*Electronic Notes in Theoretical Computer Science 50 No. 4 (2001) { Proc. VEPAS 2001 URL:* [*http://www.elsevier.nl/locate/entcs/volume50.html*](http://www.elsevier.nl/locate/entcs/volume50.html) *15 pages*

*Analyzing Automata with Presburger Arithmetic and Uninterpreted Function Symbols*

*Vlad Rusu 1 IRISA / INRIA Rennes,*

*Campus de Beaulieu, Rennes, France*

*Elena Zinovieva 2;3*

*IRISA / INRIA Rennes, Campus de Beaulieu, Rennes, France*

*Abstract*

*We study a class of extended automata de ned by guarded commands over Pres- burger arithmetic with uninterpreted functions. On the theoretical side, we show that the bounded reachability problem is decidable in this model. On the practical side, the class is useful for modeling programs with unbounded data structures, and the reachability procedure can be used for symbolic simulation, testing, and veri cation.*

## *1 Introduction*

*Modern research in automated veri cation can be divided into two categories.* The rst category, which includes languages and tools such as Spin [12], smv

*[3] and cadp [8], focuses on using well-established techniques such as state* enumeration or binary decision diagrams, and pushing them to the limits by optimizing the algorithms in every possible way.

*The second category deals with exploration of new techniques, whose im-* pact for practical applications, although in quantitative progress, still has to be assessed. Two representative examples for this category are Presburger

*1 Email:* [*rusu@irisa.fr*](mailto:rusu@irisa.fr)

*2 Email:* [*lenaz@irisa.fr*](mailto:lenaz@irisa.fr)

*3 Supported in part by DYADE action FormalCard, a joint project of INRIA and Bull/CP-8*

*c 2001 Published by Elsevier Science B. V. Open access under* [*CC BY-NC-ND license.*](http://creativecommons.org/licenses/by-nc-nd/3.0/)

*arithmetic [19] and the theory of uninterpreted functions with equality [1].* These relatively rich theories allow more accurate modeling of real-life systems, avoiding the rough simpli cations imposed by nite-state methods. The theo- ries are decidable (although their combination is not) and are eÆciently imple- mented in state-of-the-art veri cation tools such as pvs [18] and Omega [15]. Motivated by these observations, this paper investigates a class of extended automata that consist of transitions over a nite control structure, with guards and assignments in a decidable fragment of the theory of Presburger arithmetic with uninterpreted functions. The formalism is expressive (e.g., one transi- tion can modify an unbounded number of function values). It can model quite naturally programs with unbounded data structures such as parametric- sized arrays, or communication protocols with unbounded channels carrying

*unbounded data.*

*We show that reachability in a bounded number of steps is decidable in* this model. The result gives a direct procedure for symbolic simulation. The procedure computes the weakest constraints on a program's input data for a given nite sequence of transitions to be executed. Thus, we obtain an automatic solution to the test input problem [16] from software testing: given a nite path in a program, nd whether it is executable, and if this is the case, obtain input data for executing it. Then, given a coverage criterion [21] de ned by a nite set of nite paths in the program (e.g., executing every instruction at least once, or checking each condition at least once), it is possible, using our procedure, to select the paths that are executable and to synthesize input data for executing them. This results in a complete coverage for the chosen coverage criterion. Finally, symbolic analysis can also be used as a semi- decision procedure for veri cation of safety properties: if a safety property does not hold, the procedure will detect this in a nite number of steps.

*The rest of the paper is organized as follows. In Section 2 we present the* theory of Presburger arithmetic with uninterpreted function symbols. In Sec- tion 3 we de ne a class of extended automata (called pf-automata) with guards and assignments in a decidable fragment of this theory. The initial condition may constrain an unbounded number of function values, and assignments may modify an unbounded number of such values. In Section 4 we present a de- cision procedure for the bounded reachability problem in pf-automata, based on symbolic analysis techniques. The procedure is implemented using the ics decision procedure package [9] from SRI International. Section 5 presents conclusions, related work, and future work.

## *2 Background*

*In this section, we brie y review the theory of Presburger arithmetic and its* extension with uninterpreted function symbols.

*2.1 Presburger Arithmetic*

*Let Z denote the set of integers, and V be a set of integer variables. A term* is a nite, aÆne combination on the variables. An inequality is a comparison (<; >; ; ; =) between terms. A quanti er-free Presburger formula is a nite Boolean combination of inequalities. A Presburger formula is a nite Boolean combination of inequalities, in which some variables can be quanti ed. Thus, if x; y; z; u are variables, then x + 2y + 1 is a term, x y ^ x + 2y +1 > 0 is a quanti er-free Presburger formula, and 8z:x y + z ^ 9u:x + 2y + u > y is a Presburger formula. Satis ability in Presburger arithmetic is decidable [19], with time complexity triple exponential in the size of the formula [17] (but simple exponential in the quanti er-free fragment [2]).

*2.2 Presburger Arithmetic with Uninterpreted Function Symbols*

*Again, let V be a set of integer variables, and F be a set of function symbols.* For each function f 2 F , we only know its arity (a natural number n), and the fact that it is a function from Zn to Z. A term with function symbols is either a variable, or an application of a function symbol to n terms, or an aÆne combination of those. Inequalities (resp. quanti er-free Presburger formulas, resp. Presburger formulas) with function symbols are de ned as in Section 2.1, except that they are built over terms with function symbols. Quanti cation over function symbols is not allowed. Thus, if x; y; z; u are variables and f; g are functions of arity one, x + 2y + f (g(z)) is a term with function symbols, x y ^ x + 2y + f (g(z)) > 0 is a quanti er-free Presburger formula with function symbols, and 8x:x y+f (g(z))^9u:x+2y+f (z) > g(u) is a Presburger formula with function symbols. We denote by pf the theory of Presburger arithmetic with function symbols. Satis ability in pf is 1-

*1*

*complete [13].*

*2.3 Decidable Fragments*

*Satis ability is decidable in the quanti er-free fragment of pf [23]. In this* section we build on this result to de ne a larger decidable fragment.

*2.3.1 Decidability of the quanti er-free fragment.*

*The result [23] is based on a simple observation. In a quanti er-free formula* with uninterpreted function symbols, the only relevant property about func- tions is that they map equals to equals, and, by instantiating this property to nitely many terms, it is possible to obtain an equivalent Presburger arith- metic formula. Let ' be a formula of the quanti er-free fragment of pf. For simplicity, we suppose that in ' there is only one unary function symbol f , which is only applied to two terms, t1 and t2. Then, ' is satis able if and only

*if the following Presburger formula is satis able:*

*'0 : '[f (t1)=f1;f (t2)=f2] ^ (t1 = t2 f1 = f2) (1)*

*That is, in '0, the function applications f (ti) from ' are replaced by new* integer variables fi, and the general property that function f maps equals to equals is instantiated to \equality of the terms ti implies equality of the variables fi". Indeed, a model for ' trivially induces a model for '0 by choosing f1 = f (t1) and f2 = f (t2). Conversely, if there exists a model for '0, then a model for ' can be obtained by choosing the value for f such that f (t1) = f1, f (t2) = f2, and by letting f (i) be an arbitrary value at any other position than t1; t2.

*2.3.2 Decidability of the existential fragment.*

*Consider now the existential fragment of pf (i.e., only existential quanti ers* are allowed, and under the scope of an even number of negations). Modulo variable renaming, it is possible to move all quanti ers to the outermost level. Then, a formula 9x:' has a model if and only if ' also has one. Indeed, if there exist values of the functions and free variables that satisfy ', then the same values satisfy 9x:', and if there exists a model for 9x:', then this model, augmented with the \witness" value of x for the existential quanti er, is a model for '. Thus, the fragment is decidable. This reasoning can be pushed further: a pf formula ' is satis able if and only if 9f:', where f is a function, is satis able. Note that such a formula is not in pf. We use this result in Section 4.

*2.3.3 Decidability of the semi-universal fragment.*

*The universal fragment of pf consists of formulas in which only universal* quanti ers are allowed, under an even number of negations. This fragment is highly undecidable 4 . Nevertheless, universal formulas are useful: for ex- ample, specifying a property of all the elements of a parametric-sized vector requires a universal quanti er. In the remainder of this section we de ne a class of pf formulas with universal quanti ers for which satis ability is de- cidable. This result will be used in the subsequent sections, where we de ne extended automata with guards and assignments in the class.

*De nition 2.1 A shielded pf formula is a pf formula of the form : 8i: 0* where 0 is a quanti er-free formula with the property that function symbols f may only appear within terms of the form f (i). *2*

*That is, the formula 8i:(f (i) > j j > g(i)) is shielded, but 8i:(f (i + 1) = 0)* and 8i:(f (i) = f (y)) are not.

*Also, let the function depth of a formula be the deepest nesting of* function symbols in . That is, both formulas above have function depth *1.*

*4 This can be shown, e.g., by encoding the recurrence problem for 2-counter automata.*

*De nition 2.2 A pf formula # is semi-universal if it is a conjunction # :* ' ^ 1 ^ ::: ^ k of function depth 1, where ' is quanti er-free, every j (j = 1;::: ; k) is shielded, and the following semantic property holds: for every model M' of ' there exists a model M of : 1 ^ ::: ^ k such that M', M agree on the values of the free variables that occur in both '; . 2

*Example 2.3 Consider the following formula*

*# : f (x + 1) f (y)+ 1 ^ 8i:(f (i) = y + 1)*

# *| {'z } | { z }*

*It is a semi-universal formula, as it satis es the syntactic conditions of Def-* inition 2.2, i.e. has function depth 1, ' is quanti er-free, and is shielded. For the semantic condition, note that for every model M' (e.g., Equation 2) there exists also a model M (e.g., Equation 3) such that M', M agree on the value of y, i.e., the free variable that occurs in both ' and .

*M' : fx = 0;y = 2;f (1) = 3;f (2) = 3; and for all k 2= f1; 2g : f (k) = 0g*

*(2)*

*M : fy = 2; for all k : f (k) = 3g (3)* 2

*De nition 2.2 may be hard to use in practice because of the semantic* property that is involved. However, it turns out to be the exact the de nition we need in the following sections. A suÆcient (and easier to check) condition for a formula to be semi-universal is presented at the end of this section.

*Lemma 2.4 Satis ability in the class of semi-universal formulas is decidable.*

*Proof. Let # be a semi-universal formula. For simplicity, we assume without* loss of generality that the quanti ed part of # consists only of one conjunct. Thus,

*# : ' ^ 8i: 1(i; f1(i);::: ; fn(i))*

# *| { z }*

*(4)*

*where f1(i);::: ; fn(i) are all the function applications in . Similarly, let* f (t1);::: ;f (tm) be all the function applications in '. Then, # can be equiv- alently written as

*#0 : ' ^ ^ 1(i; f1(i);::: ; fn(i)) ^ 8i 2= ft1;::: ; tmg: 1(i; f1(i);::: ; fn(i))*

*i2ft1;::: ;tm g*

*| {'z0*

*| { z0*

*}*

*}*

*(5)*

*That is, formula is split into two conjuncts: in the rst one (which is itself* a conjunction), the quanti ed variable i is instantiated to all of t1;::: ; tm, and in the second one, i is required to be di erent from all these terms. We show that the satis ability of Formula (5) is equivalent to the satis ability of its rst two conjuncts, i.e. '0, which make a quanti er-free formula, whose satis ability is decidable.

*Indeed, any model of Formula (5) is also a model for '0. Conversely,* suppose ' has a model M'0 . Then, M'0 clearly induces a model M' for '. As # is semi-universal, this means there exists also a model M for such that M' and M agree on the values of the free variables common to ' and . Now, M can be extended into a model M0 of : the latter may have more

*0*

*0*

*free variables than because of the terms i 2= ft1;::: tmg, and we evaluate*

*these supplementary variables according to M'0 .*

*We build a valuation M#0 for the free variables and functions of For-* mula (5) as follows. The values of the free variables of '0 (resp. 0) are taken from M'0 (resp. M0 ). The values of the functions fj (j = 1 ::: n) are de ned as follows: at all positions i de ned by the values of t1;::: ; tm in M'0 we let fj(i) evaluate according to M'0 , and at all positions i di erent from the values of t1;::: ; tm in M'0 we let fj(i) evaluate according to M0 .

*The construction of M#0 is sound because we have assumed that ' has* function depth 1, thus, t1;::: ; tm depend only on the free variables, and M'0 , M0 agree on the values of the free variables common to ' and . We now prove that M#0 is a model for Formula (5). By construction, M#0 evaluates the rst two conjuncts of (5) just as M'0 does, thus, M#0 satis es the rst two conjuncts. Similarly, M#0 evaluates the third conjunct of (5) just as M0 does, hence, M#0 also satis es the third conjunct, and the proof is done. 2

*Let us illustrate the proof of Lemma 2.4 by the following example.*

*Example 2.5 Consider the semi-universal formula # from Example 2.3. The* formula # is equivalently rewritten as:

*#0 : f (x + 1) f (y)+ 1 ^ f (x + 1) = y +1 ^ f (y) = y +1 ^*

*| {'z0 }*

*^ 8i 2= fx + 1; yg:(f (i) = y + 1)*

*| { z0 }*

*We show that satis ability of #0 is equivalent to satis ability of its quanti er-* free part, i.e. '0. Let

*M'0 : fx = 0;y = 2;f (1) = 3;f (2) = 3; and for all k 2= f1; 2g : f (k) = 0g*

*be a model for ' . Here, M'0 is also a model for '. A corresponding model* for , which agrees with M'0 on the values of the common free variables, is

*0*

*M : fy = 2; for all k : f (k) = 3g*

*We extend M into a model for M0 of by choosing x = 0 as in M'0 , and* build a valuation M#0 as shown in the proof of Lemma 2.4. The values of free variables x and y are chosen from M'0 : x = 0, y = 2. The values of f (i) such that i = x +1 or i = y, that is, for i 2 f1; 2g, are also chosen from M'0 : f (1) = 3; f (2) = 3. The values of f (i) such that i 62 f1; 2g are chosen from M0 , that is, for all i 2= f1; 2g : f (i) = 3. Thus, the valuation M#0 , which is also a model for #, is

*0*

*M#0 : fx = 0;y = 2; for all k : f (k) = 3g*

*2*

*We now give a simple condition for a formula to be semi-universal.*

*De nition 2.6 The universal closure of a formula is obtained by univer- sally quantifying every free variable in . A formula is universally satis able if its universal closure is satis able.* *2*

*De nition 2.7 A simple semi-universal formula is a conjunction ' ^ 8i1: 1 ^*

*::: ^ 8ik: k satisfying the syntactic conditions of De nition 2.2 and the se-* mantic condition that : 8i1: 1 ^ ::: ^ 8ik: k is universally satis able. *2*

*In a simple semi-universal formula, there exist values of the functions that,* together with any values of the variables, constitute a model for the quanti ed part *In particular, the values of the variables can be chosen from a model M*

*of the unquanti ed part ', thus, the semantic constraints from De nition 2.2*

*are met. Checking De nition 2.7 is easier than checking De nition 2.2. It* can be done, e.g., using theorem proving by providing witness values for the function symbols.

*Example 2.8 Consider the following formula*

*# : f (x + 1) f (y)+ 1 ^ 8i:(f (i) = i + 1)*

# *| {'z } | { z }*

*It is a simple semi-universal formula as it satis es the syntactic conditions* of De nition 2.7, i.e. has functions depth 1, ' is quanti er-free, and is shielded. For the semantic condition, note that coincides with its universal closure and is satis able. *2*

## *3 PF-Automata*

*In this section we de ne the syntax and semantics of a class of extended* automata with guards and assignments in the semi-universal fragment of pf.

*De nition 3.1 A variable assignment to variable x is an expression of the form x0 = t, where t is a term of pf.* *2*

*De nition 3.2 A function assignment to function f is a shielded pf formula* of function of depth 1, which has the form 8i:(e1(i) f 0(i)= e2(i)), where e1 is a quanti er-free formula, e2 is a term, and f 0 does not occur in e1; e2. 2

*De nition 3.3 A pf-automaton is a tuple hQ; q0; V; F; ; T i:*

*Q = f1; 2;::: ; jQjg is a nite set of locations,*

*q0 2 Q is the initial location,*

*V is a nite set of integer variables,*

*F is a nite set of unary function symbols,*

*is a simple semi-universal pf formula, called the initial condition,*

*T is a nite set of transitions. Each transition is a tuple hq; ; ; ; q0i where*

*q 2 Q is the origin of the transition,*

*is a quanti er-free pf formula of function depth at most 1, called the* guard of the transition,

*is a nite set of variable assignments (cf. De nition 3.1). For each* variable v 2 V , there is at most one assignment to v in ,

*is a nite set of function assignments (cf. De nition 3.2). For each* function f 2 F , there is at most one assignment to f in ,

*q0 2 Q is a location called the destination of the transition.*

*2*

*Note that the initial condition is required to be a simple semi-universal* formula. As membership in this class is not decidable, other techniques (e.g., theorem proving, cf. end of Section 2.3) may be needed to establish that a given structure is a pf-automaton. We expect that pf-automata which model

*\real" programs will have rather simple initial conditions, whose satis ability*

*is not hard to assess.*

*pf-automata are useful for modeling programs with unbounded data struc-* tures such as les, bu ers, and arrays of parametric size. In De nition 3.3, we have assumed that the only basic type is integer, but, of course, other ground types (Booleans, enumerations, records, subranges) can be encoded using in- tegers. The restriction that there is at most one assignment for each variable and function application is useful for avoiding semantic complications (i.e., situations where a function gets two di erent values simultaneously). It can be dealt with in practice by introducing new transitions to sequentialize the assignments.

*Figure 1 is an example of pf-automaton, which models the sorting algo-* rithm of a vector g of parametric size m using a bubble-sorting procedure. Initially, the actual parameters g and m are equal to the formal parameters of the procedure, f and n. Then, f is sorted and copied back into g at the end of the procedure.

*In Figure 1, expressions such as f 0(i) = f (j) are abbreviations for function* assignments of the form, e.g., 8k:(k = i f 0(k) = f (j)). The meaning of

i = 0  n = m n> 2 k . (0 k < n f(k) = g(k))



**1**

j’ = i+1

j < n−1 j’ = j+1

**2**

f(i) f(j)

**4**

f(i) > f(j)

aux’ = f(i)

f ’(i) = f(j) j = n−1  i < n−2 i’ = i+1

**3**

f ’(j) = aux

i = n−2

k . (0 k < n g’(k) = f(k))

**5**

*Fig. 1. Example of pf-automaton: Vector Sorting*

*such an assignment is that the next value of f (i) will be current value of* f (j), and the next value of f (w) for w 6= j will remain equal to its current value. By convention, variables that do not appear in assignments also remain unchanged.

*Semantics of pf-automata.*

*A valuation is a mapping that assigns, to each free variable appearing in the* automaton, a value in Z, and to each function symbol, a function from Z to Z. We denote by V the set of all valuations. A state is a pair (q; v) consisting of a location q 2 Q and a valuation v 2 V. Note that, for a pf-automaton with at least one function symbol, there is an uncountably in nite set of states. An initial state is a state of the form (q0; v0) such that v0 j= , that is, the location is initial and the values of the variables and functions satisfy the initial condition . The set of states is denoted by S, and the set of initial states is denoted by S0. Each transition 2 T de nes a transition relation

*% S S, in the following way. Intuitively, s and s0 are in the relation %*

*if the location of s (resp. of s0) is the origin (resp. destination) of , and the* variables and functions in s satisfy the guard of . Moreover, the variables and functions get new values according to the assignments of .

*Formally, for a pf formula ' and a valuation v 2 V, let '[v] be the truth* value of ' when the free variables and the function symbols of ' evaluate according to v. For a term t, we denote by t[v] the integer value obtained by evaluating variables and function symbols according to v. We now de ne how valuations are modi ed by assignments.

*According to De nition 3.3, there are two kinds of assignments: variable* assignments and function assignments. Let v0 2 V be the valuation obtained from v after an assignment, then, v0 is obtained in the following way. If the assignment is of the form x0 = t, where x is a variable, then v0 is the valuation such that for all u 2 F [ (V n fxg), v0(u) = v(u), and v0(x) = t[v]. Otherwise, the assignment is of the form 8i:(e1 f 0(i) = e2), where e1 is a

*quanti er-free formula and e2 is a term, both over the variables V [ fig and* functions F . Then, v0 is the valuation such that for all u 2 V [ (F n ff g), v0(u) = v(u), and v0(f ) is de ned as follows: for any i0 2 Z, let e1[v; i=i0] (resp. e2[v; i=i0]) denote the value of e1 (resp. e2) when the free variables V evaluate according to v, and i equals i0. Then, for any i0 2 Z such that e1[v; i=i0] holds, v0(f )(i0) = e2[v; i=i0], and v0(f )(i0) = v(f )(i0) otherwise.

*Finally, for a transition = hq; ; ; ; q0i, we denote by v[ ; ] the valuation* obtained by successively transforming v according to the assignments of . (Note that the order in which this is done is not important, because there is at most one assignment per variable or function). Then, the transition relation % of transition = hq; ; ; ; q0i is the smallest relation de ned by the rule

*s; s0 2 S; s = (q; v); s0 = (q0; v0); [v] = true; v0 = v[ ; ]*

*% (s; s0)*

*We denote the transition relation of the pf-automaton by % = S 2T % . A*

*run is a sequence of states : s0; s1;::: ; sn such that s0 2 S0, and for*

*i = 0;::: ;n 1, %(si; si+1) holds. The length of a run : s0; s1;::: ; sn is n.*

## *4 Symbolic Analysis*

*We show how to decide reachability of a set of states by a bounded-length* run. For a transition and an arbitrary predicate on states, the predicate post ( ) characterizes the states s0 that can be reached by taking transition

*from some state s satisfying :*

*post ( ) : 9s:% (s; s0) ^ (s)*

*For a sequence of transitions : 1;::: ; n and a predicate , the predicate* post ( ) is de ned as post ( ) : post (post (::: post ( ))). We identify sets of states with the formulas that characterize them, and let the set of states

*n n 1* *1*

*be a quanti er-free pf formula. Clearly, is reachable in at most m steps if*

*and only if there exists a sequence of transitions, of length at most m, and* starting in the initial location, such that post (S0) \ 6= ;, where S0 is the set of all initial states. Since there are nitely many sequences of transitions up to a given length, it is enough to show that, for any nite sequence of transitions starting in the initial location, the formula post (S0) ^ is in a class where satis ability is decidable.

*Lemma 4.1 For a pf-automaton with initial set of states S0 and any se- quence of contiguous transitions of the automaton, post (S0) is of the form*

*9x ; x2 ;::: ; x(k 1) 9f ; f2 ;::: ; f (k 1) :#( x; x ; x2 ;::: ; x(k 1) ; (6)*

*f; f ; f1 ;::: ; f (k 1) )*

*where # is a semi-universal formula.*

*Here, x ; x2 ;::: ; x(k 1) and f ; f2 ;::: ; f (k 1) (resp. x and f ) are the values* of free variables and functions on previous steps (resp. on current step) of the calculation of post (S0).

*Proof. We show by induction on the length k of that post (S0) has the* form (6).

*The base step is obvious: by De nition 3.3 of pf-automata, the initial states* S0 are described by the formula pc = q0 ^ , where the initial condition is a semi-universal formula. Thus, the post-image by an empty sequence of the initial states is a formula of the desired form (6).

*For the inductive step, consider a transition = hq; ; ; ; q0i of the* pf-automaton. For simplicity, we assume without loss of generality that the pf-automaton has only one function symbol f . This means that the function assignments of transition consists of one element, of the form 8i:(e1(i; x; f ) f 0(i) = e2(i; x; f )). Then, for a state predicate of the form (6), the predicate post ( ) can be written as Formula (7)

*9x ;::: ; xk 9f ;::: ; fk :( pc = q ^ (x ;f ) ^ x = (x ;f ) ^ (7)*

*#(x ;::: ; xk ;f ;::: ; fk ) ^*

*8i:(e1(i; x ;f ) f (i) = e2(i; x ;f ) ^*

*8i:(:e1(i; x ;f ) f (i) = f (i)))*

*(See Example 4.2 for an illustration.) The meaning of Formula (7) is that the* next control is at location q, that must hold on the values of the variables and the function at the previous step, i.e. on x and f , and that must hold on the values of the variables and the function at all previous steps, i.e. on x ;::: ; xk and f ;::: ; fk . Moreover, the variables are modi ed by the assignments , depending on their previous values and that of the function, and the function is updated at all positions i where e1 holds of i, x , and f .

*What we still have to show is that the formula obtained from (7) after* removing the existential quanti ers is a semi-universal formula. By induction hypothesis, # is a semi-universal formula. Thus, # is a formula of function depth 1, of the form ' ^ 8i1 1 ^ ::: ^ 8im: m, where ' is a quanti er-free pf formula, all formulas 1;::: ; m are quanti er-free, and the semantic property of De nition 2.2 holds: for every model M' of ', there exists a model M of : 8i1: 1 ^ ::: ^ 8im: m such that the free variables among x ;::: ; xk that are common to ' and have the same values in M', M.

*Now, it is possible to write the formula obtained from (7) after removing all* existential quanti ers, as the conjunction '0 ^ 0, where '0 : x = (x ;f ) ^ ', and 0 : 8i1 1 ^ ::: ^ 8im: m ^8i:(e1(i; x ;f ) f 0(i) = e2(i; x ;f ) ^ 8i:(:e1(i; x ;f ) f 0(i) = f (i))). Clearly, '0 is a quanti er-free pf formula, all conjuncts of 0 are shielded, and all are of function depth 1. To complete the proof, we just have to show that the semantic property of De nition *2.2*

*holds. By induction hypothesis, assume that M' is a model of '. We build a*

*model M'0 for '0 as follows: the \old" variables x ;::: ; xk take their values* from M', and the \new" variables x are de ned by the equality x = (x ;f ).

*We also build a model M0 for as follows: \old" variables x ;::: ; xk and* functions f ;::: ; fk take their values from the model M. Formula 0 does not refer to the values of the \new" variables x, and the value of the \new" function f is built as follows: for all values of i, if e1(i; x ;f ) holds, then the value of f (i) equals that of e2(i; x ;f )), otherwise the value of f (i) equals that of f (i).

*0*

*We still have to show that M'0 and M0 agree on the values of their* common free variables. But these are just the variables among x ;::: ; xk that were also common to ' and . By induction hypothesis, M' and M agree on the values of these variables, and, by construction, M'0 agrees with M' on all these values, and M0 agrees with M'0 . The proof is done. *2*

*Example 4.2 For example, if true, the predicate post ( ) for the tran-* sition from location 2 to location 3 of the pf-automaton depicted in Fig *1*

*is:*

*9f 9i ;j ; aux :( pc = 3 ^ f (i ) > f (j ) ^ aux = f (i ) ^ i = i ^* j = j ^ true ^ 8k:(k = i f (k) = f (j )) ^ 8k:(k 6= i f (k) = f (k))

*Here, pc is a new integer variable used to encode presence at a given location.* For simplicity, we did not include variables m; n and function g, which are not modi ed by the transition. *2*

*Discussion.*

*The proof of Lemma 4.1 shows that checking reachability in m steps involves* checking satis ability of a semi-universal formula # with m copies of each vari- able and function symbol. This, in turn, involves instantiating every universal quanti er from the universal part of # to all the terms in its quanti er-free part (cf. Lemma 2.4). Finally, a decision procedure for quanti er-free Pres- burger arithmetic with uninterpreted function symbols is used to decide the resulting quanti er-free formula. We use the ics decision procedure package from SRI International [9].

*Preliminary results with our symbolic analysis prototype are encouraging:* for example, a symbolic simulation of a path of about ten thousand steps in a vector-sorting algorithm was completed in about twenty hours. This means ten thousand calls to the decision procedures for checking formulas with thousands of variables and function applications. As optimizations in both ics and our prototype are still being developed, we expect to be able in the future to perform symbolic simulation on real-size programs and speci cations.

*Finally, it worth noting that a simple extension of pf-automata which con-* sists in letting the guards be universal a pf formulas, is too expressive for symbolic simulation, as reachability even in one step becomes highly undecid- able (cf. Section 2.3).

## *5 Conclusion, Related Work, and Future Work*

*In search of new in nite-state models for which some veri cation problems* are still solvable, we investigate in this paper a class of extended automata, called pf-automata, with guards and assignments in a decidable fragment of Presburger arithmetic with uninterpreted function symbols. This formalism allows us to model quite naturally programs with unbounded data structures such as parametric-sized vectors and arrays, or communication protocols with unbounded channels that carry unbounded data. The model is expressive: the initial condition may constrain an unbounded number of function values, and assignments may modify an unbounded number of such values. The latter can be viewed as meta-transitions, which encode in one step the execution of an unbounded number of transitions. We present a decision procedure for the bounded reachability problem in this model. The procedure works by symbolically simulating the initial states over nite sequences of transitions of the automaton. It is implemented in caml and uses the ics decision procedures from SRI International [9]. Symbolic simulation has a number of practical applications:

*it is a useful technique for understanding and debugging programs by inter-* active execution,

*it can be employed for performing accurate structural testing: given a cover-* age criterion, which is a nite set of nite paths in the program (as computed by, e.g., a commercial tool [6]), our procedure computes input data for ex- ecuting those paths that are executable and discards those that are not, resulting in complete coverage for the given criterion, nally,

*it can also be used as a semi-decision procedure for formal veri cation of* safety properties: if a safety property does not hold, our reachability pro- cedure detects this, otherwise, it will loop forever.

*In another related work [22] we study the deductive veri cation of in nite-state* systems modeled by pf-automata. We introduce techniques for automatically verifying that a predicate is an inductive invariant in a given context, and identify a class of systems and a logic for expressing invariants and contexts for which the problem is decidable.

*Related Work.*

*There is a large amount of work on the analysis of automata extended with* integer variables. See, e.g., [4] for new results and as a good stating point. Concerning automata with uninterpreted function symbols, a recent result

*[7] shows that simulation is decidable for certain classes of such automata.*

*The only results we are aware of about automata extended with Presburger* arithmetic and function symbols are reported in [15]. Where, the Omega tool is used to analyze graphs whose edges are annotated by quanti er-free formulas in this logic. A reachable operation computes an over-approximation

*of the reachable states. The operation will work for graphs with either cycles* or functions, but not both.

*The other main area of related work is structural testing. Most commer-* cial tools (e.g., [6]) can measure the coverage with respect to a given coverage criterion for input data provided by the user. Moreover, the paths in the cri- terion are not always executable. Research on synthesizing input data using symbolic simulation was started in the seventies [20,5] and has received re- newed attention recently [11]. However, these techniques are currently limited to programs with scalar data types (vectors and arrays are not treated). Some techniques for dealing with vectors and arrays have been proposed [14,10]. To our knowledge, our model, which allows an unbounded number of values to be constrained by the initial condition and the assignments, is among the most expressive for which symbolic simulation techniques exist.

*Future Work.*

*The main direction of future work consists in optimizing the prototype sym-* bolic analysis tool and adding new features to it, such as structural path computation for several coverage criteria, analysis of variable dependencies, and slicing. We are also interested in using symbolic analysis techniques for obtaining coverage measures in conformance testing.

## *References*

*[1] W. Ackerman. Solvable Cases of the Decision Problem. North-Holland Publishing Company, Amsterdam, 1954.*

*[2] W. Bledsoe. A new method for proving certain Presburger formulas. In 4th International Joint Conference on Arti cial Intelligence, Tbilissi (USSR), pp. 15{21.*

*[3] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and J. Hwang. Symbolic model checking: 1020 states and beyond. Information and Computation, 98(2):142{170,* *1992.*

*[4] T. Bultan, R. Gerber, and W. Pugh. Model checking concurrent systems with unbounded integer variables: symbolic representations, approximations, and experimental results. ACM Transactions on Programming Languages and Systems, 21(4):747{789,* *1999.*

*[5] L.A. Clarke. A system to generate test data and symbolically execute programs. IEEE Transactions on Software Engineering, 2(3):215{22,* *1976.*

*[6] S. Cornett. Code coverage analysis. Available at* [*http://www.bullseye.com/coverage.html*](http://www.bullseye.com/coverage.html)

*[7] W. Damm, A. Pnueli, and S. Ruah. Herbrand automata for hardware veri cation. Conference on Concurrency Theory (CONCUR'98), LNCS 1466,*

*pp. 67{83, 1998.*

*[8] J-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, and M. Sighireanu. CADP: A protocol validation and veri cation toolbox. Computer- Aided Veri cation, CAV'96, LNCS 1102, pp. 437{440,* *1996.*

*[9] J-C. Filli^atre, S. Owre, H. Rue , and N. Shankar. ICS: Integrated Canonizer and Solver. To be presented at CAV'2001. Available at* [*http://www.csl.sri.com/papers/cav01/*](http://www.csl.sri.com/papers/cav01/)

*[10] A. Goldberg, T-C. Wang, and D. Zimmerman. Applications of feasible path analysis to program testing. International Symposium on Software Testing and Analysis, ISSTA'94, pp. 80{94, 1994.*

*[11] E. Gunter, and D. Peled. Path exploration tool. Tools and Algorithms for the Construction and Analysis of Systems, TACAS'99, LNCS 1579, pp. 405{479, 1999.*

*[12] G.J. Holzmann. Design and Validation of Communication Protocols. Prentice Hall, 1991.*

*[13] J. Halpern. Presburger arithmetic with uninterpreted function symbols is 1-*

*1*

*complete. Journal of Symbolic Logic, 56:637{642,* *1991.*

*[14] R. Jasper, M. Brennan, K. Williamson, B. Currier, and D. Zimmerman. Test data generation using feasible path analysis. International Symposium on Software Testing and Analysis ISSTA'94 pp. 95{107, 1994.*

*[15] W. Kelly, V. Maslov,*

*W. Pugh, E. Rosser, T. Shpiesman, and D. Wonnacott. The Omega library interface guide. Available at* [*http://www.cs.umd.edu/projects/omega.*](http://www.cs.umd.edu/projects/omega)

*[16] G.J. Myers. The Art of Software Testing. John Wiley and Sons, 1979.*

*[17] D. Oppen. A 222pn upper bound on the complexity of Presburger arithmetic. Journal of Computer and System Sciences, 16(3):323{332,* *1978.*

*[18] S. Owre, J. Rushby, N. Shankar, and F. von Henke. Formal veri cation for fault- tolerant architectures: prolegomena to the design of PVS. IEEE Transactions on Software Engineering, 21(2):107{125,* *1995.*

*[19] M. Presburger. U ber die Vollstandigkeit eines gewissen Systems der Arithmetik ganzer Zahlen in Welchem die Addition als einzige Operation hervortritt. Sprawozdanie z I Kongresu Matematykow Krajow Slowcanskich Warszawa, Poland, pp. 92{101, 1929.*

*[20] C. Ramomoorthy, S. Ho, and W. Chen. On the automated generation of program test data. IEEE Transactions on Software Engineering, 2(4):293{300, 1976.*

*[21] S. Rapps, and E. Weyuker. Selecting software test data using data ow information. IEEE Transactions on Software Engineering, 11(4):367{375, 1985.*

*[22] V. Rusu. Verifying that invariants are context-inductive. Submitted for publication.*

*[23] R. Shostak. A practical decision procedure for arithmetic with uninterpreted function symbols. Journal of the ACM, 26(2):351{360,* *1979.*