a, & \text{such that } a \neq c_t \text{ for all } t \in T, & \text{otherwise.} \end{cases}$$

To see that v is well defined in the situation when $s(X_t^x) = 0$ for all $t \in T$ first assume $C(x) = A$. Since $\text{width}(\mathbf{P(A)}) \leq 2$, there is $a \in A$ with $a \neq c_t$ for all $t \in T$. Now assume $|C(x)| = 2$. Since s is a solution of I^* , then I^* cannot contain any clause of

the form $X_f^x \Leftrightarrow \neg X_g^x$ generated by (****). Thus $|\{c_t : t \in T \text{ and } c_t \in C(x)\}| \leq 1$, so there is $a \in C(x)$ such that $a \neq c_t$ for all $t \in T$, which we choose as a value for x .

Claim 2.25 *For each $X_t^x \in V^*$ we have $s(X_t^x) = 1 \Leftrightarrow v(x) = c_t$.*

Proof Assume to the contrary that $s(X_t^x) = 1$ and $v(x) \neq c_t$. By definition of v there is $t' \in T$ such that $s(X_{t'}^x) = 1$ and $v(x) = c_{t'} \in C(x)$, so obviously $c_t \neq c_{t'}$. Since $s(X_t^x) = 1$ then $\neg X_t^x \notin I^*$, so $c_t \in C(x)$ by (****). Thus $\{c_t, c_{t'}\} \subseteq C(x)$ and by (***), or (****) we get that the clause $X_{t'}^x \Rightarrow \neg X_t^x$ is in I^* . We get a contradiction with $s(X_t^x) = s(X_{t'}^x) = 1$.

Now take $s(X_t^x) = 0$ to show $v(x) \neq c_t$. First observe that if $c_t \notin C(x)$ then obviously $v(x) \neq c_t$. Now assume $c_t \in C(x)$ to get two possibilities:

- Case 1: $s(X_{t'}^x) = 1$ for some $t' \in T$ and $v(x) = c_{t'} \in C(x)$ thus $\{c_{t'}, c_t\} \subseteq C(x)$. Since the clause $X_{t'}^x \Rightarrow X_t^x$ is not satisfiable then, by (***), or (****), we get that $c_t \neq c_{t'}$ and thus $v(x) \neq c_t$, as required.
- Case 2: $s(X_{t'}^x) = 0$ for all $t' \in T$. By the definition of v we get $v(x) \neq c_t$. \square

To prove that v is a solution of I' take an equation of the form $f(x) \approx g(y)$ from I' .

- Case 1: $f(c_f) = g(c_g)$. Since s is a solution of I^* we get by (**) that $s(X_f^x) = 1 \Leftrightarrow s(X_g^y) = 1$. Thanks to Claim 2.25 and (*) the following equivalences hold:

$$f(v(x)) = f(c_f) \Leftrightarrow v(x) = c_f \Leftrightarrow s(X_f^x) = 1 \Leftrightarrow s(X_g^y) = 1 \Leftrightarrow$$

$$v(y) = c_g \Leftrightarrow g(v(y)) = g(c_g) \Leftrightarrow g(v(y)) = f(c_f).$$

Thus $f(v(x)) = g(v(y))$ as $f(A) = g(A)$ has only two elements.

- Case 2: $f(c_f) \neq g(c_g)$, so that $s(X_f^x) = 1 \Leftrightarrow s(X_g^y) = 0$. The proof is very similar to Case 1. This time the following equivalences hold:

$$f(v(x)) = f(c_f) \Leftrightarrow v(x) = c_f \Leftrightarrow s(X_f^x) = 1 \Leftrightarrow s(X_g^y) = 0 \Leftrightarrow$$

$$v(y) \neq c_g \Leftrightarrow g(v(y)) \neq g(c_g) \Leftrightarrow g(v(y)) = f(c_f).$$

Thus $f(v(x)) = g(v(y))$ as required. \square

Now we are ready to state the main theorem of this section.

Theorem 2.26 [Bro06] *For a unary algebra \mathbf{A} with at most three elements, $\text{SYSTEMSAT}(\mathbf{A})$ is in \mathbf{P} (in fact it is $O(n^2)$) if \mathbf{A} is not proper or $\text{width}(\mathbf{P}(\mathbf{A})) \leq 2$ holds, otherwise it is \mathbf{NP} -complete.*

Proof Note that $\text{width}(\mathbf{P}(\mathbf{A})) = 1$ for any two-element algebra. On the other hand on the two-element set all four unary operations are not generic, thus Lemma 2.19 gives us that $\text{SYSTEMSAT}(\mathbf{A})$ is in \mathbf{P} . For $|A| = 3$ we directly apply Lemmas 2.23 and 2.24. \square

2.3 Width and Complexity

We have seen in Theorem 2.26 that for three-element unary algebras \mathbf{A} the complexity of $\text{SYSTEMSAT}(\mathbf{A})$ is fully characterized by the width of the corresponding preorder $\mathbf{P}(\mathbf{A})$.

Unfortunately this characterization does not extend to larger algebras. We are going to show that width and computational complexity are independent.

Observation 2.27 For the following algebra $\mathbf{A} = (A, F)$ with $F = \{f, g, h, r, s, 1\}$ defined by:

x	f	g	h	r	s	1
0	0	0	0	0	0	1
1	0	0	0	0	0	1
2	0	0	0	0	0	1
3	0	0	0	0	0	1
4	1	2	3	2	1	1
5	2	3	1	1	1	1
6	3	1	2	1	1	1

we have $\text{width}(\mathbf{P}(\mathbf{A})) = 1$ and $\text{SYSTEMSAT}(\mathbf{A})$ is **NP**-complete.

Proof First observe that $\text{Clo}_1(\mathbf{A}) = F \cup \{0, id\}$. Thus $\mathbf{P}(\mathbf{A})$ is a chain $\text{Ker}(id) \subseteq \text{Ker}(f) = \text{Ker}(g) = \text{Ker}(h) \subseteq \text{Ker}(r) \subseteq \text{Ker}(s) \subseteq \text{Ker}(1) = \text{Ker}(0)$, so that $\text{width}(\mathbf{P}(\mathbf{A})) = 1$.

Next consider the following system of term equations over \mathbf{A} :

$$\begin{cases} s(a) \approx 1 \\ f(a) \approx g(b) \\ f(a) \approx h(c) \end{cases}$$

and observe that there are only three possibilities for values of the quadruple $(a, r(a), r(b), r(c))$ in the solutions of the above system:

a	$r(a)$	$r(b)$	$r(c)$
4	2	1	1
5	1	2	1
6	1	1	2

This allows us to make a reduction from **POSITIVE-1-IN-3-SAT** similar to the one presented in Lemma 2.18. A formula $F = C_1 \wedge \dots \wedge C_n$ is transformed into $6n$ equations as follows:

$$C_i = x \vee y \vee z \rightsquigarrow \begin{cases} s(a_i) \approx 1 \\ f(a_i) \approx g(b_i) \\ f(a_i) \approx h(c_i) \\ r(a_i) \approx x \\ r(b_i) \approx y \\ r(c_i) \approx z, \end{cases}$$

where a_i, b_i, c_i are new variables not occurring in F .

It is easy to check that a solution of these $6n$ equations exists if and only if the formula F is satisfiable according to POSITIVE-1-IN-3-SAT rules. If a_i takes the value 4 (5 or 6) then only x (y or z , respectively) is valued by 2, while other two variables from $\{x, y, z\}$ are valued by 1. Transforming 2 and 1 into true and false boolean values we make C_i true according to POSITIVE-1-IN-3-SAT rules. \square

To give examples of algebras \mathbf{A} with arbitrarily large width ($\mathbf{P}(\mathbf{A})$), but for which $\text{SYSTEMSAT}(\mathbf{A})$ is solvable in a polynomial time, we need the following Lemma:

Lemma 2.28 *Let $\mathbf{A} = (A, F)$ be a unary algebra in which there is $a_0 \in A$ such that for all non-constant $f \in F$:*

- (1) $f(a_0) = a_0$.
- (2) $|f^{-1}(a)| \leq 1$ for all $a \neq a_0$.

Then $\text{SYSTEMSAT}(\mathbf{A})$ is in \mathbf{P} .

Proof Given an instance I of $\text{SYSTEMSAT}(\mathbf{A})$ we apply the `Simplify` procedure to get an instance I' of $\text{CSYSTEMSAT}(\mathbf{A})$. If `Simplify` procedure returns `True` then we value all variables in V by a_0 . Observe that by (1) all generic operations $t \in \text{Clo}_1(\mathbf{A})$ satisfy $t(a_0) = a_0$. Since I' has only equations with generic operations, we get that all equations in I' are satisfied. One can check that for all definable constraints C such that $|C| > 1$ we have $a_0 \in C$, so that the above valuation is a solution of the instance I' of $\text{CSYSTEMSAT}(\mathbf{A})$. \square

One consequence of Lemma 2.28 blocks a natural generalization of Theorem 2.26:

Observation 2.29 For each n there exists a proper algebra \mathbf{A}_n such that width ($\mathbf{P}(\mathbf{A}_n)$) = n and $\text{SYSTEMSAT}(\mathbf{A}_n)$ is in \mathbf{P} .

Proof Put $\mathbf{A}_n = (\{0, \dots, n\}, f_1, \dots, f_n, c)$, where:

x	f_1	f_2	f_3	\dots	f_n	c
0	0	0	0	\dots	0	1
1	1	0	0	\dots	0	1
2	0	1	0	\dots	0	1
3	0	0	1	\dots	0	1
\vdots	\vdots	\vdots	\vdots	\ddots	\vdots	\vdots
n	0	0	0	\dots	1	1

The algebra \mathbf{A}_n is proper and satisfies (1) and (2) of Lemma 2.28 with $a_0 = 0$. Therefore $\text{SYSTEMSAT}(\mathbf{A}_n)$ is in \mathbf{P} . On the other hand one can easily check that $\text{width}(\mathbf{P}(\mathbf{A}_n)) = n$. \square

Note however, that the problem $\text{CSYSTEMSAT}(\mathbf{A}_n)$ for $n \geq 3$ is \mathbf{NP} -complete. It is because in CSYSTEMSAT we are allowed to use arbitrarily chosen constraints, e.g. of the form $C(x) = \{1, 2, 3\}$. Such constraint makes possible a reduction similar to the one from Lemma 2.18 (from $\text{POSITIVE-1-IN-3-SAT}$) with additional twist given by putting $C(v_i) = \{1, 2, 3\}$ for each variable v_i . Therefore we introduced the concept of *definable* constraints, which prevents us from such situations when translating instances of SYSTEMSAT into instances of CSYSTEMSAT in the `Simplify` procedure.

From Observation 2.27 we know that width 1 does not suffice to put SYSTEMSAT into \mathbf{P} . However an assumption stronger than width 1, namely that the algebra \mathbf{A} has exactly one non-constant operation is sufficient, as can be seen from the following Lemma:

Lemma 2.30 *For a unary algebra $\mathbf{A} = (A, F)$ with exactly one non-constant operation in F the problem $\text{SYSTEMSAT}(\mathbf{A})$ is in \mathbf{P} .*

Proof Given an instance I of $\text{SYSTEMSAT}(\mathbf{A})$ we apply the `Simplify` procedure to get an instance I' of $\text{CSYSTEMSAT}(\mathbf{A})$. Denote by f the only non-constant operation in F . Since in I' we are left only with equations with generic operations, all of the operations in I' can be expressed by f^i , for some $i > 0$. Now pick a variable $x \in \text{Var}(I')$ and choose minimal m such that $f^m(x)$ can be found somewhere in I' . This means that $i \geq m$ whenever $f^i(x)$ occurs in I' . We replace every $f^i(x)$ with $f^{i-m}(x')$ for a new variable x' to get instance I'' . We also copy constraint function from I' to I'' and then put $C(x') := f^m(C(x))$ in I'' . It is easy to check that I' and I'' are equivalent and equal in size.

Now observe that at least one equation in I'' contains $\text{id}(x')$, since m was chosen to be minimal. Thus we can apply `Simplify` procedure once again, to get instance smaller than I' and repeat the above process. Whenever `Simplify` procedure returns `False` we know that there is no solution of the starting instance I . Otherwise we will end up with empty instance meaning that the solution exists. The number of steps is at most linear (in the size of I), so that the whole algorithm works in a polynomial time. \square

References

- [Bro06] Broniek P (2006) Solving equations over small unary algebras, *Discrete Math Theoret Comput Sci Proc AF*, 49–60
- [FMS04] Feder Tomás, Madelaine Florent, Stewart Iain A (2004) Dichotomies for classes of homomorphism problems involving unary functions. *Theoret Comput Sci* 314(1–2):1–43
- [GJ79] Garey MR, Johnson DS (1979) *Computers and intractability. A guide to the theory of NP-completeness*. W. H. Freeman and Co., San Francisco, California

- [Pap94] Papadimitriou CH (1994) Computational complexity. Addison-Wesley Publishing Company, Reading, MA
- [Sch78] Schaefer TJ (1978) The complexity of satisfiability problems, In: Conference record of the tenth annual ACM symposium on theory of computing (San Diego, California, 1978), ACM, New York, pp. 216–226

Computational Complexity of Solving Equation Systems

Broniek, P.

2015, IX, 64 p. 1 illus., Softcover

ISBN: 978-3-319-21749-9