Equation Satisfiability and Program Satisfiability for Finite Monoids

David Mix Barrington[1], Pierre McKenzie[2], Cris Moore[3], Pascal Tesson[4], and Denis Thérien[4*]

[1] Dept. of Computer Science, University of Massachussets
[2] Dept. d'Informatique et de Recherche Opérationnelle, Université de Montréal
[3] Dept. of Computer Science, University of New Mexico
[4] School of Computer Science, McGill University

Abstract. We study the computational complexity of solving equations and of determining the satisfiability of programs over a fixed finite monoid. We partially answer an open problem of [4] by exhibiting quasi-polynomial time algorithms for a subclass of solvable non-nilpotent groups and relate this question to a natural circuit complexity conjecture. In the special case when M is aperiodic, we show that PROGRAM SATISFIABILITY is in P when the monoid belongs to the variety **DA** and is NP-complete otherwise. In contrast, we give an example of an aperiodic outside **DA** for which EQUATION SATISFIABILITY is computable in polynomial time and discuss the relative complexity of the two problems. We also study the closure properties of classes for which these problems belong to P and the extent to which these fail to form algebraic varieties.

1 Introduction

In [4], Goldmann and Russell investigated the computational complexity of determining if an equation over some fixed finite group has a solution. Formally, an equation over a group, or more generally over a monoid M, is given as

$$c_0 X_{i_1} c_1 \dots c_{n-1} X_{i_n} c_n = m$$

where the $c_i \in M$ are constants, $m \in M$ is the target and the X_i's are variables, not necessarily distinct. The EQUATION SATISFIABILITY problem for M (which we will denote EQN-SAT$_M$) is to determine if there is an assignment to the variables such that the equation is satisfied. It is shown in [4] that this problem is NP-complete for any non-solvable group but lies in P for nilpotent groups. To prove the latter result, they introduced the harder problem of determining whether a given program over a group G outputs some $g \in G$.

* P. McKenzie, P. Tesson and D. Thérien are supported by NSERC and FCAR grants. A part of the work was completed during workshops held respectively by DIMACS-DIMATIA (June 99) and McGill University (February 00). The authors wish to thank the organizers of both events.

We complete and extend their results by considering the computational complexity of both Equation Satisfiability and Program Satisfiability over a fixed monoid M. When M is a group, we show that lower bounds on the length of programs over M that can compute AND yield upper bounds on the time complexity of PROG-SAT and EQN-SAT. In particular, we give a quasi-polynomial time algorithm for a class of solvable, non-nilpotent groups. We also uncover a tight relationship between the complexity of PROG-SAT for groups and the conjecture that any bounded-depth circuit built solely of MOD_m gates requires exponential size to compute AND [2].

When M is an aperiodic monoid, we prove that PROG-SAT is solvable in polynomial time when M belongs to the variety **DA** and is NP-complete otherwise. This is in sharp contrast with the fact that there are aperiodics lying outside **DA** for which EQN-SAT is in P and others for which it is NP-complete.

We also investigate the complexity of #PROG-SAT and show that it is #P-complete for aperiodics outside of **DA** and non-solvable groups, but that it is in #L for monoids in **DA** and not #P-complete under polynomial-time transformations for a subclass of solvable groups, including nilpotent groups.

Finally, we discuss the relative complexity of PROG-SAT and EQN-SAT and the closure properties of classes of monoids for which these problems are in P. Section 2 outlines the necessary background on finite monoids and programs over monoids. Sections 3 and 4 describe our results for groups and aperiodic monoids respectively. Finally, the differences among the two problems are explored in Section 5 and open questions are presented in the conclusion.

2 Preliminaries

2.1 Finite Monoids

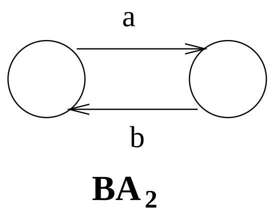
A monoid M is a set with a binary associative operation and an identity. This operation defines a canonical surjective morphism $eval_M : M^* \to M$ by $eval_M(m_1 m_2 \dots m_k) = m_1 \cdot m_2 \cdots m_k$ which just maps a sequence of monoid elements to their product in M. In this paper, we will only be considering finite monoids, with the exception of the free monoid A^*.

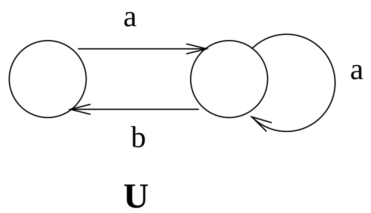
We say that a monoid N divides monoid M and write $N \prec M$ if N is the homomorphic image of a submonoid of M. A class of finite monoids $\mathcal{M}$ is said to be a (pseudo-)variety if it is closed under direct product and division. Varieties are the natural way of classifying finite monoids.

A monoid M is aperiodic or group-free if no subset of it forms a non-trivial group or, equivalently, if it satisfies $m^{t+1} = m^t$ for some integer t and all $m \in M$. We will also be interested in the subvariety of aperiodics called **DA**. The monoids in **DA** are the aperiodics satisfying $(stu)^n t (stu)^n = (stu)^n$, for some $n \geq 0$. It is known that M belongs to **DA** iff for all $F \subseteq M$ the set $\{w \in M^* : eval_M(w) = m \in F\}$ can be expressed as the disjoint union of languages of the form $L = A_0^* a_1 A_1^* a_2 \dots a_k A_k^*$, where $a_i \in M$ and $A_i \subseteq M$, and this concatenation is unambiguous, meaning that if $w \in L$ then there is a unique way of factoring w

as $w_0 a_1 w_1 \dots a_k w_k$ with $w_i \in A_i^*$ [7]. This variety is omnipresent in investigations of complexity-related questions on finite monoids.

We will also be particularly concerned with two aperiodic monoids BA_2 and U that do not belong to **DA**. They are the transition monoids associated with the following partial automata (missing arcs map to an implicit sink state):





Both BA_2 and U have 6 elements (namely $\{1, a, b, ab, ba, 0\}$) but in BA_2 we have $a^2 = b^2 = 0$ while in U we have $b^2 = 0$ but $a^2 = a$. Note also that for both monoids we have $bab = b$ and $aba = a$. It is known that for any finite aperiodic monoid M that does not belong to **DA** either $U \prec M$ or $BA_2 \prec M$. These two monoids are thus minimal examples of aperiodics outside **DA**.

The wreath product of two monoids M, N, denoted $M \circ N$, is the set $M^N \times N$ with an operation defined as $(f_1, n_1) \cdot (f_2, n_2) = (f_1 f', n_1 n_2)$ where $f' : N \to M$ is defined as $f'(x) = f_2(x n_1)$. Note that the product $x n_1$ is well defined since $x \in N$. A group G is solvable iff it divides the wreath product of cyclic groups. (A proof of this fact and a detailed introduction to finite monoids and algebraic automata theory can be found e.g. in [3].)

2.2 Programs over Monoids

The formalism of programs over finite monoids (or non-uniform DFA's as they were first called) was introduced by Barrington [1] to show the equivalence of NC^1 and bounded-width branching programs of polynomial length.

An n-input program over M is a sequence of instructions

$$\phi = (i_1, s_1, t_1)(i_2, s_2, t_2) \dots (i_l, s_l, t_l)$$

where the $i_j \in [n]$ are bit positions in the input and $s_j, t_j \in M$. Given an input $x \in \{0,1\}^n$, the program ϕ outputs the string $\phi(x) = w_1 w_2 \dots w_l$ where $w_j = s_j$ if the i_j^{th} bit of the input is 0 and $w_j = t_j$ if this bit is 1. We say that a language $L \subseteq \{0,1\}^n$ is recognized by this program if there exists $F \subseteq M$ such that x is in L iff $eval_M(\phi(x))$ is in F.

There is a deep connection between such programs and subclasses of NC^1. In particular, a language is in NC^1 iff it can be recognized by a polynomial length program over a finite monoid. Similarly AC^0 is exactly the class of languages recognizable by polynomial length programs over aperiodic monoids and CC^0 circuits (polynomial size bounded depth circuits using only MOD_m gates) correspond to programs of polynomial length over solvable groups. A survey of such results can be found in [5].

Given an n-input program ϕ over a monoid M and a target set $F \subseteq M$, the PROGRAM SATISFIABILITY problem (denoted PROG-SAT$_M$) is to determine whether there exists $x \in \{0,1\}^n$ such that $\phi(x)$ evaluates to $m \in F$. We will also study the corresponding counting problem #PROG-SAT.

Determining the satisfiability of a program over M is always at least as hard as determining the satisfiability of equations over the same monoid.

Lemma 1. *For any finite monoid M, EQN-SAT$_M \leq_P$ PROG-SAT$_M$.*

Proof. Suppose the equation $c_0 X_{i_1} c_1 \ldots X_{i_n} c_n = m$ has t variables. It can be satisfied iff the following M-program over $t \cdot |M|$ variables can be satisfied: we replace every constant c_i by the instruction $(1, c_i, c_i)$ and each occurrence of the variable X_i by a sequence of $|M|$ instructions querying variables $Y_{j_1}, \ldots, Y_{j_{|M|}}$ of the form $(Y_{j_1}, 1, m_1)(Y_{j_2}, 1, m_2) \ldots (Y_{j_{|M|}}, 1, m_{|M|})$ where $m_1 \ldots m_{|M|}$ are the elements of M. Note that for any $m \in M$, there is an assignment of the Y_j's such that this sequence of $|M|$ instructions evaluates to m.

As we will see, the converse of this fact is not true and we will highlight the differences in the complexity of these two similar problems.

Note that throughout these notes, we (shamelessly) assume that $P \neq NP$.

3 Groups

It was shown by Goldmann and Russell in [4] that EQN-SAT$_G$ is NP-complete for any non-solvable group G. They also proved that both EQN-SAT and PROG-SAT can be solved in polynomial time for a nilpotent group. Using Lemma 1, we thus know that PROG-SAT$_G$ is also NP-complete for non-solvable groups. It is much easier, however, to prove this directly using well known properties of programs over these groups.

Theorem 1. *For any non-solvable group G, we have PROG-SAT$_G$ is NP-complete and #PROG-SAT$_G$ is #P-complete.*

This follows from the fact [1] that given a boolean formula $\psi(X_1, X_2, \ldots X_n)$, we can, for any non-solvable G build an n-input program $\phi(X_1, X_2, \ldots X_n)$ whose length is polynomially related to the length of ψ and such that $\phi(b_1, b_2, \ldots b_n) = 1$ iff the truth assignment $(b_1, b_2, \ldots b_n)$ satisfies ψ. We can thus transform SAT and #SAT to PROG-SAT$_G$ and #PROG-SAT$_G$ respectively.

The complexity of both EQN-SAT and PROG-SAT for solvable non-nilpotent groups is closely tied to whether or not there exist programs over these groups computing the AND function in sub-exponential length. We will say that a finite group G is AND-strong if there exists a family of polynomial length programs over G computing AND. We will say that G is AND-weak if programs over G computing the AND of t variables require length $2^{\Omega(t)}$.

As we have mentioned, all non-solvable groups are provably AND-strong, but the only groups known to be AND-weak are groups which divide the wreath product of a p-group by an abelian group. In particular, these include S_3 and A_4. It has been conjectured, however, that all solvable groups are AND-weak or, equivalently, that CC^0 circuits require exponential size to compute AND (see [2] for a detailed discussion).

Theorem 2. *If G is AND-weak then PROG-SAT$_G$ is solvable in quasi-polynomial time.*

Proof. We claim that if a program in s variables over G can be satisfied, then it can be satisfied by an assignment of weight logarithmic in the length of the program. Suppose not. We define the weight $|x|_1$ of $x \in \{0,1\}^*$ as the number of 1's in x. Let w be a satisfying assignment of minimal weight, with $|w|_1 = k$. Assume W.L.O.G. that the first k bits of w are 1. By fixing $X_{k+1}, \dots, X_s$ to 0, we obtain a program over G which is computing the AND of k bits because w was assumed to have minimal weight. Since G is AND-weak, we must have $n \geq 2^{\Omega(k)}$, so $k \leq O(\log n)$. It is thus sufficient to consider only the $O(\binom{n}{O(\log n)}) = O(n^{O(\log n)})$ assignments of weight at most k, so we have a quasi-polynomial time algorithm.

Similarly, PROG-SAT$_G$ can be shown [4] to be in P for a nilpotent group G by using the fact [2,6] that a satisfiable program over a nilpotent group can be satisfied by an assignment of weight bounded by a constant independent of the program's length.

Since we do not know that all solvable groups are AND-weak, we would like to give results on the complexity of EQN-SAT and PROG-SAT for solvable groups which are unrelated to their computational power. However, the following theorem shows how closely tied these two questions really are.

Theorem 3. *If G is AND-strong, then PROG-SAT is NP-complete for the wreath product $G \circ Z_k$ for any cyclic group Z_k of order $k \geq 4$.*

Proof. We want to build a reduction from 3-SAT. Define the function f_{g_0,g_1} : $Z_k \to G$ as

$$f_{g_0,g_1}(x) = \begin{cases} g_0 & \text{if } x = 0 \\ g_1 & \text{if } x \neq 0 \end{cases}$$

Also, denote by id the function such that $id(x) = 1_G$ for all $x \in Z_k$. Consider now the following 3-input program over $G \circ Z_k$

$$\phi = (1,(id,0),(id,1))\ (2,(id,0),(id,1))\ (3,(id,0),(id,1))\ (\mathbf{1},(\mathbf{f_{g_0,g_1}},\mathbf{0}),(\mathbf{f_{g_0,g_1}},\mathbf{0}))$$
$$(1,(id,0),(id,-1))\ (2,(id,0),(id,-1))\ (3,(id,0),(id,-1))$$

First note that the Z_k component of ϕ's output will always be 0. Note also that the middle instruction's output is independent of the value of the bit queried. It is also the only instruction affecting the G^{Z_k} component of the output. This component is a function f such that $f(0) = g_1$ if one of the input bits is on and $f(0) = g_0$ otherwise. To see this note that when we execute the middle instruction, the product in Z_k so far is not equal to zero iff one of the instructions yielded a $+1$. Thus, ϕ is recognizing the OR of these three variables.

Suppose the 3-SAT instance has m clauses. By assumption there is a G-program of length m^c that computes the AND of m variables. If we replace every instruction (i, g_0, g_1) by a program over $G \circ Z_k$ as above, we obtain a program of length $7 \cdot m^c$ which is satisfiable iff the 3-SAT instance is satisfiable.

The wreath product of a solvable group by an abelian group is also solvable and, as we can see from the proof of Theorem 2, super-polynomial lower bounds on the length of programs recognizing the AND over a group G translate into sub-exponential upper bounds on the time complexity of PROG-SAT$_G$. Thus, assuming that no sub-exponential time algorithm can solve an NP-hard problem, there exists an AND-strong solvable group iff there exists a solvable group for which PROG-SAT is NP-complete.

We have seen that #PROG-SAT is #P-complete for non-solvable groups. In contrast, that problem is not #P-complete under parsimonious reductions for any nilpotent group. Indeed, a program over a nilpotent group is either unsatisfiable or is satisfied by a constant fraction of the inputs [6]. This immediately disqualifies it as a #P-complete problem since most problems in #P fail to have this property. Similarly, a polynomial length program over an AND-weak group can not, by definition, have only a single satisfying assignment. #PROG-SAT is thus not #P-complete under parsimonious reductions for these groups either.

For an abelian group A, counting the number of satisfying assignments to a program is in #L. The #L machine only has to guess the assignment one bit at a time. Then, scanning through the program ϕ it can calculate the effect of this choice on the output of ϕ using commutativity. Similar ideas do not seem to yield good algorithms for #PROG-SAT for non-abelian groups.

4 Aperiodic Monoids

In the case of groups, we have no evidence of any significant difference between the complexity of solving equations versus that of satisfying programs. The situation is remarkably different in the case of aperiodic monoids. In this section, we characterize the subclass of aperiodics for which PROG-SAT can be solved in polynomial time but show that there are aperiodics for which PROG-SAT is NP-complete but EQN-SAT belongs to P. We are also able to make explicit the link between the algebraic properties of an aperiodic and the complexity of the corresponding #PROG-SAT problem.

Theorem 4. *For any monoid M in the variety* **DA**, *PROG-SAT$_M$ is in P.*

Proof. As we mentioned, for any monoid M in **DA**, the sets $\{w : eval_M(w) = m \in F \text{ for some } F \subseteq M\}$ can always be expressed as the finite disjoint union of languages of the form $A_0^* a_1 A_1^* \dots a_k A_k^*$, where $a_i \in M$, $A_i \subseteq M$, and the concatenation is unambiguous.

Hence it is sufficient to consider the at most $\binom{l}{k}$ k-tuples of instructions of the program of length l that could be held responsible for the presence of the subword $a_1 a_2 \dots a_k$. For each of them, we need to check if there is an assignment such that the output of the program belongs to $A_0^* a_1 A_1^* \dots a_k A_k^*$ and that can clearly be done in linear time.

Theorem 5. *For any monoid M in the variety* **DA**, *#PROG-SAT$_M$ is in #L.*

Proof. The idea is similar. The #L machine will guess which k-tuple of instructions yields the bookmarks $a_1, \dots, a_k$. It then non-deterministically guesses the value of each of the n variables and checks that this never makes an instruction sitting between a_i and a_{i+1} output an element outside A_i. Only $(k+1)$ indices are kept in memory throughout the computation. Note that we will not count anything twice since the concatenation $A_0^* a_1 A_1^* \dots a_k A_k^*$ is unambiguous.

Let us look now at the two minimal examples of aperiodic monoids that do not belong to **DA**.

Theorem 6. *EQN-SAT for BA_2 is NP-complete.*

Proof. We use a reduction from EXACTLY-ONE-IN-3 SAT. Each variable v_i in the 3-SAT instance is represented by two variables v_i^+ and v_i^- representing v_i and its complement in the equation. We build the following equation with target ab. First, we concatenate, for each i the segments $abv_i^+ v_i^- bv_i^- v_i^+ b$ and for each clause e.g. $(v_i, \overline{v_j}, v_k)$ we concatenate $abv_i^+ v_j^- v_k^+ b$.

It is easy to see that the first half of the equation forces us to choose one of v_i^+, v_i^- as 1 and the other as a. If we now interpret a as TRUE and 1 as FALSE, the equation is satisfiable iff we can choose assignments to the v_i such that in every segment e.g. $abv_i^+ v_j^- v_k^+ b$ exactly one of the variables is set to a.

A similar proof will provide the same result for a larger class of aperiodics outside **DA**. At this point, one might even be tempted to conjecture that EQN-SAT is NP-complete for any aperiodic outside of **DA**, but this is not the case:

Theorem 7. *EQN-SAT for U can be decided in polynomial time.*

Proof. To show this we will use the fact that, in U, $axa = a$ whenever $x \neq 0$. Intuitively, we use the fact that a's are our friends. In particular, we have that if $xyz = a$ then $xaz = a$.

We will show that for any target, if the equation is satisfiable then it can be satisfied by an assignment with a very precise structure. We are given the expression $E : c_0 X_{i_1} c_1 \dots X_{i_n} c_n$ and a target m.

If $m = 0$, the equation is trivially satisfiable by setting any variable to 0, and if $m = 1$, it is satisfiable iff all the c_i's are 1. Since the equation is 0 if any of the c_i is 0, we will assume that the constants are non-zero.

If $m = a$, then E is satisfiable iff it is satisfied when all the variables are set to a, namely when we have both $c_0 \in \{1, a, ab\}$ and $c_n \in \{1, a, ba\}$.

If $m = ba$, and E can be satisfied, then it can be satisfied by one of the $O(n)$ assignments of the following form: all the variables occurring before some point j in the equation (which might be a constant or a variable) are set to 1, the variable at point j is set to ba and the other variables are set to a. To see this, consider any satisfying assignment to E. If the first b in the induced string comes from a constant, then all the variables occurring before it must have been set to 1. Moreover, all we have to insure now is that there are no consecutive b's in the suffix. So we can set all the variables that haven't yet occurred to a without affecting the target. If the first b occurs in a variable, the same reasoning shows that we can set this variable to ba and the variables not yet considered to a. So it is sufficient to check a linear number of possible assignments to decide satisfiability. The case $m = ab$ is handled in a similar, symmetrical way.

Finally if $m = b$, it suffices to consider the following $O(n^2)$ assignments: variables occurring before some j or after some k are set to the identity, the variable at point j is set to ba or b, the one at point k to ab or b, and all remaining variables are set to a. Again, if we now consider any satisfying assignment and call j and k the first and last occurrence of b in the induced word over M^*, then we know that all variables occurring before j or after k were set to 1. The variable or constant at point j must be b or ba, the one at point k being b or ab so we still have a satisfying assignment if we set the rest of the variables to a.

The example of U also proves that the complexity of EQN-SAT and PROG-SAT can be radically different since we also have:

Theorem 8. *PROG-SAT$_U$ is NP-complete and #PROG-SAT$_U$ is #P-complete.*

Proof. The program $\phi_i = (X_{i_1}, b, ba)(X_{i_2}, 1, a)(X_{i_3}, 1, a) \dots (X_{i_k}, ba, a)$ (over U) outputs ba if one of the X_{i_j}'s is set to 1 and 0 otherwise. By concatenating such ϕ_i's we get a program whose output is ba if all ϕ_i's have one of their input variables set to 1 and 0 otherwise. So we can simulate a CNF formula and obtain a reduction from SAT. It is clear that this reduction is parsimonious, hence the #P-completeness result.

From Theorem 6, we know that PROG-SAT$_{BA_2}$ is also NP-complete. It is clear that one can get a parsimonious reduction from EXACTLY-ONE-IN-3 SAT to PROG-SAT$_{BA_2}$ using an idea similar to that of the previous proof. #PROG-SAT$_{BA_2}$ is thus also #P-complete.

As we shall see in the next section, the NP and #P-completeness results for U and BA_2 yield completeness results for all aperiodic monoids outside **DA** .

5 Closure Properties and EQN-SAT vs PROG-SAT

One would expect that when EQN-SAT (or PROG-SAT) is easy to solve for monoid M, N, the problem is also easy for submonoids and homomorphic images of M or N as well as for $M \times N$. However, the previous section provides us with a counterexample to this intuition: EQN-SAT$_{U \times U}$ is in P but BA_2 is a submonoid of $U \times U$ with EQN-SAT$_{BA_2}$ NP-complete.

In a perfect world, the classes of monoids for which EQN-SAT and PROG-SAT belong to P would form varieties. Although this is not the case, at least for equations, these classes do have interesting closure properties which further highlight the slight difference in the nature of the two problems.

Theorem 9. *The class $\mathcal{M}_E = \{M : EQN\text{-}SAT_M \in P\}$ is closed under factors (homomorphic images) and direct products.*

In contrast, the class $\mathcal{M}_P = \{M : PROG\text{-}SAT_M \in P\}$ is closed under factors (homomorphic images) and formation of submonoids.

Proof. $\mathcal{M}_E$ is closed under finite direct products since an equation over the product $M \times N$ can simply be regarded as a pair of independent equations for which satisfiability can be checked individually.

For closure under factors, suppose we are given an equation E over the factor N of monoid M. We can build an equation E' over M by replacing each constant by an arbitrary pre-image. If for *any* pre-image of the target we can find an assignment over M satisfying E' then we can map this to a satisfying assignment over N for E. Moreover, if E is satisfiable, then E' must be satisfiable for some pre-image of our original target and there are finitely many pre-images to try.

$\mathcal{M}_P$ is also closed under factors using a similar argument. We simply have to choose representatives for the pre-images of any element of the smaller monoid. Then the program over the larger monoid can be satisfied for at least one pre-image of the original target iff the original program was satisfiable.

Finally, $\mathcal{M}_P$ is closed under submonoids in a trivial way. Note that instructions of an instance of PROG-SAT over the submonoid produce, by definition, elements of the submonoid. The target of this instance is also in the submonoid. So if we can decide satisfiability over the larger monoid, we can naturally use this to decide satisfiability over the submonoid as well.

This theorem justifies our claim of Section 4 that PROG-SAT is NP-complete for all aperiodics outside of **DA**. Indeed, PROG-SAT is NP-complete for both BA_2 and U and any aperiodic outside **DA** has one of them as a divisor. Since the intersection of $\mathcal{M}_P$ with aperiodic monoids is thus **DA**, a variety, it is closed under direct products. However, we do not know how to prove this fact directly.

Intuitively, $\mathcal{M}_E$ is not closed under submonoids because we have no mechanism to guarantee that the variables are only assigned values in the submonoid. The proofs in [4] used the notion of "inducible" subgroups, i.e. subgroups for which such a mechanism does exist. Our results seem to show that this is a necessary evil.

On the other hand, PROG-SAT is easy for submonoids but could be hard for direct products. We can certainly view a program over $M \times N$ as a pair of programs on M and N respectively which are both satisfiable if the original program is, but, conversely, there is no obvious way to check whether the sets of satisfying assignments for each of them are disjoint or not. Thus, polynomial time algorithms for PROG-SAT$_M$ and PROG-SAT$_N$ do not seem to help us get an algorithm for PROG-SAT$_{M \times N}$, however we have the following partial result:

Theorem 10. *Suppose N is such that $PROG\text{-}SAT_N$ is in P, then for any $M \in$* **DA** *, $PROG\text{-}SAT_{M \times N}$ is also in P.*

Proof. Let us first look only at the M-component of the program. As in the proof of Theorem 4, we can consider each of the polynomially many k-tuples of instructions that can give rise to the bookmarks $a_1, a_2, \ldots a_k$. For any variable X_i, we can now check if setting it to 0 or 1 causes some instructions to throw us out of the unambiguous concatenation $L = A_0^* a_1 A_1^* \ldots a_k A_k^*$. This will force an assignment on some of the variables (or even prove that the target is unreachable given this k-tuple) and leave others free. Given these restrictions, we now look at the program over N obtained after applying these restrictions. By assumption, satisfiability for this program can be checked in P.

The respective closure properties of $\mathcal{M}_E$ and $\mathcal{M}_P$ indicate that beyond their apparent similarities, EQN-SAT and PROG-SAT present different computational challenges.

6 Conclusion

We have shown how the algebraic properties of finite monoids can influence the complexity of solving equations and satisfying programs over them. A few interesting questions still have to be answered. The first is the complexity of PROG-SAT for groups that are solvable but not nilpotent. Although fully resolving this would seem to require new circuit complexity results, we might still be able to use the lower bounds for AND-weak groups to get polynomial-time algorithms. We conjecture that this is not possible, but it would be interesting to find convincing evidence to support this. It is possible that finding good upper bounds for #PROG-SAT for solvable and nilpotent groups could help in that respect. Because we have to count the number of solutions, we would probably need to move away from the brute force approach used in our algorithms.

We would like to thank Mikael Goldmann and Alex Russell for helpful discussions.

References

1. D. A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comput. Syst. Sci.*, 38(1):150–164, Feb. 1989.
2. D. A. M. Barrington, H. Straubing, and D. Thérien. Non-uniform automata over groups. *Information and Computation*, 89(2):109–132, Dec. 1990.
3. S. Eilenberg. *Automata, Languages and Machines.* Academic Press, 1976.
4. M. Goldmann and A. Russell. The complexity of solving equations over finite groups. In *Proceedings of the 14th Annual IEEE Conference on Computational Complexity (CCC-99)*, pages 80–86, 1999.
5. P. McKenzie, P. Péladeau, and D. Thérien. NC^1: The automata theoretic viewpoint. *Computational Complexity*, 1:330–359, 1991.
6. P. Péladeau and D. Thérien. Sur les langages reconnus par des groupes nilpotents. *Compte-rendus de l'Académie des Sciences de Paris*, pages 93–95, 1988.
7. J.-É. Pin, H. Straubing, and D. Thérien. Locally trivial categories and unambiguous concatenation. *J. Pure Applied Algebra*, 52:297–311, 1988.