DECISION PROCEDURES FOR VERIFICATION

by
Harvey M. Friedman
Distinguished University Professor of Mathematics, Philosophy, Computer Science
Ohio State University
friedman@math.ohio-state.edu
http://www.math.ohio-state.edu/~friedman/

Joint Theory of Computation and Programming Language Seminar
Department of Computer Science
Harvard University
delivered October 9, 2009

ABSTRACT

We focus on two formal methods contexts which generate investigations into decision problems for finite strings.

- RESOLVE Verification Conditions (VCs)
- JAVA Pathfinder

At Ohio State and elsewhere, formal specifications are given and annotated programs are written (providing loop invariants) that are designed to meet those specifications. This generates mathematical statements called VCs (verification conditions), which guarantee that the annotated program meets the specifications.

If the context is finite strings, then decision procedures for finite strings can be very useful.

We discuss such a decision procedure which we formulated based on our examination of the VCs generated at Ohio State from string processing programs written in RESOLVE. We also discuss the boundary between the decidable and undecidable.

A second source of decision procedure investigations is suggested by a tool for JAVA programs called JAVA PATHFINDER. This is a tool to automatically detect dead code in JAVA programs. It exploits the structure of JAVA programs, and is based on recognizing the impossibility of satisfying finitely many conditions. This naturally leads to a very wide ranging investigation into decision procedures involving the primitives in JAVA libraries, such as string replacement x[y/z]. We discuss some decidability and undecidability results for this context.

INTEGERS, OBJECTS, STRINGS, CONCATENATION FOR VCs

Language L has three sorts: integers, objects, and (finite) strings (of) objects. Variables n_i, x_i, α_i.

Linear arithmetic, strict linear ordering on objects, empty string, concatenation, length of string, object to length 1 string, n-th term of string, weakly increasing string.

Terms, atomic formulas, universal formulas, universal sentences, are defined in the expected way.

In all interpretations, everything is predetermined except the choice of the linear ordered set of objects.

WHICH UNIVERSAL SENTENCES ARE TRUE IN ALL INTERPRETATIONS?

THEOREM. A universal sentence is true in all interpretations if and only if it is true in THE interpretation where the objects are the integers with the usual ordering.

Henceforth, we work only with this standard interpretation. Thus we only speak of the TRUTH or FALSITY of a sentence of L.

INTEGERS, OBJECTS, STRINGS, CONCATENATION FOR VCs

We present the language L formally. We use INT, OBJ, STR.

- binary relation symbols $<,\leq,=,\neq$ of type INT × INT.
- binary function symbols +,- of type INT × INT ⇒ INT.
- unary function symbols | |,- of type INT ⇒ INT.
- constant symbols 0,c of type INT, where c is a nonempty string of base ten digits, not beginning with 0.
- unary function symbol | | of type STR ⇒ INT.
- binary function symbols •,div,mod of type INT × INT ⇒ INT.
- binary relation symbols $<,\leq,=,\neq$ of type OBJ × OBJ.
- ternary relation symbol VAL of type STR × INT × OBJ.
- constant symbol Λ of type STR.
- binary relation symbols =,≠ of type STR × STR.
- binary function symbol ^ of type STR × STR ⇒ STR.
- unary relation symbol WINC of type STR.
- unary function symbol < > of type OBJ ⇒ STR.

NOTE: Use of •,div,mod will be restricted to stay within linear arithmetic.

We have overloaded $<,\leq,-,\neq,|\ |$. This is harmless by the strong typing.

INTEGERS, OBJECTS, STRINGS, CONCATENATION FOR VCs

We now define the terms of L.

- every INT variable and INT constant is an INT term.
- every OBJ variable is an OBJ term.
- every STR variable and STR constant is an STR term.
- let s,t be INT terms. Then s+t, s-t, |s|, -s, are INT terms.
- let c be an INT constant and t be an INT term. Then c•t, t div c, t mod c are INT terms.
- let s be an OBJ term. Then <s> is a STR term.
- let s,t be STR terms. Then s^t is a STR term and |s| is an INT term.

Note that the OBJ terms are just the OBJ variables.

The atomic formulas of L are defined as follows.

- let s,t be OBJ terms. Then $s < t$, $s \leq t$, $s = t$, $s \neq t$ are atomic formulas of L.
- let s,t be STR terms of L. Then WINC(s), $s = t$, $s \neq t$ are atomic formulas of L.
- let s be an STR term, t an INT term, r an OBJ term. Then VAL(s,t,r) is an atomic formula of L.
- let s,t be INT terms. Then $s < t$, $s \leq t$, $s = t$, $s \neq t$ are atomic formulas of L.

INTEGERS, OBJECTS, STRINGS, CONCATENATION FOR VCs

The semantics needs only a few comments.

- the object sort is Z, with its usual order.
- | | on strings is length.
- VAL(α,i,x) holds if and only if the i-th term of the string α is the object x.
- for objects x, <x> is the string of length 1 consisting of x.
- ^ is concatenation of strings.
- WINC(α) if and only if the string α is weakly increasing, in the sense that each term is ≤ the next.

We have given a decision procedure for determining the truth of any universal sentence in L, subject to a natural condition that is always obeyed by the VCs generated by the OSU project (sometimes after trivial preprocessing).

To state this condition, we use the notion of positive/negative occurrences in propositional formulas.

i. p is a positive occurrence in p.

ii. the positive (negative) occurrences in A **∧** B, A **∨** B are the positive (negative) occurrences in A,B.

iii. the positive (negative) occurrences in ¬A are the negative (positive) occurrences in A.

iv. the positive (negative) occurrences in A ⇒ B are the negative (positive) occurrences in A and the positive (negative) occurrences in B.

INTEGERS, OBJECTS, STRINGS, CONCATENATION FOR VCs

There are three types of equations of L: INT, OBJ, and STR equations.

Let A be a quantifier free formula of L. We say that A obeys the positive (negative) STR equation restriction if and only if every string variable occurs at most once in the totality of all positive (negative) occurrences of STR equations in A.

THEOREM. There is a decision procedure for determining the truth value of all universal sentences of L whose quantifier free part obeys the negative STR equation restriction.

The procedure works well in practice. Jeremy Avigad implemented the algorithm in Isabelle for a couple of interesting examples, with very good results. Later, it has been coded as a standalone application by the OSU group, again with very good results.

REVERSE of strings can also be added, as well as SINC (strictly increasing), with the algorithm appropriately modified.

We can also add COUNT(x,α) = number of occurrences of the object x in the string α. We can also add α ≡ β for "having the same set of terms".

Also, WINC, SINC can be generalized to appropriate universal conditions.

INTEGERS, OBJECTS, STRINGS, CONCATENATION FOR VCs

Not clear if the universal sentences of L are decidable with no restriction. This is closely related to the decidability of the observed to be difficult problem of the existential theory of string in a finite alphabet with length equality. Obviously undecidability of the latter immediately implies undecidability of the former.

This difficult problem is discussed, for example, in http://logic.pdmi.ras.ru/~yumat/talks/turku2006/FibonacciWordsAbstract.pdf where Matiyasevich remarks that such an undecidability result would give an entirely new solution to Hilbert's 10th Problem (that there is no decision procedure for determining whether an integral polynomial has an integral solution).

However, we have shown undecidability of the universal sentences of L with no restriction, if we add COUNT. The undecidability uses the negative solution to Hilbert's 10th problem.

The decidability of the satisfiability of unrestricted word equations with constants (no length equations) has a long history. The state of the art is PSPACE. See

W. Plandowski, Satisfiability of Word Equations with Constants is in PSPACE, JACM, vol 51, issue 3, May 2004, p. 483-496.

TWO EXAMPLES

- $\alpha\hat{}\beta = \gamma\hat{}\delta \wedge (|\alpha| = |\gamma| \vee |\beta| = |\delta|) \Rightarrow \alpha = \gamma \wedge \beta = \delta$.
- $\mathrm{WINC}(\alpha\hat{}\langle x\rangle\hat{}\beta) \wedge \mathrm{WINC}(\alpha\hat{}\langle y\rangle) \wedge x < y \wedge |\gamma| = |\alpha|+1 \wedge \gamma\hat{}\delta = \alpha\hat{}\langle x\rangle\hat{}\beta \Rightarrow \mathrm{WINC}(\gamma\hat{}\langle y\rangle)$.

Jeremy Avigad worked up my algorithm in Isabelle, and, in the second case, it found 111 subgoals to prove, in the initial round of reductions.

SETUP FOR THE ALGORITHM

GOAL: Determine whether a quantifier free formula A is satisfiable in integers, objects, and finite strings of objects. Here the objects are also integers. We also assume that the variable condition holds.

First put A into disjunctive normal form, B1 ∨ ... ∨ Bn, where the B's are conjunctions of atomic formulas. The variable condition now just says that in the aggregate of the string equations among the conjuncts, no string variable appears more than once.

We also can drive negation signs in according to these replacements. Below as always, x,y are object variables, s',t' are integer terms, and s,t are string terms.

¬x = y by x ≠ y.
¬x ≤ y by y < x.
¬x < y by y ≤ x.
¬x ≠ y by x = y.
¬s' = t' by s' ≠ t'.
¬s' ≤ t' by t' < s'.
¬s' < t' by t' ≤ s'.
¬s' ≠ t' by s' = t'.
¬s = t by s ≠ t.

A EQUIVALENT TO B_1 ∨ ... ∨ B_n

By pushing negation signs in as indicated, we can assume that A is equivalent to B_1 **∨** ... **∨** B_n, where each B_i is a finite set of atomic formulas of the forms (interpreted conjunctively)

*)
x < y
x ≤ y
x = y
x ≠ y
s = t
s ≠ t
s' < t'
s' ≤ t'
s' = t'
s' ≠ t'
WINC(s)
¬WINC(s)
VAL(s,s',x)
¬VAL(s,s',x)

where x,y are OBJ variables, s',t' are INT terms, s,t are STR terms, and no STR variable appears more than once in the totality of STR equations s = t.

THE INITIAL TREE $T_0(A)$
STRATEGY FOR BUILDING T(A)

The root of $T_0(A)$ is labeled by A. There are n sons of the root, and they are labeled by the finite sets $B_1,...,B_n$, respectively. These are also the leaves of $T_0(A)$.

We wish to determine if A is satisfiable. Note that this is equivalent to "some leaf is satisfiable".

We will now successively modify $T_0(A)$ to eventually create a finite tree T(A), by a nondeterministic process. At each stage, we either keep the same vertices, or we add sons to one of the leaves L. In the former case, we change the label to one leaf. In the latter case, we do not change the label of L.

We will argue that the nondeterministic process must finish, and result in a finite tree T(A). The leaves of T(A) will have rather restricted atomic formulas.

We will also argue that "A is satisfiable if and only if some leaf is satisfiable" holds at any stage of the nondeterministic construction.

MORE STRATEGY FOR BUILDING T(A)

In addition to the equivalence of the satisfiability of the root (A) with satisfiability of some leaf, we also want

i. the variable restriction holds at every vertex.
ii. the label of every nonleaf is “greater” than the labels of each of its sons.

These conditions will be checked incrementally. I.e., as we modify a tree with these properties, we show that these properties are preserved.

Some care is needed to use the right notion of “greater” here. We choose one so that there are no infinite descending sequences.

We now introduce our transformation rules.

RULES

1.1. Λ as proper subterm of a string term. Remove Λ.
1.2. s = Λ or Λ = s, where s has an OBJ variable. Replace conjunction by 1 = 0.
1.3. s = Λ or Λ = s, where s is a concatenation of one or more STR variables. Remove, and replace all occurrences of the variables in s by Λ.
1.4. Λ = Λ. Remove.
1.5. |Λ|. Replace by 0.
1.6. α = t or t = α. Remove, and replace every occurrence of α by t.
1.7. α ≠ Λ or Λ ≠ α. Replace by |α| ≠ 0.
1.8. <x> ≠ Λ or Λ ≠ <x>. Remove.
1.9. ¬WINC(s). Replace by VAL(s,n,x), VAL(s,m,y), n < m, y < x, where n,m,x,y are new variables.
1.10. |t|, where t is not an STR variable. First replace by $|t_1'| + \ldots + |t_n'|$, where $t_1',\ldots,t_n'$ are the components of t, from left to right. Then replace each summand |<x>| by 1, |Λ| by 0.
1.11. ¬VAL(s,t',x). Split with {|s| < t'}, {t' < 1}, {VAL(s,t',y), x ≠ y}, where y is a new variable.
1.12. s ≠ t, where this inequation has at least one variable. Split with {|s| ≠ |t|}, {VAL(s,n,x), VAL(t,n,y), x ≠ y}, where n,x,y are new variables.

MORE RULES

2.1. WINC(<x>), WINC(Λ). Remove.

2.2. WINC($u_1u_2...u_p$), $p \geq 2$, where the u's are either STR variables or some <x>. For each $1 \leq i < p$, let S_i = {WINC(u_i), VAL(u_i,|u_i|,v_i), VAL(u_{i+1},1,w_{i+1}), $v_i \leq w_{i+1}$}. Here $v_1,...,v_{p-1},w_2,...,w_p$ are new OBJ variables. Let S_i' be the result of removing WINC(u_i) in case u_i is some <x>. Replace by S_1' **U** ... **U** S_{p-1}'.

3.1. VAL(Λ,s',x). Replace the conjunction by 1 = 0.

3.2. VAL(<y>,s',x). Replace by s' = 1, y = x.

3.3. VAL(<y>t,s',x). Split with {s' = 1, x = y}, {s' > 1, VAL(t,s'-1,x)}.

3.4. VAL(t,s',x). Split with {s' = 1, VAL(t,s',x)}, {s' > 1, VAL(t,s'-|α|,x)}.

YET MORE RULES

4.1. <x> = <y>. Replace by x = y.

4.2. <x> = <y>t or <y>t = <x>. Replace by x = y, t = Λ.

4.3. <x> = αt or αt = <x>. Split with {<x> = t, α = Λ}, {<x> = α, t = Λ}. Follow the first split by replacing all occurrences of α by Λ. Follow the second split by replacing all occurrences of α by <x>.

4.4. <x>s = <y>t. Replace by x = y, s = t.

4.5. <x>s = αt or αt = <x>s. Split with {α = Λ, <x>s = t}, {s = βt, α = <x>β}, where β is a new STR variable. Follow the first split by replacing all occurrences of α by Λ. Follow the second split by replacing all occurrences of α by <x>β.

4.6. αs = βt. Split with {|α| ≤ |β|, β = αγ, s = γt}, {|β| < |α|, α = βγ, γs = t}, where γ is a new STR variable. Follow the first split by replacing all occurrences of β by αγ. Follow the second split by replacing all occurrences of α by βγ.

LEMMAS

LEMMA 4.1. During the successive application of these rules, starting with $T_0(A)$, the variable restriction applies to every leaf. I.e., no STR variable appears more than once in the totality of STR equations on a leaf.

LEMMA 4.2. Suppose the successive application of these rules, starting with the finite tree $T_0(A)$, results in a finite tree T(A), where no further applications of the rules are possible. Let $\{E_1,...,E_k\}$ be the label of a leaf of T(A). Then the E's are of the forms

$x < y$	concatenation removed
$x \leq y$	$\neg$WINC, $\neg$VAL, Λ removed
$x = y$	
$x \neq y$	
$s' < t'$	
$s' \leq t'$	
$s' = t'$	
$s' \neq t'$	
WINC(α)	
VAL(α,s',x)	

where x,y are OBJ variables, α is a STR variable, and s',t' are INT terms. Furthermore, the only STR terms that appear are STR variables.

MORE LEMMAS

LEMMA 4.3. Let T be any tree that arises during the application of these rules, starting with T0(A). Then A is satisfiable if and only if some leaf of T is satisfiable.

Let a finite set of atomic formulas be given, satisfying condition *) of slide 12. We associate the following 5 nonnegative integers.

Q_1 = total number of occurrences of variables in STR equations, plus the total number of occurrences of WINC,VAL, and ≠ between STR terms.

For Q_2, first list the occurrences of |t|, WINC(t), VAL(t,t',x), where t is not an STR variable.

Q_2 = the total number of occurrences of variables in the STR terms t above, plus the total number of occurrences of Λ anywhere.

LEMMA 4.4. On any application of any of the non splitting rules to a conjunction satisfying condition *), the pair (Q_1,Q_2) is lowered lexicographically, or becomes all 0's. On any application of any of the splitting rules, the pair (Q_1,Q_2) is lower at each of the two sons.

LAST CRUCIAL LEMMA FOR T(A)

LEMMA 4.5. The process of applying the rules in any way, starting with $T_0(A)$, must terminate in a finite tree T(A). We make no claims of uniqueness.

Proof: Suppose the process continues forever.

case 1. Only finitely many vertices are generated. Then after some stage, the tree is fixed, and the labels only are changing. Since there are only finitely many leaves, some leaf has its label updated infinitely often. But each time its label is updated, its (Q_1,Q_2) drops lexicographically. This is impossible.

case 2. Infinitely many vertices are generated. Since the splits are finite (at most 3), an infinite path of vertices will be generated, by the Konig tree lemma. But by looking at their (Q_1,Q_2), we get an infinite descending sequence lexicographically, which is impossible.

QED

T(A) is any resulting finite tree.

SATISFIABILITY OF LEAVES OF T(A)

We reduced satisfiability of A to satisfiability of some leaf of T(A). Recall leafs have sets of formulas

$x < y$ $\quad x \leq y$ $\quad x = y$ $\quad x \neq y$
$s' < t'$ $\quad s' \leq t'$ $\quad s' = t'$ $\quad s' \neq t'$
WINC(α)
VAL(α,s',x)

where x,y are OBJ variables, α an STR variable, s',t' are INT terms, all STR terms appearing are variables.

$(\exists n_1,\ldots,n_p,x_1,\ldots,x_q \in Z)(\exists \alpha_1,\ldots,\alpha_r \in Z^*)(E_1 \wedge \ldots \wedge E_k)$

can be converted to an existential sentence in linear arithmetic. This is because the a's can be eliminated. Recall that concatenation has been eliminated. So strings exist with WINC iff the named positions are compatible with WINC - because we can fill the gaps with copies of the last value before the gap starts.

Existential linear arithmetic is NP complete.

FURTHER INVESTIGATIONS

The method supports much more comprehensive extensions. Two among many important additions to what we have are

1. CT(α,x) = number of occurrences of x in α.
2. α is a permutation of β.

Satisfiability should still be decidable (with the variable restriction). However

THEOREM. If we add 1 or 2, and also allow αβ = βα, then satisfiability is not computable.

Proof: Note that αβ = βα ∧ 0 < |α| < |β| ∧ CT(α,x) = 1 ∧ CT(β,x) = |α| implies $\beta = \alpha^{|\alpha|}$. This means that we have a hook into the squaring operation, looking at lengths. Also, using lengths and concatenation, we have an obvious hook into addition. Therefore we can code up Diophantine equations, and obtain the non computability. We can use 3 instead as follows. WINC(a) ∧ the first two terms of α are distinct ∧ αβ = βα ∧ WINC(γ) ∧ γ is a permutation of αβ ∧ γ = δε ∧ last(δ) = first(α) ∧ first(ε) = second(α) ∧ |δ| = |α| implies $|\alpha\beta| = n^2$. This gives the required hook into squaring. QED

JAVA STRING LIBRARY

These are only a limited part of the Library.

compareTo(String anotherString)
Compares two strings lexicographically.

String

concat(String str)

endsWith(String suffix)
Tests if this string ends with the specified suffix.

boolean

equals(Object anObject)
Compares this string to the specified object.

boolean

equalsIgnoreCase(String anotherString)
Compares this `String` to another `String`, ignoring case considerations.

byte[]

getBytes()
Convert this `String` into bytes according to the platform's default character encoding, storing the result into a new byte array.

byte[]

getBytes(String enc)
Convert this `String` into bytes according to the specified character encoding, storing the result into a new byte array.

void

getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)
Copies characters from this string into the destination character array.

int

hashCode()
Returns a hashcode for this string.

int

indexOf(int ch)
Returns the index within this string of the first occurrence of the specified character.

int

indexOf(int ch, int fromIndex)
Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.

int

indexOf(String str)
Returns the index within this string of the first occurrence of the specified substring.

int

indexOf(String str, int fromIndex)
Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

JAVA STRING LIBRARY

String
intern()
Returns a canonical representation for the string object.

int
lastIndexOf(int ch)
Returns the index within this string of the last occurrence of the specified character.

int
lastIndexOf(int ch, int fromIndex)
Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.

int
length()
Returns the length of this string.

boolean
regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)
Tests if two string regions are equal.

String
replace(char oldChar, char newChar)
Returns a new string resulting from replacing all occurrences of `oldChar` in this string with `newChar`.

boolean
startsWith(String prefix)
Tests if this string starts with the specified prefix.

boolean
startsWith(String prefix, int toffset)
Tests if this string starts with the specified prefix beginning a specified index.

String
substring(int beginIndex)
Returns a new string that is a substring of this string.

String
substring(int beginIndex, int endIndex)
Returns a new string that is a substring of this string.

char[]
toCharArray()
Converts this string to a new character array.

JAVA LIBRARY PRIMITIVES
STRING REPLACEMENT
FOR JAVA PATHFINDER

One of the many JAVA primitives is the string replacement operation x[y/z]. Here x,y,z are strings. In case y is empty, return the default empty string.

x[y/z] is the result of the following process. Lay out the occurrences of y as a substring of x, making sure that they do not overlap by resetting after the end of the previous occurrence. Then replace each of these occurrences by z.

Now let A be an alphabet. We allow A to be infinite. Let A* be the set of all finite strings from A, including Λ. The variables are the v_i, $i \geq 0$. The A-terms are defined as follows.

- Every variable, and every $c \in A^*$ is a term.
- If $c \in A^*$ and t is a term, then ct, tc are terms.
- If s,t,r are terms, then s[t/r] is a term.

The atomic A-formulas are simply equations between terms. The A-formulas are defined in the usual way, using connectives and quantifiers over A*.

JAVA LIBRARY PRIMITIVES
STRING REPLACEMENT
FOR JAVA PATHFINDER

THEOREM. Let A have at least 3 elements. There is no algorithm for determining the truth value of existential A-sentences.

The proof uses the negative solution to Hilbert's 10th Problem.

We can put the quantifier free part of the existential sentence into disjunctive normal form, and take the disjunctions out, getting a disjunction of existentially quantified finite sets of A-literals (atomic A-formulas and their negations). Thus, it suffices to consider solvability in A* of the finite set of A-literals.

By introducing new variables, we can focus on the satisfiability of a finite set of literals of the form

$r_1[s_1/t_1] = p_1$
...
$r_k[s_k/t_k] = p_k$
$x_1 \neq a_1$
...
$x_n \neq a_n$

where each r_j, s_j, t_j, p_j is either a variable or an element of A*, each x_i is a variable among the r_j, s_j, t_j, p_j, and each a_i is a variable among the r_j, s_j, t_j, p_j or an element of A*.

JAVA LIBRARY PRIMITIVES
STRING REPLACEMENT
FOR JAVA PATHFINDER

$r_1[s_1/t_1] = p_1$
...
$r_k[s_k/t_k] = p_k$
$x_1 \neq a_1$
...
$x_n \neq a_n$

where each r_j, s_j, t_j, p_j is either a variable or an element of A*, each x_i is a variable among the r_j, s_j, t_j, p_j, and each a_i is a variable among the r_j, s_j, t_j, p_j or an element of A*.

Our undecidability proof shows that there is a fixed k such that for at most k equations, we have undecidability. However, the k coming from our proof is allied with numbers of elementary operations associated with the negative solution to Hilbert's 10th problem, and so k is going to be, at least, in the 100s.

On the other hand, we suspect that if k is small, then this existential problem is decidable.

To test the waters, we considered the case k = 1. We established decidability, with a fairly elaborate detailed analysis. k = 2 will be very considerably harder, but probably achievable. For k = 3, we may already be intractable territory, and be forced to settle for decidability of fragments.

GENERAL METHODOLOGY

Universal, or dually, Existential, problems based on string operations are fundamental mathematically, and have existing and potential importance for formal methods.

Undecidability is the norm when there is a reasonably healthy dose of primitives.

How do we cope?

- Examine cases generated by real world examples. Spot a fragment of the problem, and prove decidability.
- And/or put the problem in a convenient normal form, and focus on cases where some of the parameters are very small. Slowly increase them.