Decision Problems in Strings and Formal Methods

by
Harvey M. Friedman
Ohio State University
friedman@math.ohio-state.edu
http://www.math.ohio-state.edu/%7Efriedman/

Theory/POP Seminar
School of Computer Science
Carnegie Mellon University
March 27, 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.

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.

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.

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_i, s_i, t_i, p_i is either a variable or an element of A*, each x_i is a variable among the r_i, s_i, t_i, p_i, a_i is a variable among the r_i, s_i, t_i, p_i 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_i, s_i, t_i, p_i is either a variable or an element of A*, each x_i is a variable among the r_i, s_i, t_i, p_i, a_i is a variable among the r_i, s_i, t_i, p_i 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.