Symbolic Encoding of String Lengths

Fang Yu
University of California, Santa Barbara
yuf@cs.ucsb.edu

Tevfik Bultan
University of California, Santa Barbara
bultan@cs.ucsb.edu

Oscar H. Ibarra
University of California, Santa Barbara
ibarra@cs.ucsb.edu

ABSTRACT

We present a novel construction for length automata which accept the unary or binary representations of the length of a regular language. The construction can be used for verification of systems having unbounded strings and integers.

1. MOTIVATION

This project is motivated by an increasing interest in static analysis for real-world programs [1, 4]. We model these programs as infinite state systems having string and integer variables and present an automata-based approach for verification of these systems. Particularly, we are interested in the relationship among the values of integer variables and the lengths of strings. In general, the precise analysis of infinite state system is undecidable due to the fact that the values of integer variables are unbounded, the lengths of string variables are unbounded, and the reachability analysis of such infinite state systems is undecidable. To overcome this complexity hurdle, checking these infinite-state systems can be achieved by encoding infinite sets of states as regular languages and computing conservative approximations of the reachable states. A conservative approach is commonly adopted using deterministic finite automata (DFAs) to approximate the set of string values that string variables can take at certain program points, as well as to widen the arithmetic constraints that symbolically represent the set of values that integer variables can take. Previous works on static analysis of infinite state systems focus on either string variables [1,3,6] or integer variables [2,5]. We are interested in both. We adopt [6] to deal with string variables and its operations, and [2] for integer variables. Both infinite states of string and integer variables are approximated as regular languages and encoded as DFAs. In addition, we investigate the length constraint on the language accepted by a string automaton. Based on this constraint, one can catch the relation among the lengths of string variables and the values of integer variables. We believe this relation can be used to perform sophisticated verification on systems having unbounded string and integer variables.

In this abstract, we show that the length of the language accepted by a DFA forms a semiliner set. Given an arbitrary DFA, we are able to construct DFAs that accept either unary or binary representation of the length of its accepted words. The unary automaton can be used to identify the coefficients of the semilinear set, while the binary automaton can be used to compose with other arithmetic automata on integer variables.

The performance of our analysis relies on efficient automata manipulation. We implement all functions using a symbolic automata representation (MBDD representation from the MONA automata package) and leverage efficient manipulations on MBDDs, e.g., determinization and minimization. We believe that the symbolic representation (compared to explicit representation) of automata can be better scaled to model large systems and facilitates our tool to analyze real-world programs.

2. LENGTH AUTOMATA CONSTRUCTION

We are interested in identifying what length can be among the accepted words of a string automaton. Given a string automaton M, we aim to construct a DFA M_b (over a binary alphabet) such that M_b accepts the binary encodings (from the least significant bit) of the lengths of the words accepted by M. The following property is well known.

Property 1: For any DFA M, $\{n|n = |w|, w \in L(M)\}$ forms a semilinear set.

However, identifying the length set of arbitrary regular language is not trivial, e.g., the length set of $((baaab)^+ab)^+$ is $\{7, 12, 14, 17, 19, 21, 22, 24, 25, 26, 27, 28\} \cup \{29 + k|k \geq 0\}$. We tackle this problem by constructing the automaton M_u that accepts the unary encodings of the lengths of the accepted words of a string automaton M. Given $M = \langle Q, q_0, B^k, \delta, F\rangle$, where each symbol is encoded as a k-bit string, M_u is constructed by determinizing the NFA $\langle Q, q_0, B^1, \delta', F\rangle$, where $\delta'(q, 1) = q'$ if $\exists\alpha, \delta(q, \alpha) = q'$. M_u uniquely identifies a semilinear formula $\bigvee_i x = c_i \vee \bigvee_j \exists k.x = C + r_j + Rk$, where c_i, r_j, C, R are constants, and $\forall i, c_i < C$, and $\forall j, r_j < R$. The length set is exactly the set of values of x that satisfy the formula.

In the following, we propose an incremental algorithm to construct a DFA that accepts the binary encodings (from the least significant bit) of the values of x that satisfy a given semilinear formula.

The construction is achieved by calling the procedure CONSTRUCT_BLA. At line 3, we first compute the set of reachable binary states Q^b by calling the recursive procedure ABS (Add Binary State). A binary state is the value of a triple (t, v, b). $t \in \{0, 1, 2\}$ is the type of the binary state, which indicates the meaning of the value of v and b. While $t = 0$, v is equal to the value of the binary word accepted from the initial state to the current state, and b is equal to the binary value of the previous bit in the word. While $t \neq 0$, v is equal to the remainder of which the dividend is the value of the binary word accepted from the initial state to the current state and the divisor is R; b is the remainder of which the dividend is the binary value of the previous bit

in the accepted word and the divisor is R. $t = 1$ indicates the value of the binary word accepted from the initial state to the current state is greater or equal to C; $t = 2$ indicates the value is less than C. Each binary state is further associated with an index, a true branch and a false branch, which are used to construct the state graph later. Briefly, ABS is a recursive procedure which incrementally adds the encountered binary state if it has never been explored. Initially, the binary state is $(0, 0, \perp)$. We assume $2 \perp = 1$. Since binary states are finite, ABS is guaranteed to terminate. Upon termination, all reached binary states are added to Q^b. For each binary state in Q^b, as line 4 to 9, we iteratively generate a state q, and set its transition and final status. We construct the final automaton at line 10.

Algorithm 1 ABS(Q, C, R, t, v, b)

```
1: if ∃q = (t, v, b) ∈ Q then
2:    return q.index;
3: else
4:    Create q = (t, v, b);
5:    q.index = ♯Q; q.true = q.false = −1;
6:    Add q to Q;
7:    if t == 0 ∧ (v + 2 × b ≥C) then
8:       q.true =ABS( Q, C, R, 1, (v+2×b)%R, (2×b)%R);
9:       q.false =ABS( Q, C, R, 2, v%R, (2 × b)%R);
10:   else if t == 0 ∧ (v + 2 × b < C) then
11:      q.true =ABS( Q, C, R, 0, v + 2 × b, 2 × b);
12:      q.false =ABS( Q, C, R, 0, v, 2 × b);
13:   else if t == 1 then
14:      q.true =ABS( Q, C, R, 1, (v+2×b)%R, (2×b)%R);
15:      q.false =ABS( Q, C, R, 1, v%R, (2 × b)%R);
16:   else if t == 2 then
17:      q.true =ABS( Q, C, R, 1, (v+2×b)%R, (2×b)%R);
18:      q.false =ABS( Q, C, R, 2, v%R, (2 × b)%R);
19:   end if
20:   return q.index;
21: end if
```

Algorithm 2 CONSTRUCT_BLA(C, R, $\mathcal{C} = \{c_1, c_2, \ldots c_n\}$, $\mathcal{R} = \{r_1, r_2, \ldots r_m\}$)

```
1: Q^b = ∅;
2: Q = ∅;
3: init =ABS( Q^b, C, R, 0, 0,⊥);
4: for each q^b ∈ Q^b do
5:    Add q = q_{q.index} to Q;
6:    δ(q, 1) = (q^b.true ≠ −1?q_{q^b.true} : q_sink);
7:    δ(q, 0) = (q^b.false ≠ −1?q_{q^b.false} : q_sink);
8:    F(q) = ((q^b.t == 0 ∧ ∃c ∈ C.q^b.v == c) ∨(q^b.t ==
      1 ∧ ∃r ∈ R.q^b.v == (r+C)%R) :' +'?'−');
9: end for
10: Construct M = ⟨Q ∪ {q_sink}, q_init, B^1, δ, F⟩;
```

Implementation.

We have implemented the above algorithms using MONA DFA packages. We give an example in Figure 1. The length automaton accepts the binary representatin of the semilinear set $\{7 + 5k|k \geq 0\}$. Consider the number 1087, whose binary encoding is 10000111111. One can check that the bit string from the least significant is accepted by the automaton shown in Figure 1.

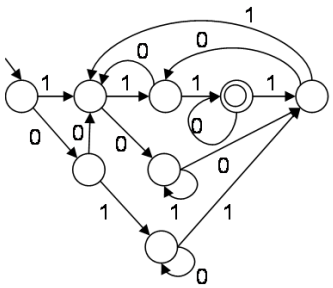

Figure 1: The Length Automata of $(baaab)^+ab$. The Semilinear Set is $\{7 + 5k|k \geq 0\}$

3. CONCLUSION

We have presented the algorithms to construct DFAs that accept unary or binary encodings of the length of a regular language.The DFAs that accept the binary encodings can be further composed to one multi-track arithmetic automaton [2] with other integer variables. This arithmetic automaton may catch the relation among the lengths of string variables and the values of integer variables. One can check the length properties of string variables or enforce the constraints on the lengths of string variables using the length set derived from the multi-track arithmetic automaton.

To complete our work, we will focus on: (1) the forward image computation of string and arithmetic automata against common string and arithmetic operations, (2) the widening operator to accelerate the fixed point computation, and (3) the general symbolic verification framework on systems having strings and integers. We plan to apply these techniques to check buffer overflows.

4. REFERENCES

[1] D. Balzarotti, M. Cova, V. Felmetsger, N. Jovanovic, C. Kruegel, E. Kirda, and G. Vigna. Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications. In *Proceedings of the Symposium on Security and Privacy*, 2008.

[2] Constantinos Bartzis and Tevfik Bultan. Efficient symbolic representations for arithmetic constraints in verification. *Int. J. Found. Comput. Sci.*, 14(4):605–624, 2003.

[3] Aske Simon Christensen, Anders Møller, and Michael I. Schwartzbach. Precise analysis of string expressions. In *Proc. 10th International Static Analysis Symposium, SAS '03*, volume 2694 of *LNCS*, pages 1–18. Springer-Verlag, June 2003.

[4] Christian Kirkegaard, Anders Møller, and Michael I. Schwartzbach. Static analysis of xml transformations in java. *IEEE Transactions on Software Engineering*, 30(3), March 2004.

[5] Pierre Wolper and Bernard Boigelot. On the construction of automata from linear arithmetic constraints. In *TACAS*, pages 1–19, 2000.

[6] Fang Yu, Tevfik Bultan, Marco Cova, and Oscar H. Ibarra. Symbolic string verification: An automata-based approach. In *15th International SPIN Workshop on Model Checking of Software*, 2008.