SOME ALGORITHMS FOR STRINGS WITH APPLICATIONS TO PROGRAM VERIFICATION

by
Harvey M. Friedman*
September 7, 2009

ABSTRACT. We present an algorithm for deciding the validity of certain universal sentences in a three sorted language involving integral linear arithmetic, a linearly ordered set of objects, and finite strings of objects. The language supports a number of basic operations, including string concatenation and string length. We analyze the complexity of the algorithm. We explore the boundary between decidability and undecidability by establishing the undecidability of certain expansions of this language.

1. THE THREE SORTED LANGUAGE L_1.

L1 has three sorts: integers, objects, and (finite) strings (of objects).

The integer sort will always be interpreted as Z, equipped with linear arithmetic. The objects can be any nonempty linearly ordered set. The strings will always be interpreted as the finite strings of these objects, including the empty string. Thus the interpretations of L1 involve only the choice of the nonempty linearly ordered set of objects.

We develop algorithm for determining whether certain universal sentences in this language are valid; i.e., are true in all interpretations of L_1.

We now present the syntax of the three sorted language L_1.

INT variables $n_1,n_2,\ldots$.
OBJ variables $x_1,x_2,\ldots$.
STR variables are $\alpha_1,\alpha_2,\ldots$.

We divide the symbols into five groups.

1. Integral Linear Arithmetic.

a. Binary relation symbols $<,\leq,=,\neq$ of type INT $\times$ INT.
b. Binary function symbols $+,-$ of type INT $\times$ INT $\rightarrow$ INT.

c. Unary function symbols $| \ |$,- of type INT $\rightarrow$ INT.
d. Constant symbols 0,c of type INT, where c is a nonempty string of base ten digits, not beginning with 0. (Integer constants).
e. For each $n \geq 2$, div_n,mod_n of type INT $\rightarrow$ INT. (Division/remainder for each specific modulus).

2. Linearly Ordered Objects. Binary relation symbols $<,\leq,=,\neq$ of type OBJ $\times$ OBJ.

3. String Construction and Comparison.

a. Constant symbol Λ of type STR. (Empty string).
b. Unary function symbol $| \ |$ of type STR $\rightarrow$ INT. (Length of a string).
c. Unary function symbol $< \ >$ of type OBJ $\rightarrow$ STR. (Length 1 string construction).
d. Binary relation symbols $=,\neq$ of type STR $\times$ STR.
e. Concatenation from STR $\times$ STR to STR. We will take advantage of the associativity to avoid parentheses.

4. Attributes.

a. Unary relation symbol WINC of type STR. (The successive terms in the string are weakly increasing ($\leq$)).
b. Ternary relation symbol VAL of type STR $\times$ INT $\times$ OBJ. (The i-th term of a given string is a given object, where we count positions in strings from 1 through their length).

Note that we have overloaded $<,\leq,=,\neq,| \ |$. This is harmless because of the strong typing.

The INT terms, OBJ terms, and STR terms of L1 are defined simultaneously as follows. (We will ignore parsing issues).

A. OBJ Terms. The OBJ Terms are exactly the OBJ variables.

B. INT Terms.

1. Every INT variable and INT constant is an INT term.
2. Let s',t' be INT terms and $n \geq 2$. Then s'+t', s'-t', |s'|, -s', DIV_n(t'), MOD_n(t') are INT terms.
3. Let t' be an INT term and t be a STR term. Then |t'| and |t| are INT terms.

C. STR Terms.

1. Every STR variable and Λ is a STR term.
2. Let x be an OBJ variable. Then <x> is an STR term.
3. Let s,t be STR terms. Then st is an STR term.

The atomic formulas of L1 are defined as follows.

1. Let x,y be OBJ variables. Then $x < y$, $x \leq y$, $x = y$, $x \neq y$ are atomic formulas of L1.
2. Let s,t be STR terms of L_1. Then WINC(s), $s = t$, $s \neq t$ are atomic formulas.
3. Let s be a STR term, s' an INT term, and x be an OBJ variable. Then VAL(s,s',x) is an atomic formula.
4. Let s',t' be INT terms. Then $s' < t'$, $s' \leq t'$, $s' = t'$, $s' \neq t'$ are atomic formulas.

The formulas of L_1 are defined as follows.

1. Every atomic formula is a formula.
2. Let A,B be formulas. $\neg A$, $A \vee B$, $A \wedge B$, $A \rightarrow B$, $A \leftrightarrow B$ are formulas.

Note that we do not allow quantifiers in L_1.

An interpretation of L_1 is a pair (D,<), where D is a nonempty set and < is a strict linear ordering on D. A D-assignment is a function f whose domain is the set of variables of L_1. INT variables must be assigned integers, OBJ variables must be assigned elements of D, and STR variables must be assigned finite strings from D.

We define

Val((D,<),T,f)

where T is a term of L1 and f is a D-assignment, by induction on T in the obvious way, according to the descriptions of the symbols given above.

We also define

Sat((D,<),φ,f)

where φ is a formula of L1 and f is a D-assignment, by induction on φ, again according to the descriptions of the symbols given above.

We write Sat((D,<),φ) if and only if Sat((D,<),φ,f) holds for all D-assignments f.

We say that φ is valid if and only if for all interpretations (D,<) of L1, Sat((D,<),φ).

We say that φ is satisfiable if and only if for some interpretation (D,<) of L1, Sat((D,<),φ).

We say that φ is realizable if and only if for some interpretation (D,<) of L1 and some D-assignment f, Sat((D,<),φ,f).

An INT equation is an equation between two INT terms. An OBJ equation is an equation between two OBJ variables. A STR equation is an equation between two STR terms.

2. STATEMENT OF DECIDABILITY RESULT.

As background information, the VCs (verification conditions) that have been shown to us by the RSRG group at the CSE Department, The Ohio State University (led by Professor Bruce Weide), mostly take the following form.

1) Suppose $A_1,...,A_r$ holds. Then $B_1,...,B_s$ hold.

1') $A_1 \wedge ... \wedge A_r \rightarrow B_1 \wedge ... \wedge B_s$.

Here $A_1,...,A_r,B_1,...,B_s$ are atomic formulas of L_1 without VAL.

There are some exceptions, where some of the A's, for example, have an ∨. Our algorithm handles such features, and also handles VAL.

The crucial point is this. From what we have seen, there is a strong but sensible and plausible restriction on the STR equations s = t appearing among the hypothesized A's in 1).

OBSERVED RESTRICTION. Let $s_1 = t_1$, ..., $s_i = t_i$ be STR equations among the A's. Then no STR variable appears more than once in the totality of $s_1,...,s_i,t_1,...,t_i$.

This restriction only applies to the A's, and not to the B's. It also does not apply to any STR terms that may appear inside | | or WINC or VAL.

We will broaden 1) for our decidability result, incorporating a corresponding restriction on the STR equations.

For our formulation of these results, we use the notion "positive and negative occurrences of letters in formulas in propositional calculus". This definition immediately extends to "positive and negative occurrences of atomic formulas in quantifier free formulas in languages such as L_1".

i. p is a positive occurrence in p.
ii. The positive (negative) occurrences in $A \wedge B$, $A \vee B$ are the positive (negative) occurrences in A,B.
iii. The positive occurrences in $A \rightarrow B$ are the negative occurrences in A and the positive occurrences in B.
iv. The negative occurrences in $A \rightarrow B$ are the positive occurrences in A and the negative occurrences in B.
v. The positive (negative) occurrences in $\neg A$ are the negative (positive) occurrences in A.

Let A be a formula of L_1. We say that A has the positive (negative) STR equation restriction if and only if the following holds.

Let $s_1 = t_1, \ldots, s_i = t_i$ be a list of all STR equations that have some positive (negative) occurrence in A.
Then every STR variable appears at most once in the totality of $s_1,\ldots,s_i,t_1,\ldots,t_i$.

THEOREM 2.1. There is a decision procedure for determining the validity of all formulas A of L_1, obeying the negative STR equation restriction. Moreover, the problem is co NP complete, and conveniently reducible to the validity of quantifier free formulas in linear integer arithmetic.

It is conceptually clearer to prove the following dual result, which is obviously equivalent.

THEOREM 2.2. There is a decision procedure for determining the satisfiability of all formulas A of L_1, under some assignment, obeying the positive SEQ equation restriction. Moreover, the problem is NP complete, and conveniently reducible to the satisfiability of quantifier free formulas in linear integer arithmetic.

We first show that we need only consider the one special interpretation where the (D,<) is simply (Z,<).

LEMMA 2.3. A formula of L_1 is valid in the (Z,<) interpretation if and only if it is valid in all (D,<) interpretations if and only if it is valid in all (D,<) interpretations with D finite.

Proof: Suppose the formula A fails in some (D,<) interpretation under some (D,<) assignment, f. Since there are only finitely many variables in the formula, A fails in some (D,<) interpretation under some (D,<) assignment, f, where D is finite. We can assume that D is some {1,...,n}, $n \geq 1$, and extend f in any way to be a (Z,<) assignment. This establishes that A fails in the (Z,<) interpretation under some (Z,<) assignment. QED

In light of Lemma 2.3, we only use the (Z,<) interpretation. We thus speak of truth or falsity of sentences, and truth or falsity of formulas under assignments (which are understood to be (Z,<) assignments).

In fact, we will use the word "satisfiable" to mean true under some (Z,<) assignment in the (Z,<) interpretation.

3. DISJUNCTIVE NORMAL FORM, AND T_0(A).

Our goal is to give a decision procedure for the satisfiability of formulas A of L_1, under the (Z,<) interpretation. This suffices to establish the decision procedure called for in Theorem 2.2, via Lemma 2.3.

Let A be a formula of L_1. We put A into disjunctive normal form as usual. But we need to go further and drive many negation signs in as follows. Replace

$\neg x = y$ by $x \neq y$.
$\neg x \leq y$ by $y < x$.
$\neg x < y$ by $y \leq x$.
$\neg x \neq y$ by $x = y$.
$\neg s' = t'$ by $s' \neq t'$.
$\neg s' \leq t'$ by $t' < s'$.
$\neg s' < t'$ by $t' \leq s'$.
$\neg s' \neq t'$ by $s' = t'$.
$\neg s = t$ by $s \neq t$.
$\neg s \neq t$ by $s = t$.

Let the resulting formula be B. Of course, negation signs will remain in front of WINC and VAL. Here x,y are OBJ variables and s,t are STR terms.

$T_0(A)$ is a finite labeled tree constructed as follows. The label of the root is A. For each disjunct φ of B, the root has one son labeled with φ.

It is obvious that A is satisfiable if and only if some leaf of $T_0(A)$ is satisfiable.

Furthermore, it is clear that each disjunct of B has the positive STR equation restriction - henceforth called the variable restriction.

3. TREE CONSTRUCTION AND OVERVIEW.

We start with a formula A with the positive STR equation restriction, and the finite tree $T_0(A)$. We wish to test A for satisfiability.

Each leaf of $T_0(A)$ is labeled with a conjunction of the form

$$D_1 \wedge \ldots \wedge D_k,\ k \geq 1$$

where the D's are of the forms

*)
$x < y$
$x \leq y$
$x = y$
$x \neq y$
$s = t$
$s \neq t$
$s' < t'$
$s' \leq t'$
$s' = t'$
$s' \neq t'$
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.

In section 4, we build a tree T[A], which extends the tree $T_0(A)$, although some vertices in $T_0(A)$ may have their labels changed during the construction process.

We will show that, once again, the satisfiability of A is equivalent to the existence of a satisfiable leaf of T(A).

It will be the case that the labels of the leaves of T(A) are of the form

$$E_1 \wedge \ldots \wedge E_p$$

where the E's are of the forms

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

where x,y are OBJ variables, s',t' are INT terms, and α is an STR variable. There is no Λ, <x>, and no concatenation.

In section ?, we will show how to reduce the satisfiability of any conjunction of the form μ) to an existential sentence in Presburger arithmetic. This is a well known NP complete problem that has many practical implementations in use and in the literature.

Thus we have reduced the original problem of the satisfiability of A, to the truth of an existential sentence in Presburger arithmetic.

There are a number of implementation issues that arise. From the theoretical standpoint, the focus is on

i. The size of the tree T(A).
ii. The size of the labels of the leaves of the tree T(A).

An implementation of this algorithm has been constructed at the RSRG group of the CSE Department at The Ohio State

University in collaboration with a group at Clemson University. See ... This software has been applied to many VC's (verification conditions) arising in connection with programs for string processing. The implementation works well in practice.

4. RULES AND T(A).

We now present our rules for the construction of the finite tree T(A), in four natural groups. Each rule states the form(s) to which it can be applied, followed by the action to be taken. A particular instance of (one of the) form(s) is located in a particular leaf of the current tree. The remove or replace action takes place at that location.

The rules calling for splits are 1.11, 1.12, 3.3, 3.4, 4.3, 4.5, 4.6. Splits are implemented as follows. First sons are created for each of the (at most 3) splits, with the label of the located leaf copied. Then the chosen instance is replaced accordingly in the copied leaves.

As the rules are implemented, in any manner whatsoever, the tree is transformed, starting with T0(A). We prove that no matter how the rules are implemented, the process terminates - i.e., no rules can be applied. This nondeterministic process results in a finite tree T(A) - which may not be unique.

1.1. Λ as proper subterm of a string term. Remove Λ.

1.2. $s = \Lambda$ or $\Lambda = s$, where s has an OBJ variable. Replace conjunction by $1 = 0$.

1.3. $s = \Lambda$ or $\Lambda = s$, 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. $\Lambda = \Lambda$. Remove.

1.5. $|\Lambda|$. Replace by 0.

1.6. $\alpha = t$ or $t = \alpha$. Remove, and replace every occurrence of α by t.

1.7. $\alpha \neq \Lambda$ or $\Lambda \neq \alpha$. Replace by $|\alpha| \neq 0$.

1.8. $\langle x\rangle \neq \Lambda$ or $\Lambda \neq \langle x\rangle$. Remove.

1.9. $\neg$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 $|t_1'| + ... + |t_n'|$, where $t_1',...,t_n'$ are the components of t, from left to right. Then replace each summand |<x>| by 1, $|\Lambda|$ by 0.

1.11. $\neg$VAL(s,t',x). Split with $\{|s| < t'\}$, $\{t' < 1\}$, {VAL(s,t',y), $x \neq y$}, where y is a new variable.

1.12. $s \neq t$, where this inequation has at least one variable. Split with $\{|s| \neq |t|\}$, {VAL(s,n,x), VAL(t,n,y), $x \neq y$}, where n,x,y are new variables.

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 $S_1' \cup ... \cup S_{p-1}'$.

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(αt,s',x). Split with {s' = 1, VAL(α,s',x)}, {s' > 1, VAL(t,s'-$|\alpha|$,x)}.

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, $\alpha = \Lambda$}, {<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. $\langle x\rangle s = \alpha t$ or $\alpha t = \langle x\rangle s$. Split with $\{\alpha = \Lambda, \langle x\rangle s = t\}$, $\{s = \beta t, \alpha = \langle x\rangle\beta\}$, 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 $\langle x\rangle\beta$.

4.6. $\alpha s = \beta t$. Split with $\{|\alpha| \leq |\beta|, \beta = \alpha\gamma, s = \gamma t\}$, $\{|\beta| < |\alpha|, \alpha = \beta\gamma, \gamma s = t\}$, where γ is a new STR variable. Follow the first split by replacing all occurrences of β by $\alpha\gamma$. Follow the second split by replacing all occurrences of α by $\beta\gamma$.

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.

Proof: It suffices to show that the conjunction(s) created by a single application of these rules to a conjunction with the variable restriction, still has (have) the variable restriction.

To verify this, we need only look at those rules which may create new STR equations. These are 1.7, 4.2 - 4.6.

1.7. After removal, α does not occur in any STR equation, and so no new STR equations are created.
4.2. Note that t is part of the equation being replaced.
4.3. In the first (second) son, the new STR equations either have an STR variable removed, or is $\langle x\rangle = t$ or $t = \Lambda$, where t is part of the STR equation being replaced.
4.4. The new equation is $s = t$, which is a part of the STR equation being replaced.
4.5. In the first son, the new STR equations either have an STR variable removed, or is part of the STR equation being replaced. In the second son, the new variable β occurs just once, and s,t are part of the STR equation being replaced. Also α isn't replaced in any STR equations in the second son.
4.6. In the first son, β is not replaced in any STR equations, and $s = \gamma t$ is part of the STR equation being replaced. The same is true in the second son.

QED

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 $E_1 \wedge \ldots \wedge E_k$ be the label of a leaf of T(A). Then the E's are of the forms

μ)
$x < y$
$x \leq y$
$x = y$
$x \neq y$
$s' < t'$
$s' \leq t'$
$s' = t'$
$s' \neq t'$
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.

Proof: By 1.9, 1.11, and 1.12, there are no ¬WINC, ¬VAL, $s \neq t$. Now let t be a maximal STR term that appears. We have the following cases related to the occurrence of t.

case 1. |t|. By 1.10, t is an STR variable.

case 2. VAL(t,t',x). By 3.1 - 3.4, t is an STR variable.

case 3. WINC(t). By 2.1, 2.2, t is an STR variable.

case 4. s = t or t = s. If Λ appears in s or t, then by 1.1, s or t is Λ. By 1.2 - 1.4, this is impossible. By 4.1 - 4.3, s,t do not begin with any <x>. Hence s,t begin with an STR variable. By 1.7, s,t are not STR variables. Now apply 4.6 to arrive at a contradiction.

QED

LEMMA 4.3. Let T be a 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.

Proof: It suffices to show that the rules have the following properties. If a rule does not involve splitting, then the satisfiability of the relevant conjunction is equivalent to the satisfiability of the resulting conjunction. If a rule does involve splitting, then the satisfiability of the relevant conjunction is equivalent to the satisfiability of at least one of the conjunctions that arise. This is verified by inspection. QED

Let a conjunction of atomic formulas be given, satisfying condition *) of section 3. 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.

Proof: We go through each rule.
1.1. Q_2 lowered; rest unchanged.
1.2. Q's become all 0's.
1.3. Q_1 lowered.
1.4. Q_2 lowered; Q_1 unchanged.
1.5. Q_2 lowered; Q_1 unchanged.
1.6. Q_1 lowered.
1.7. Q_1 lowered.
1.8. Q_1 lowered.
1.9. Q_1 lowered; Q_2 unchanged.
1.10. Q_2 lowered; Q_1 unchanged.
1.11. In each son, Q_1 lowered.
1.12. In each son, Q_1 lowered.
2.1. Q_2 lowered; Q_1 unchanged.
2.2. Q_2 lowered; Q_1 unchanged.
3.1. Q's become all 0's.
3.2. Q_2 lowered; Q_1 unchanged.
3.3. In each son, Q_2 lowered; Q_1 unchanged.
3.4. In each son, Q_1 lowered; Q_2 unchanged.
4.1. Q_1 lowered; Q_2 unchanged.
4.2. Q_1 lowered.
4.3. In each son, Q_1 lowered.

4.4. Q_1 lowered; Q_2 unchanged.
4.5. In each son, Q_1 lowered.
4.6. In each son, Q_1 lowered.

QED

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

We need some further work to get a decent estimate on the number of leaves of T(A) and the size of the labels of the leaves of T(A). In particular, we need to have an estimate as to how much Q_2 may go up when Q_1 goes down, as we implement the rules. We do not expect any serious difficulties in developing decent exponential, or double exponential, estimates, but postpone this investigation.

5. SATIFIABILITY OF LEAVES OF T(A).

We have reduced the satisfiability of A to the existence of a satisfiable leaf of T(A).

Let $E_1 \wedge \ldots \wedge E_k$ be a conjunction, where the E's are of the forms

$\mu)$
$x < y$
$x \leq y$
$x = y$

$x \neq y$
$s' < t'$
$s' \leq t'$
$s' = t'$
$s' \neq t'$
$\mathrm{WINC}(\alpha)$
$\mathrm{VAL}(\alpha,s',x)$

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.

Let the variables appearing in the conjunction be

i. INT variables $n_1,\ldots,n_p$.
ii. OBJ variables $x_1,\ldots,x_q$.
iii. STR variables $\alpha_1,\ldots,\alpha_r$.

We wish to convert

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

to an equivalent existential sentence in Presburger arithmetic.

It is convenient to rewrite 1) in the form

2) $(\exists n_1,\ldots,n_p,x_1,\ldots,x_q \in Z)(\exists\beta_1,\ldots,\beta_i,\gamma_1,\ldots,\gamma_j \in Z^*)(\mathrm{WINC}(\beta_1) \wedge \ldots \wedge \mathrm{WINC}(\beta_r) \wedge \mathrm{VAL}(\delta_1,s_1,v_1) \wedge \ldots \wedge \mathrm{VAL}(\delta_a,s_a,v_a) \wedge \mathrm{VAL}(\nu_1,t_1,w_1) \wedge \ldots \wedge \mathrm{VAL}(\nu_b,t_b,w_b) \wedge \varphi(n_1,\ldots,n_p,x_1,\ldots,x_q))$

where the δ's are among the β's, the ν's are among the γ's, the s's are INT terms in $n_1,\ldots,n_p$, and the v's and w's are among the x's. Also φ is the conjunction of the E's that do not mention STR variables. Note that the conjuncts of φ are either inequalities between x's or atomic formulas involving only the n's.

Our task is to eliminate the existential quantifiers over the β's and γ's. We first divide the displayed VAL clauses into equivalence classes according to "having the same STR variable". And these equivalence classes are divided into two groups - those where the STR variable is among the β's, and those where the STR variable is among the γ's.

Each one of these equivalence classes gives rise to a set of formulas not mentioning the STR variable. First consider an equivalence class

$$VAL(\gamma,t_1,w_1),\ldots,VAL(\gamma,t_c,w_c).$$

This equivalence class gives rise to the implications

$$t_d = t_e \rightarrow w_d = w_e$$

where $1 \leq d \leq e \leq c$.

Now consider an equivalence class

$$VAL(\beta,t_1,w_1),\ldots,VAL(\beta,t_c,w_c).$$

This equivalence class gives rise to the implications

$$t_d < t_e \leftrightarrow w_d < w_e$$

where $1 \leq d,e \leq c$. Each equivalence class among the VAL clauses are replaced in 2) by these associated formulas. Then the existential STR quantifiers are dropped, leaving an existential Presburger sentence equivalent to 1) above.

____

* This material is based upon work supported by the National Science Foundation under Grants No. DMS-0701260 and CCF-0811737. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.