A Decision Procedure for Regular Membership and Length Constraints over Unbounded Strings*

Tianyi Liang[1], Nestan Tsiskaridze[1], Andrew Reynolds[2], Cesare Tinelli[1], and Clark Barrett[3]

[1] Department of Computer Science, The University of Iowa
[2] École Polytechnique Fédérale de Lausanne
[3] Department of Computer Science, New York University

Abstract. We prove that the quantifier-free fragment of the theory of character strings with regular language membership constraints and linear integer constraints over string lengths is decidable. We do that by describing a sound, complete and terminating tableaux calculus for that fragment which uses as oracles a decision procedure for linear integer arithmetic and a number of computable functions over regular expressions. A distinguishing feature of this calculus is that it provides a completely algebraic method for solving membership constraints which can be easily integrated into multi-theory SMT solvers. Another is that it can be used to generate symbolic solutions for such constraints, that is, solved forms that provide simple and compact representations of entire sets of complete solutions. The calculus is part of a larger one providing the theoretical foundations of a high performance theory solver for string constraints implemented in the SMT solver CVC4.

1 Introduction

The study of word algebra and regular expressions has a long history in mathematics and computer science. There has been much renewed interest lately for these topics within the software verification and computer security communities because of the increasing importance of reasoning about character strings and regular expressions when proving safety properties or trying to detect security violations in programs that process string values.

To support these applications, several systems have been developed recently that check the satisfiability of constraints over a rich set of string operations including string equalities and inequalities, string length, regular language membership, and additional functions over strings besides string concatenation [33,1,19,29]. A lot of this work focuses on generally (refutation) incomplete methods to detect the unsatisfiability of these constraints, a practical approach for making progress in program analysis applications. A major difficulty in providing complete methods is that any reasonably comprehensive theory of character strings

* This work was partially funded by NSF grants #1228765 and #1228768.

is undecidable [7,22,26]. However, several more restricted, but still quite useful, theories of strings do have a decidable satisfiability problem. These include any theories of fixed-length strings, which are trivially decidable because their domains are finite, but also some fragments over unbounded strings (e.g., word equations [21,24]). Recent research has focused on identifying decidable fragments suitable for program analysis and, more crucially, on developing efficient solvers for them.

In previous work, we described a comprehensive approach, based on algebraic techniques and described abstractly as a calculus, to reason efficiently about quantifier-free formulas in a rich theory of unbounded strings with length and regular language membership [19]. And based on that approach, we constructed an efficient string solver, fully integrated into the multi-theory SMT solver CVC4. The calculus developed in that work is both refutation and solution sound but refutation incomplete.

Contribution and significance We have developed an improved version of the calculus presented in [19] that is also complete and terminating over a restriction of the general language to membership and length constraints. In this paper, we present a simplified version of that calculus which can be used to prove that the fragment in question is decidable. Strictly speaking, this decidability result is not new, as it is implicitly implied by some recent results from Abdulla *et al.* [1], although that work does not mention the result. We provide a full proof based on the calculus presented here. This contribution is significant not only because of the importance of the fragment but also for the following reasons. First, contrary to previous approaches for solving membership constraints which rely on reductions to finite state automata problems, our approach is completely algebraic and works directly with regular expressions. This facilitates the creation of efficient *incremental* solvers which can be more easily incorporated into modern SMT solvers since they do not rely on eager conversion to automata problems. Second, our completeness argument shows how to produce *symbolic solutions* for satisfiable problems with regular membership constraints, that is, intensional representations of (possibly infinite) sets of concrete solutions. This is useful for security analysis applications like filter generation and automatic exploit generation (AEG), where any assignment satisfying the constraints generated from a program is a security exploit. A symbolic solution enables AEG applications, for example, to generate fewer, more general exploits, thus also reducing the number of exploits that would need to be examined by a user.

Although our eventual goal is overall efficiency in practice, the calculus presented here focuses (for simplicity) on proving the decidability result. As a consequence, it uses a few auxiliary functions that apply generally inefficient eager (but algebraic) conversions from and to regular expressions. We plan to present in future work a version of the calculus that lifts these conversions to a set of additional derivation rules, making them amenable to lazy and selective application based on search heuristics.

1.1 Related work

There have been a number of different approaches for solving string constraints with regular expressions. The earliest and perhaps most established approach is based on reductions to automata decision problems. One of these was implemented in the system DPrle, used to check programs against SQL injection vulnerabilities [13]. The approach followed in that system has the strong limitation of imposing an upper bound on the length of string variables, a hard to overcome drawback shared by various later works. This approach was later improved by the same author with a method for generating automata lazily from the input problem which does not requiring any *priori* length bounds [14]. At the same time, a comprehensive set of algorithms and data structures for performing fast automata operations was developed to support constraint solving over strings, for instance in [12].

Current automata-based approaches to reason about regular expressions can be divided in two classes depending on whether their transitions processing a single character a time (e.g., [9,32]) or a set of them (e.g., [30,31,14]). Most of the tools based on these approaches offer very limited support to reason about constraints mixing strings and other data types. Also, automata refinement may constitute a performance bottleneck, even though it is very useful in solving membership constraints. Further discussion can be found in [10,18]. Other approaches for solving regular expression constraints are based on reductions to other theories, such as bit-vectors [15] or linear integer arithmetic constraints [28], [7], and using constraint solvers for those theories.

Three notable systems that solve regular membership constraints are Rex [31,30], Mona [11] and the Java String Analyzer (JSA) [8]. Rex too is based on automata. In contrast to the work described in [14] where each transitions covers an integer interval, Rex encodes strings as symbolic finite automata (SFA) first. Each SFA transition uses a logical predicate over linear arithmetic to represent a set of character-level candidates. This allows Rex to encode transitions as SMT constraints which it then sends to an SMT solver for a model. This approach provides an efficient encoding for solving membership constraints, however, it currently does not support mixed constraints over additional theories.

Mona is a solver for monadic second-order logic with built-in support for string constraints. Although Mona is an automata-based, it uses Multi Terminal BDDs to represent automata. This kind of implementation requires sophisticated engineering techniques (see [16]) which make it difficult to build in additional theories to support solving of combined constraints. Pisa [27] is another string solver based monadic second-order logic. However, the language of Pisa is rather restrictive, e.g., no binary operations between two variables are allowed.

JSA is geared specifically to Java string constraints. It first translates them to a flow graph, and then analyzes the graph by converting it to a context-free language. This language is approximated with the Mohri-Nederhof algorithm to a regular one and encoded as a multi-level automaton. Compared to our work, JSA focuses exclusively on Java string analysis, approximation, and automaton conversion, while our approach does not depend on any particular language, and solves string constraints natively with no approximations.

It is well-known that regular languages are closed under common operations (e.g., concatenation, union, intersection, complementation); however, the complexity of performing most of these operations is high as a consequence of the high complexity of the corresponding membership problem. For example, membership in the intersection of two regular languages is PSPACE-complete [17]. Thus, in practice many procedure implementing regular language operations are approximate (e.g., [6,25]). In contrast, the calculus we present here does not approximate.

Our calculus decides a fragment that combines regular membership constraints with string length constraints. To the best of our knowledge, there are no explicit claims about the decidability of this fragment. The work in [1] implies that the fragment is indeed decidable, although the paper contains no proof, or mention, of this. The method described in that paper replaces all characters in regular expressions with a single arbitrary character, and reduces the expression to their Parikh images [23], generating a set of *semi-linear* integer constraints which can then be checked for satisfiability using any linear arithmetic solver. Since our approach does not use rely on approximations it can build a model directly when the constraints are satisfiable. This part of work our has some similarities with the Parikh image described in [4], although we developed it independently.

1.2 Formal preliminaries

We work in the context of many-sorted first-order logic with equality ($\approx$). We assume the reader is familiar with the notions of many-sorted signature, term, literal, formula, free variable, interpretation, and satisfiability of a formula in an interpretation (see, e.g., [5] for more details). A *theory* is a pair $T = (\Sigma, \mathbf{I})$ where Σ is a signature and $\mathbf{I}$ is a class of Σ-interpretations, the *models* of T, that is closed under variable reassignment. If $\mathcal{I}$ is an interpretation and t is a term, we denote by $t^{\mathcal{I}}$ the value of t in $\mathcal{I}$. A Σ-formula φ is *T-satisfiable* (resp., *T-unsatisfiable*) if it is satisfied by some (resp., no) interpretation in $\mathbf{I}$. A set Γ of formulas *entails in T* a Σ-formula φ, written $\Gamma \models_T \varphi$, if every interpretation in $\mathbf{I}$ that satisfies all formulas in Γ satisfies φ as well. The set Γ is *satisfiable in T* if $\Gamma \not\models_T \bot$ where $\bot$ is the universally false atom. If e is a term or a formula, we denote by $\mathcal{V}(e)$ the set of e's free variables, extending the notation to sets of terms or formulas as expected. Two Σ-formulas φ and ψ are *T-equisatisfiable* if for every model $\mathcal{I}$ of T that satisfies one, there is a model of T that satisfies the other and differs from $\mathcal{I}$ at most over the free variables not shared by φ and ψ.

2 A theory of strings and regular language membership

We consider a theory T_{LR} of strings with length and regular language membership constraints over a signature Σ_{LR} with three sorts, Str, Int, and Lan, and an infinite

$\epsilon : \mathsf{Str}$	$\_\cdot\_ : \mathsf{Str}\times\mathsf{Str}\to\mathsf{Str}$	$c : \mathsf{Str}$ for all $c\in\mathcal{A}$	$\lvert\_\rvert : \mathsf{Str}\to\mathsf{Int}$
$\mathsf{Ch} : \mathsf{Lan}$	$\_\cdot\_ : \mathsf{Lan}\times\mathsf{Lan}\to\mathsf{Lan}$	$\_\sqcup\_ : \mathsf{Lan}\times\mathsf{Lan}\to\mathsf{Lan}$	$\_^{*} : \mathsf{Lan}\to\mathsf{Lan}$
$\varnothing : \mathsf{Lan}$	$\_\,\mathsf{in}\,\_ : \mathsf{Str}\times\mathsf{Lan}$	$\_\sqcap\_ : \mathsf{Lan}\times\mathsf{Lan}\to\mathsf{Lan}$	$\ulcorner\_\urcorner : \mathsf{Str}\to\mathsf{Lan}$

Fig. 1. Basic set of string and regular expression function and predicate symbols.

$$(\_)^{(\_)} : \mathsf{Lan}\times\mathsf{Int}\to\mathsf{Lan} \qquad \mathsf{sh} : \mathsf{Lan}\times\mathsf{Lan}\times\mathsf{Lan}\to\mathsf{Lan}$$

Fig. 2. Additional regular expression function symbols.

set of variables for each of these sorts. This theory is essentially the theory of a single many-sorted structure and its models differ only on how the variables are interpreted. All models of T_{LR} interpret Int as the set of integer numbers, Str as the language $\mathcal{W}$ of all words over some fixed finite alphabet $\mathcal{A}$ of *characters*, and Lan as the power set of $\mathcal{W}$. The signature includes: the usual symbols of linear integer arithmetic, interpreted as expected; all the elements of $\mathcal{W}$ as constant symbols, or *string constants*, interpreted as themselves; and all the function symbols given in Figure 1 with their rank. In that figure, the two $\cdot$ symbols denote word concatenation and language concatenation, respectively; $|\_|$ denotes word length; and $\ulcorner\_\urcorner$ denotes the singleton set constructor, mapping each word $w\in\mathcal{W}$ to the language $\{w\}$; the symbols ϵ, Ch, $\varnothing$, and in respectively denote the empty word, the language of one-character words, the empty language, and the language membership predicate; the symbols $\sqcup$, $\sqcap$, and $(\_)^*$ respectively denote language union, intersection and Kleene closure.

We call a *string term* any term of sort Str or of the form $|s|$; an *arithmetic term* any term of sort Int all of whose occurrences of $|\_|$ are applied to a variable; and a *regular expression* any *variable-free* term of sort Lan. A string term is *atomic* if it is a variable or a string constant. An *arithmetic constraint* is a (dis)equality $(\neg)u \approx v$ or an inequality $u \geq v$ where u and v are arithmetic terms. A *membership constraint* is a literal of the form $(\neg)(s \in r)$ where s is a string term and r is a regular expression. A T_{LR}*-constraint* is an arithmetic or a membership constraint. Note that we do not consider here equalities between terms of sort Str. Also note that if x is a string variable, $|x|$ is both a string and an arithmetic term. By the definition of T_{LR}, a regular expression r is interpreted as the same language in every model of T_{LR}. We call that the *language generated by* r and denote it by $\mathcal{L}(r)$.

Expanding the language The calculus we present later is able to compute a *solved form* for a satisfiable input set of T_{LR}-constraints with string variables $x_1,\ldots,x_n$. This solved form consists of a set $\{x_i \;\mathsf{in}\; q_i\}_{i=1,\ldots,n}$ of membership constraints where, for all i, q_i is a *solved-form* term, a term of sort Lan over integer variables and a signature that includes string constants, the symbols Ch, $\cdot$ and $\ulcorner\_\urcorner$ from Figure 1, and the two function symbols from Figure 2. Note that the latter two symbols are not in the (input) language of T_{LR}-constraints; they are used only in solved forms. We expand the models of T_{LR} to these two symbols so that the following holds.

$$\begin{array}{rcl@{\qquad}rcl@{\qquad}rcl}
(s_1 \cdot s_2) \cdot s_3 & \to & s_1 \cdot (s_2 \cdot s_3) & s \cdot \epsilon & \to & s & \epsilon \cdot s & \to & s \\
|s_1 \cdot s_2| & \to & |s_1| + |s_2| & |c| & \to & 1 & |\epsilon| & \to & 0 \\
r_1 \cdot (r_2 \sqcup r_3) & \to & (r_1 \cdot r_2) \sqcup (r_1 \cdot r_3) & \ulcorner \epsilon \urcorner \cdot r & \to & r & \varnothing \cdot r & \to & \varnothing \\
(r_1 \sqcup r_2) \cdot r_3 & \to & (r_1 \cdot r_3) \sqcup (r_2 \cdot r_3) & r \cdot \ulcorner \epsilon \urcorner & \to & r & r \cdot \varnothing & \to & \varnothing \\
\ulcorner s_1 \urcorner \cdot \ulcorner s_2 \urcorner & \to & \ulcorner s_1 \cdot s_2 \urcorner & r^{**} & \to & r^* & \ulcorner \epsilon \urcorner^* & \to & \ulcorner \epsilon \urcorner \\
r \sqcup r & \to & r & (r \sqcup \ulcorner \epsilon \urcorner)^* & \to & r^* & \varnothing^* & \to & \ulcorner \epsilon \urcorner \\
r_1 \sqcap r_2 & \to & \pi(r_1, r_2) & \varnothing \sqcup r & \to & r & \varnothing \sqcap r & \to & \varnothing
\end{array}$$

Fig. 3. Term normalization rules, defined modulo commutativity of $\sqcup$ and $\sqcap$; $\pi(r_1, r_2)$ is the regular expression computed by the function π defined in Figure 8.

- For all integers n and regular expressions r, $\mathcal{L}(r^n) = \{\epsilon\}$ if $n \leq 0$ and $\mathcal{L}(r^n) = \mathcal{L}(r \cdot r^{n-1})$ otherwise.
- For all regular expressions r, r', q, $\mathcal{L}(\mathsf{sh}(r, r', q)) = \{w_1 w'_1 \cdots w_n w'_n \in \mathcal{L}(q) \mid n > 0,\ w_1 \cdots w_n \in \mathcal{L}(r),\ w'_1 \cdots w'_n \in \mathcal{L}(r')\}$.[4]

Intuitively, the strings generated by $\mathsf{sh}(r, r', q)$ can be obtained by *shuffling* together a word w generated by r and a word w' generated by r', as long as the resulting word is in the language generated by q. Shuffling is achieved by breaking w and w' arbitrarily into n segments and merging the two lists of segments together.

Notational conventions We use c, d to denote *character constants*, that is, string constants of length one; l for arbitrary string constants; x for string variables; s, t for string terms; z for integer variables; u, v for arithmetic terms; and q, r for regular expressions. We will omit applications of the $\ulcorner \_ \urcorner$ operator, treating (variable-free) terms of sort Str as the corresponding regular expression. When convenient, we will treat a multi-character constant l as the term $c \cdot l'$ where c is the first character of l and l' is the rest of l. We will write $\models_{\mathsf{LR}}$ instead of $\models_{T_{\mathsf{LR}}}$.

3 A calculus for constraint satisfiability in T_{LR}

We are interested in checking the satisfiability in T_{LR} of finite sets of T_{LR}-constraints as defined in Section 2. In this section, we describe a tableaux-style calculus that can be used to construct a decision procedure for this problem.

Configurations The calculus applies to a finite set of T_{LR}-constraints with the goal of determining their T_{LR}-satisfiability. It consists of derivation rules that operate over *configurations*. A configuration is either the distinguished configuration unsat or a tuple of the form $\langle A, R, V \rangle$, where: A is a set of arithmetic constraints and implications of the form $z_1 \approx 0 \Rightarrow z_2 \approx 0$; R is a set of *positive* membership constraints; and V is a set of membership constraints in solved form.

[4] Any of the words $w_1, \ldots, w_n, w'_1, \ldots, w'_n$ in the definition of sh could be empty. We use juxtaposition to denote word concatenation at the semantic level.

$$\textsf{A-Conflict}\ \frac{\mathsf{A} \models_{\mathsf{LIA}} \bot}{\mathsf{unsat}} \qquad \textsf{EmptyS}\ \frac{\epsilon \mathrel{\mathsf{in}} r \in \mathsf{R} \quad \mathrm{not}\ \varepsilon(r)}{\mathsf{unsat}} \qquad \textsf{EmptyR}\ \frac{s \mathrel{\mathsf{in}} \varnothing \in \mathsf{R}}{\mathsf{unsat}}$$

$$\textsf{Assign-1}\ \frac{\mathsf{R} = R,\ x \mathrel{\mathsf{in}} l}{\mathsf{A} := \mathsf{A},\ |x| \approx |l|{\downarrow} \quad \mathsf{R} := (R\{x \mapsto l\}){\downarrow} \quad \mathsf{V} := \mathsf{V}, x \mathrel{\mathsf{in}} l}$$

$$\textsf{Assign-2}\ \frac{\mathsf{R} = R,\ x \mathrel{\mathsf{in}} r \quad x \notin \mathcal{V}(R) \quad \mathrm{top}(r) \notin \{\sqcup, \varnothing\} \quad \gamma(r) = (q, u, A)}{\mathsf{A} := \mathsf{A},\ |x| \approx u{\downarrow}\,,\ A{\downarrow} \quad \mathsf{R} := R \quad \mathsf{V} := \mathsf{V},\ x \mathrel{\mathsf{in}} q}$$

$$\textsf{Consume-1}\ \frac{\mathsf{R} = R,\ c \mathrel{\mathsf{in}} r}{\mathsf{R} := R,\ \epsilon \mathrel{\mathsf{in}} (\partial_c\, r){\downarrow}} \qquad \textsf{Consume-2}\ \frac{\mathsf{R} = R,\ c \cdot s \mathrel{\mathsf{in}} r}{\mathsf{R} := R,\ s \mathrel{\mathsf{in}} (\partial_c\, r){\downarrow}}$$

$$\textsf{Split}\ \frac{\mathsf{R} := R,\ x \cdot s \mathrel{\mathsf{in}} r}{\|_{(r_1, r_2) \in \beta(r)}\ \mathsf{R} := R,\ x \mathrel{\mathsf{in}} r_1{\downarrow}\,, s \mathrel{\mathsf{in}} r_2{\downarrow}}$$

$$\textsf{Inter}\ \frac{\mathsf{R} := R,\ s \mathrel{\mathsf{in}} r_1,\ s \mathrel{\mathsf{in}} r_2}{\mathsf{R} := R,\ s \mathrel{\mathsf{in}} (r_1 \sqcap r_2){\downarrow}} \qquad \textsf{Union}\ \frac{\mathsf{R} := R,\ s \mathrel{\mathsf{in}} r_1 \sqcup r_2}{\mathsf{R} := R,\ s \mathrel{\mathsf{in}} r_1 \quad \| \quad \mathsf{R} := R,\ s \mathrel{\mathsf{in}} r_2}$$

Fig. 4. Derivation Rules. $R\{x \mapsto l\}$ is the result of applying the substitution $\{x \mapsto l\}$ to every term in R; $\mathrm{top}(r)$ is the top symbol of term r.

Informally, the sets A and R initially store a T_{LR}-equisatisfiable variant of the input set and progressively receive additional constraints derived by the calculus; V, which is initially empty, represents the solution computed so far (each string variable in V is associated with a set of possible values using solved-form terms).

By standard transformations, one can convert any finite set of T_{LR}-constraints into a T_{LR}-equisatisfiable set $A \cup R$ where R is a set of positive membership constraints[5] and A is a set of arithmetic constraints that includes a constraint of the form $|x| \geq 0$ for every string variable $x \in \mathcal{V}(A)$ and contains no string variables that do not occur in R. We assume that all terms in such configurations are irreducible by the rewrite system in Figure 3 which can be shown to be equivalence-preserving and terminating over Σ_{LR}-terms.[6] The rewrite system uses the auxiliary function π, closely based on one by Lu [20], which maps two regular expressions r_1 and r_2 to a regular expression that generates the same language as $r_1 \sqcap r_2$ (i.e., $\mathcal{L}(\pi(r_1, r_2)) = \mathcal{L}(r_1 \sqcap r_2)$) but contains no occurrences of $\sqcap$. If t is a Σ_{LR}-term, we denote by $t{\downarrow}$ any normal form of t with respect to the rewrite system in Figure 3, and extend this notation to sets of Σ_{LR}-terms as expected. We call a term t *normalized* if $t = t{\downarrow}$.

Without loss of generality, *we will consider for our calculus only starting configurations $\langle A, R, \emptyset \rangle$ where A, R are as above.*

The calculus assumes the availability of a procedure for checking entailment in the (decidable) theory of linear integer arithmetic ($\models_{\mathsf{LIA}}$). The only significant

[5] Each negative membership constraint $s \notin r$ can be replaced by $s \in r^{\mathrm{c}}$ where r^{c} is a regular expression generating the complement of $\mathcal{L}(r)$. This replacement is effective although current procedures for computing r^{c} are generally inefficient in practice.

[6] The system is not confluent but we do not need it to be.

deviation we require is that the procedure be able to accept terms of the form $|x|$, where x is a string variable, by treating the whole term as an arithmetic variable. In essence, the calculus models a solver for T_{LR}-constraints that is based on the cooperation of a standard subsolver for linear arithmetic constraints and a novel subsolver that processes membership constraints natively, without reduction to automata problems. This is done by processing regular expressions by means of algebraic manipulations and non-deterministic choices. The two subsolvers communicate by exchanging linear arithmetic constraints over string lengths.

Derivation rules The rules of the calculus are provided in Figure 4 in *guarded assignment form* where fields A, R, and V store, in order, the components of a current configuration $\langle A, R, V \rangle$. A derivation rule applies to a current configuration C if all of the rule's premises hold for C *and* the resulting configuration is different from C. A rule's conclusion describes how each component of C is changed, if at all. In the rules, we write S, t as an abbreviation for $S \cup \{t\}$. Rules with two or more conclusions separated by the symbol $\|$ are non-deterministic branching rules.

The derivation rules rely on several computable functions and predicates, described below and defined formally in Figures 5, 6, 7, 8, and 9, which apply to $\sqcap$-free regular expressions.

- The family of functions $(\partial_c)_{c \in \mathcal{A}}$ computes the *partial derivative* of the input with respect to character c. Concretely, $\partial_c(r)$ is a regular expression whose language is the set of all words w (including the empty one) such that $cw \in \mathcal{L}(r)$.
- The predicate ε holds exactly for those regular expressions whose language contains the empty string ϵ.
- The function γ produces three outputs from a normalized regular expression r with top symbol other than $\varnothing$ or $\sqcup$: a solved-form term q, an arithmetic term u, and a set A of arithmetic constraints over the (integer) variables in q and u. Intuitively, u and A together express constraints on the possible lengths of the words in $\mathcal{L}(r)$.
- The function β returns a finite set of regular expression pairs. Each pair $(r_1, r_2) \in \beta(r)$ is such that $\mathcal{L}(r) = \mathcal{L}(r_1 \cdot r_2)$. Moreover, $\beta(r)$ is exhaustive in the sense that for every pair of words w_1, w_2 such that $w_1 w_2 \in \mathcal{L}(r)$, there is a pair $(r_1, r_2) \in \beta(r)$ such that $w_1 \in \mathcal{L}(r_1)$ and $w_2 \in \mathcal{L}(r_2)$.

The definition of the partial derivative functions is due to Antimirov [2]; the functions γ and β are novel. Given these auxiliary predicates and functions, the calculus rules should be self-explanatory, with the possible exception of Assign-2. This rule considers a membership constraint $(x \mathsf{\ in\ } r)$ where r is not a union and (by construction) contains no occurrences of $\varnothing$ and $\sqcap$. If x occurs in no other membership constraints in the R component of the configuration, the rule uses γ to compute a solution form of $(x \mathsf{\ in\ } r)$ and stores it in the V component.

Derivation trees and derivations The rules in this calculus are used to construct derivation trees. A *derivation tree* is a tree where each node is a configuration and each non-root node is obtained from its parent node by applying

$$\begin{array}{rl} \varepsilon(r) \quad \text{iff} & (r = r_1 \cdot r_2 \text{ and } \varepsilon(r_1) \text{ and } \varepsilon(r_2)) \;\text{ or }\; r = \epsilon \;\text{ or }\; r = r_1^* \;\text{ or} \\ & (r = r_1 \sqcup r_2 \text{ and } \varepsilon(r_1)) \;\text{ or }\; (r = r_1 \sqcup r_2 \text{ and } \varepsilon(r_2)) \end{array}$$

Fig. 5. Definition of predicate ε.

$$\begin{array}{lll} \partial_c \varnothing \quad = \varnothing & \partial_c(r_1 \sqcup r_2) = \partial_c r_1 \sqcup \partial_c r_2 & \partial_c(c \cdot s) = s \\ \partial_c \epsilon \quad\; = \varnothing & \partial_c(r_1 \cdot r_2) \;\; = (\partial_c r_1 \cdot r_2) \sqcup \partial_c r_2 \quad \text{if } \varepsilon(r_1) & \\ \partial_c \mathsf{Ch} \;\, = \epsilon & \partial_c(r_1 \cdot r_2) \;\; = \partial_c r_1 \cdot r_2 \quad \text{if not } \varepsilon(r_1) & \\ \partial_c(r^*) = (\partial_c r) \cdot r^* & \partial_c(d \cdot s) \quad\; = \varnothing \qquad \text{if } c \neq d & \end{array}$$

Fig. 6. Definition of partial derivative function ∂_c.

one of the derivation rules. We call the root of a derivation tree an *initial* configuration. A branch of a derivation tree is *saturated* if no rules apply to its leaf, it is *closed* if it ends with unsat. A derivation tree is *closed* if all of its branches are closed.

A derivation tree *derives* from a derivation tree T if it is obtained from T by the application of exactly one of the derivation rules to one of T's leaves. A *derivation* is a sequence $(T_i)_{i \geq 0}$ of derivation trees such that T_0 is a one-node tree whose root is an initial configuration and T_{i+1} derives from T_i for all $i \geq 0$.

Let S be a set of Σ_{LR}-constraints. A *refutation of set S* is a derivation that starts with a one-node tree with a configuration $\langle A, R, \emptyset \rangle$ where $A \cup R$ is T_{LR}-equisatisfiable with S, and ends with a closed tree.

Example 1. Consider the satisfiable initial configuration with $\mathsf{A} = \emptyset$, $\mathsf{V} = \emptyset$, and $\mathsf{R} = \{bc \cdot x \;\mathsf{in}\; ((aa \sqcup b)^* \cdot c)^* \sqcup a \cdot c^*\}$ where x is a variable of sort String and a, b, c are characters. A derivation in the calculus can start with an application of the Union rule. In the branch $bc \cdot x \;\mathsf{in}\; a \cdot c^*$, Consume-2 will apply and replace the constraint with $c \cdot x \;\mathsf{in}\; \varnothing$ which then will be closed by EmptyR. In the branch $bc \cdot x \;\mathsf{in}\; ((aa \sqcup b)^* \cdot c)^*$, Consume-2 will be applied twice: once for b, resulting in $\mathsf{R} = \{c \cdot x \;\mathsf{in}\; (aa \sqcup b)^* \cdot c \cdot ((aa \sqcup b)^* \cdot c)^*\}$; and once for c, resulting in $\mathsf{R} = \{x \;\mathsf{in}\; ((aa \sqcup b)^* \cdot c)^*\}$. Now, by applying Assign-2 to the resulting configuration, we will have the following saturated configuration:

$$\begin{array}{l} \mathsf{A} = \{z_1 \geq 0,\; z_2 \geq 0,\; z_3 \geq 0,\; z_4 \geq 0,\; z_1 \approx 0 \Rightarrow z_2 \approx 0\} \quad \mathsf{R} = \emptyset \quad \mathsf{V} = \{x \;\mathsf{in}\; q_2\} \\ \quad \cup \{z_2 \approx z_3 + z_4,\; |x| \approx 2 * z_3 + z_4 + z_1\} \end{array}$$

where $q_2 = \mathsf{sh}(q_1, c^{z_1}, r_1)$, $q_1 = \mathsf{sh}((aa)^{z_3}, b^{z_4}, r_2)$, $r_1 = ((aa \sqcup b)^* \cdot c)^{z_1}$, $r_2 = (aa \sqcup b)^{z_2}$, and $z_1, \ldots, z_4$ are fresh variables of sort Int. The set in A is satisfiable, for instance with the variable assignment $\{z_1 \mapsto 1, z_2 \mapsto 2, z_3 \mapsto 1, z_4 \mapsto 1, |x| \mapsto 4\}$. Given this assignment one can evaluate—deterministically—the term q_2 inside out and obtain $q_2 = \{aabc, baac\}$ after evaluating q_1 to $\{aab, baa\}$. At this point, any element of q_2 is a solution for x in the original problem. As we show later, any other satisfying assignment for A will lead to a ground expression for q_2 that is guaranteed to generate a non-empty language of solutions for x. □

$$\beta(\varnothing) = \emptyset \qquad \beta(c) = \{(c, \epsilon), (\epsilon, c)\} \qquad \beta(r_1 \sqcup r_2) = \beta(r_1) \cup \beta(r_2)$$
$$\beta(\epsilon) = \{(\epsilon, \epsilon)\} \qquad \beta(\mathsf{Ch}) = \{(\mathsf{Ch}, \epsilon), (\epsilon, \mathsf{Ch})\}$$
$$\beta(r^*) = \beta(\epsilon) \cup \{(r^* \cdot r_1,\, r_2 \cdot r^*) \mid (r_1,\, r_2) \in \beta(r)\}$$
$$\beta(r_1 \cdot r_2) = \{(r_{11},\, r_{12} \cdot r_2) \mid (r_{11},\, r_{12}) \in \beta(r_1)\} \cup \{(r_1 \cdot r_{21},\, r_{22}) \mid (r_{21},\, r_{22}) \in \beta(r_2)\}$$

Fig. 7. Definition of splitting function β.

$$\pi(r, r') = \pi'(r, r', \emptyset) \qquad \pi'(r, r', C) = y_{r,r'} \text{ if } y_{r,r'} \in C \qquad \pi'(r, \varnothing, C) = \varnothing$$
$$\pi'(\epsilon, r, C) = \epsilon \text{ if } \varepsilon(r) \qquad \pi'(\epsilon, r, C) = \varnothing \text{ if not } \varepsilon(r) \qquad \pi'(\varnothing, r, C) = \varnothing$$
$$\pi'(r, \epsilon, C) = \epsilon \text{ if } \varepsilon(r) \qquad \pi'(r, \epsilon, C) = \varnothing \text{ if not } \varepsilon(r) \qquad \pi'(r, r, C) = r$$
$$\pi'(r, r', C) = r_1^* \cdot r_1' \text{ if } v_{r,r'} \notin C \text{ and } \varepsilon(r) \text{ and } \varepsilon(r') \text{ where}$$
$$(r_1, r_1') = \rho_{v_{r,r'}}(\epsilon \sqcup \textstyle\bigsqcup_{c \in \mathcal{A}} c \cdot \pi'(\partial_c\, r, \partial_c\, r', C')), \quad C' = C \cup \{v_{r,r'}\}$$
$$\pi(r, r', C) = r_1^* \cdot r_1' \text{ if } v_{r,r'} \notin C \text{ and not } (\varepsilon(r) \text{ and } \varepsilon(r')) \text{ where}$$
$$(r_1, r_1') = \rho_{v_{r,r'}}(\textstyle\bigsqcup_{c \in \mathcal{A}} c \cdot \pi'(\partial_c\, r, \partial_c\, r', C')), \quad C' = C \cup \{v_{r,r'}\}$$

$$\rho_y(\varnothing) = (\varnothing, \varnothing) \qquad \rho_y(y) = (\epsilon, \varnothing) \qquad \rho_y(r) = (\epsilon, r) \text{ if } y \notin \mathcal{V}(r)$$
$$\rho_y(r) = (r_1 \cdot r_{21}, r_{22}) \text{ if } y \in \mathcal{V}(r),\ r = r_1 \cdot r_2, \text{ and } (r_{21}, r_{22}) = \rho_y(r_2)$$
$$\rho_y(r) = (r_{11} \sqcup r_{21}, r_{12} \sqcup r_{22}) \text{ if } y \in \mathcal{V}(r),\ r = r_1 \sqcup r_2, \text{ and } (r_{i1}, r_{i2}) = \rho_y(r_i)$$

Fig. 8. Definition of intersection function π.

Example 2. Suppose we start with the unsatisfiable configuration with $\mathsf{A} = \{|x| \approx 2 * k + 1\}$, $\mathsf{R} = \{x \cdot x \;\mathsf{in}\; r\}$, and $\mathsf{V} = \emptyset$ where $r = (aaaa)^*$ and a is a character. One possibility is to apply the Split rule. Since $\beta(r) = \{(\epsilon,\, \epsilon),\, (r \cdot a,\, aaa \cdot r),\, (r \cdot a,\, aaa \cdot r),\, (r \cdot aa,\, aa \cdot r),\, (r \cdot aaa,\, a \cdot r)\}$, four branches will be created. In the first branch, $\mathsf{R} = \{x \,\mathsf{in}\, \epsilon\}$. The rule $\mathsf{Assign\text{-}1}$ can be applied, adding $x \;\mathsf{in}\; \epsilon$ to V and $|x| \approx 0$ to A. After that, the branch can be closed by $\mathsf{A\text{-}Conflict}$. In the second branch, $\mathsf{R} = \{x \;\mathsf{in}\; r \cdot a,\ x \;\mathsf{in}\; aaa \cdot r\}$ to which Inter can be applied, replacing the constraints in R with $x \,\mathsf{in}\, \varnothing$. Then the branch can be closed by EmptyS. Something, similar can be done on the fourth branch. In the third branch, R can become $\{x \;\mathsf{in}\; aa \cdot r\}$ by Inter. Then $|x| \approx 2 + 4 * z$ and $z \geq 0$ can be added to A by $\mathsf{Assign\text{-}2}$, with z a fresh integer variable. That branch can be closed by $\mathsf{A\text{-}Conflict}$, yielding a refutation of the input problem. □

4 Calculus Correctness

We prove the correctness of the calculus in by showing that (*i*) it has no infinite derivations; (*ii*) its rules preserve satisfiability in T_{LR}; (*iii*) every saturated branch in a derivation tree determines a model of T_{LR} that satisfies the initial configuration. Together with the termination of the auxiliary functions and procedures used by the calculus, this implies the decidability of the quantifier-free satisfiability problem for T_{LR}.[7]

[7] Full proofs can be found in the appendix.

$$
\begin{aligned}
\gamma(r) &= \gamma'(r, \emptyset) \\
\gamma'(l, A) &= (l, |l|\!\downarrow, A) \qquad\qquad \gamma'(\mathsf{Ch}, A) = (\mathsf{Ch}, 1, A) \\
\gamma'(r_1 \cdot r_2, A) &= (q_1 \cdot q_2, u_1 + u_2, A_1 \cup A_2) \text{ where } (q_i, u_i, A_i) = \gamma'(r_i, A) \text{ for } i = 1, 2 \\
\gamma'(r^*, A) &= (q, u, B \cup \{z_1 \geq 0\}) \text{ where } (q, u, B) = \gamma'(r^{z_1}, A) \\
\gamma'(l^z, A) &= (l^z, z \times |l|\!\downarrow, A) \qquad \gamma'(\mathsf{Ch}^z, A) = (\mathsf{Ch}^z, z, A) \\
\gamma'((r_1 \sqcup r_2)^z, A) &= (\mathsf{sh}(q_1, q_2, (r_1 \sqcup r_2)^z),\ u_1 + u_2,\ B) \\
&\qquad \text{where } B = A_1 \cup A_2 \cup \{z \approx z_1 + z_2, z_1 \geq 0, z_2 \geq 0\} \\
&\qquad\qquad (q_i, u_i, A_i) = \gamma'(r_i^{z_i}, A) \text{ for } i = 1, 2 \\
\gamma'((r_1 \cdot r_2)^z, A) &= (\mathsf{sh}(q_1, q_2, (r_1 \cdot r_2)^z),\ u_1 + u_2, A_1 \cup A_2) \\
&\qquad \text{where } (q_i, u_i, A_i) = \gamma'(r_i^z, A) \text{ for } i = 1, 2 \\
\gamma'((r^*)^z, A) &= \gamma'(q,\ u,\ B \cup \{z \approx 0 \Rightarrow z_1 \approx 0, z_1 \geq 0\}) \text{ where } (q, u, B) = \gamma'(r^{z_1}, A)
\end{aligned}
$$

Fig. 9. Definition of function γ. The letters z_1 and z_2 denote fresh integer variables variables variables variables variables.

4.1 Termination

Proving the termination of the auxiliary functions and predicates is a simple exercise.

Proposition 1. *The function π is well defined and computable over the set of all regular expressions. The predicate ε and the functions ∂_c, β and γ are well defined and computable over the set of all $\sqcap$-free regular expressions.*

By Proposition 1, every rule is effective. To prove the termination of the calculus it suffices to define a well-founded ordering of configurations and show that every rule application produces a smaller configuration along that ordering.

Proposition 2. *Every derivation in the calculus is finite.*

Proof (Sketch). One can show that every application of a derivation rule to a leaf of a derivation tree produces smaller configurations with respect to a well-founded relation $\succ$ over configurations which implies that no derivation tree can be grown indefinitely.

The relation $\succ$ is defined as follows. To each configuration $\langle A, R, V \rangle$ we associate a tuple $(\mathcal{V}(R), \mathrm{ms}(R), \mathrm{occ}(R))$ where $\mathrm{ms}(R)$ is the *multiset* $\{s \mid s \ \mathsf{in}\ r \in \mathsf{R}\}$ and $\mathrm{occ}(R)$ is the number of occurrences of $\sqcup$ in R. Let $\succ_{\mathsf{Str}}$ be the ordering over string terms such that $s \succ_{\mathsf{Str}} t$ iff s has a greater term size than t, with the convention that ϵ has size 0. Let $\succ_{\mathrm{lex}}$ be the lexicographic extension of the following orderings to tuples like $(\mathcal{V}(R), \mathrm{ms}(R), \mathrm{occ}(R))$ above: the set inclusion ordering; the multiset ordering extending $\succ_{\mathsf{Str}}$; the $>$ ordering over natural numbers. Finally, define $\succ$ where *(i)* $\langle A_1, R_1, V_1 \rangle \succ \langle A_2, R_2, V_2 \rangle$ iff $(\mathcal{V}(R_1), \mathrm{ms}(R_1), \mathrm{occ}(R_1)) \succ_{\mathrm{lex}}$ $(\mathcal{V}(R_2), \mathrm{ms}(R_2), \mathrm{occ}(R_2))$ and *(ii)* $\langle A, R, V \rangle \succ \mathsf{unsat}$. The well foundedness of $\succ$ follows by standard results (see e.g., [3]). □

4.2 Correctness

To prove the correctness of the calculus we use the following properties of the various auxiliary functions.

Lemma 1 (Correctness of Normalization). *Every rule in Figure 3 preserves term equivalence in* T_{LR}.

Lemma 2 (Correctness of π). *For any regular expressions* r_1 *and* r_2, $\pi(r_1, r_2)$ *contains no occurrences of* $\sqcap$. *Moreover,* $\mathcal{L}(\pi(r_1, r_2)) = \mathcal{L}(r_1 \sqcap r_2)$.

Lemma 3. *For all normalized regular expressions* r *and for all characters* $c \in \mathcal{A}$, *the following hold:*

1. $\varepsilon(r)$ *iff* $\epsilon \in \mathcal{L}(r)$;
2. $\mathcal{L}(\partial_c r) = \{w \mid cw \in \mathcal{L}(r)\}$;
3. *for all* $(r_1, r_2) \in \beta(r)$, $\mathcal{L}(r_1 \cdot r_2) = \mathcal{L}(r)$;
4. *for all* $w_1 w_2 \in \mathcal{L}(r)$, *there is a* $(r_1, r_2) \in \beta(r)$ *s.t.* $w_1 \in \mathcal{L}(r_1)$ *and* $w_2 \in \mathcal{L}(r_2)$.

Lemma 4. *Let* x *be a string variable, let* r *be a normalized regular expression with* $\text{top}(r) \notin \{\varnothing, \sqcup\}$, *let* A *be a set of arithmetic constraints, and let* $(r_\gamma, u_\gamma, A_\gamma) = \gamma(r)$.

1. *The constraint set* $S := \{x \mathrel{\mathsf{in}} r\} \cup A$ *is satisfied by a model* $\mathcal{I}$ *of* T_{LR} *iff the set* $S_\gamma := \{x \mathrel{\mathsf{in}} r_\gamma, |x| \approx u_\gamma\} \cup A \cup A_\gamma$ *is satisfied by a model* $\mathcal{I}_\gamma$ *of* T_{LR} *where* $\mathcal{I}$ *and* $\mathcal{I}_\gamma$ *agree on the variables of* S.
2. *All models* $\mathcal{I}$ *of* T_{LR} *satisfying* A_γ *are such that for all* $w \in r_\gamma^{\mathcal{I}}$, *the length of* w *equals* $u_\gamma^{\mathcal{I}}$.

We say that a configuration $\langle A, R, V\rangle$ is *satisfied* by an interpretation $\mathcal{I}$ if the set $A \cup R \cup V$ is satisfied by $\mathcal{I}$. We consider unsat to be satisfied by no interpretation.

Lemma 5. *For every rule of the calculus, the premise configuration is satisfied by a model* $\mathcal{I}_p$ *of* T_{LR} *iff one of its conclusion configurations is satisfied by a model* $\mathcal{I}_c$ *of* T_{LR} *where* $\mathcal{I}_p$ *and* $\mathcal{I}_c$ *agree on the variables shared by the two configurations.*

Using the previous lemma in the left-to-right direction together with a structural induction argument on derivation trees, one can readily show that the root of every closed derivation tree is unsatisfiable. From this, the *refutation soundness* of the calculus easily follows.

Proposition 3 (Refutation Soundness). *Every set of* T_{LR}*-constraints that has a refutation is* T_{LR}*-unsatisfiable.*

Thanks to earlier lemmas and the one below one can also prove that the calculus is *solution sound*.

Lemma 6. *If $\langle A, R, V\rangle$ is a saturated leaf of a derivation tree with root $\langle A_0, R_0, \emptyset\rangle$ then for every (string) variable x in R_0 there is a constraint of the form $(x \,\mathsf{in}\, q)$ in V.*

Proposition 4 (Solution Soundness). *For every saturated leaf $\langle A, R, V\rangle$ of a derivation tree with root $\langle A_0, R_0, \emptyset\rangle$ there is a model $\mathcal{I}$ of T_{LR} that satisfies $A_0 \cup R_0$ and is such that $x^{\mathcal{I}} \in q^{\mathcal{I}}$ for all $(x \,\mathsf{in}\, q) \in V$.*

Proof. Let $K := \langle A, R, V\rangle$ be as above. It is not difficult to show based the derivation rules that $\mathcal{V}(A_0 \cup R_0) \subseteq \mathcal{V}(A \cup R \cup V)$ and $A_0 \subseteq A$. Moreover, every integer variable of V is in A, by definition of γ, and each string variable of R occurs in V exactly once.

The set R contains at most constraints of the form $(\epsilon \,\mathsf{in}\, r)$ with $\epsilon \in \mathcal{L}(r)$; otherwise, one of the derivation rules would apply to K, against the assumption that it is saturated. This makes R trivially satisfiable. The set A is satisfiable as well, otherwise $\mathsf{A\text{-}Conflict}$ would apply. Let $\mathcal{J}$ be a model of T_{LR} satisfying A and let $(x \,\mathsf{in}\, q)$ be any element of V. We claim that the set $q^{\mathcal{J}}$ is nonempty and contains only words of length $|x|^{\mathcal{J}}$. In fact, if $(x \,\mathsf{in}\, q)$ was added to V by $\mathsf{Assign\text{-}1}$, then q is a literal l and $|x| \approx |l|\!\downarrow\, \in A$. If $(x \,\mathsf{in}\, q)$ was added to V by $\mathsf{Assign\text{-}2}$, then $\gamma(r) = (q, u_\gamma, A_\gamma)$ for some r, where $A_\gamma \subseteq A$ and $|x| \approx u_\gamma \in A$. Since $\mathcal{J}$ satisfies A_γ, by Lemma 4(2), all words in $q^{\mathcal{J}}$, if any, are of length $u_\gamma^{\mathcal{J}}$ which is the same as $|x|^{\mathcal{J}}$. To argue that $q^{\mathcal{J}}$ is non-empty, by Lemma 4(2), it is enough to argue that $\mathcal{L}(r)$ is nonempty. This can be seen by observing that, by definition of the the rewrite rules in Figure 3, and by Lemma 1 and Lemma 2, r is guaranteed to contain no occurrences of $\varnothing$ or $\sqcap$, and containing such symbols is a necessary condition for a regular expression to have an empty language. The statement of the lemma follows by the generality of $(x \,\mathsf{in}\, q)$. $\square$

Proposition 5 (Refutation Completeness). *Every set of T_{LR}-constraints unsatisfiable in T_{LR} has a refutation.*

Proof. Contrapositively, suppose that the set of T_{LR}-constraints does not have a refutation. Then, by Proposition 2, it must have a derivation that generates a tree with a saturated branch. By Proposition 4 the set is satisfiable in T_{LR}. $\square$

4.3 Decidability

Proposition 6 (Decidability). *The T_{LR}-satisfiability of quantifier-free Σ_{LR}-formulas with no regular expression variables is decidable.*

Proof. By standard methods, the T_{LR}-satisfiability of quantifier-free Σ_{LR}-formulas with no variables of sort Lan can be effectively reduced to the T_{LR}-satisfiability of T_{LR}-constraints. The existence of a terminating procedure to check such constraints is a consequence of Proposition 1 and Proposition 2. The correctness of the procedure is a consequence of Propositions 3 and 5. $\square$

5 Conclusion and Further Work

We have presented an algebraic approach for solving regular membership constraints and linear length constraints in the theory of strings. This approach works directly on regular expressions without the need to translate them to automata. Moreover, it does not require imposing any *a priori* length bounds on string variables. We have proved that our approach is sound, complete and terminating, thus it is a decision procedure for this fragment. In addition, when the constraints are satisfiable, our approach provides a model—in fact a generator of a set of models. Therefore, it has all the properties required for integration into an SMT solver.

In ongoing work, we are investigating a possible extension of our procedure to word equations over unbounded strings. Although the satisfiability of sets of word equations is also decidable, the decidability of the combined language is still an open problem. We hope to find a fragment that is sufficiently expressive for real-world problems, while also being decidable, or at least effective for solving problems in practice.

Additionally, we have identified two bottlenecks in the calculus presented here: the computation of the intersection and the complement operations over regular expressions. Therefore, we plan to focus on developing approaches for computing these operations that are efficient in practice. We are also working on an extension to symbolic regular expressions, specifically, regular expressions that contain string variables.

References

1. Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Lukas Holik, Ahmed Rezine, Philipp Rummer, and Jari Stenman. String constraints for verification. In Armin Biere and Roderick Bloem, editors, *Proceedings of the 26th International Conference on Computer Aided Verification*, volume 8559 of *Lecture Notes in Computer Science*. Springer, 2014.
2. Valentin Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *Theor. Comput. Sci.*, 155(2):291–319, March 1996.
3. Franz Baader and Tobias Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.
4. Bahareh Badban and Mohammad Torabi Dashti. Semi-linear parikh images of regular expressions via reduction. In *Proceedings of the 35th International Conference on Mathematical Foundations of Computer Science*, MFCS'10, pages 653–664, Berlin, Heidelberg, 2010. Springer-Verlag.
5. Clark Barrett, Roberto Sebastiani, Sanjit Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185, chapter 26, pages 825–885. IOS Press, February 2009.
6. Gerard Berry and Ravi Sethi. From regular expressions to deterministic automata. *Theor. Comput. Sci.*, 48(1):117–126, December 1986.
7. Nikolaj Bjørner, Nikolai Tillmann, and Andrei Voronkov. Path feasibility analysis for string-manipulating programs. In *Proceedings of the 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems: Held*

as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009,, pages 307–321. Springer-Verlag, 2009.

8. Aske Simon Christensen, Anders Møller, and Michael I. Schwartzbach. Precise analysis of string expressions. In *Proceedings of the 10th International Conference on Static Analysis*, SAS'03, pages 1–18, Berlin, Heidelberg, 2003. Springer-Verlag.
9. Xiang Fu and Chung chih Li. A string constraint solver for detecting web application vulnerability. In *Proceedings of the 22nd International Conference on Software Engineering and Knowledge Engineering*, SEKE'2010. Knowledge Systems Institute Graduate School, 2010.
10. Indradeep Ghosh, Nastaran Shafiei, Guodong Li, and Wei-Fan Chiang. JST: An automatic test generation tool for industrial Java applications with strings. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 992–1001, Piscataway, NJ, USA, 2013. IEEE Press.
11. Jesper G. Henriksen, Jakob L. Jensen, Michael E. Jørgensen, Nils Klarlund, Robert Paige, Theis Rauhe, and Anders Sandholm. Mona: Monadic second-order logic in practice. In *Proceedings of the First International Workshop on Tools and Algorithms for Construction and Analysis of Systems*, TACAS '95, pages 89–110, London, UK, UK, 1995. Springer-Verlag.
12. Pieter Hooimeijer and Margus Veanes. An evaluation of automata algorithms for string analysis. In *Proceedings of the 12th international conference on Verification, model checking, and abstract interpretation*, pages 248–262. Springer-Verlag, 2011.
13. Pieter Hooimeijer and Westley Weimer. A decision procedure for subset constraints over regular languages. In *Proceedings of the 2009 ACM SIGPLAN conference on Programming language design and implementation*, pages 188–198. ACM, 2009.
14. Pieter Hooimeijer and Westley Weimer. Solving string constraints lazily. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*, pages 377–386. ACM, 2010.
15. Adam Kiezun, Vijay Ganesh, Philip J. Guo, Pieter Hooimeijer, and Michael D. Ernst. HAMPI: a solver for string constraints. In *Proceedings of the eighteenth international symposium on Software testing and analysis*, pages 105–116. ACM, 2009.
16. Nils Klarlund, Anders Møller, and Michael I. Schwartzbach. Mona implementation secrets. In *Revised Papers from the 5th International Conference on Implementation and Application of Automata*, CIAA '00, pages 182–194, London, UK, UK, 2001. Springer-Verlag.
17. Dexter Kozen. Lower bounds for natural proof systems. In *FOCS*, pages 254–266. IEEE Computer Society, 1977.
18. Guodong Li and Indradeep Ghosh. Pass: String solving with parameterized array and interval automaton. In Valeria Bertacco and Axel Legay, editors, *Hardware and Software: Verification and Testing*, volume 8244 of *Lecture Notes in Computer Science*, pages 15–31. Springer International Publishing, 2013.
19. Tianyi Liang, Andrew Reynolds, Cesare Tinelli, Clark Barrett, and Morgan Deters. A DPLL(T) theory solver for a theory of strings and regular expressions. In Armin Biere and Roderick Bloem, editors, *Proceedings of the 26th International Conference on Computer Aided Verification*, volume 8559 of *Lecture Notes in Computer Science*. Springer, 2014.
20. Kenny Zhuo Ming Lu. *XHaskell - Adding Regular Expression Type to Haskell*. PhD thesis, National University of Singapore, 2009.
21. G. S. Makanin. The problem of solvability of equations in a free semigroup. *English transl. in Math USSR Sbornik*, 32:147–236, 1977.

22. Yuri Matiyasevich. Hilbert's tenth problem and paradigms of computation. In *Proceedings of the First International Conference on Computability in Europe: New Computational Paradigms*, CiE'05, pages 310–321. Springer-Verlag, Berlin, Heidelberg, 2005.
23. Rohit J. Parikh. On context-free languages. *J. ACM*, 13(4):570–581, October 1966.
24. Wojciech Plandowski. Satisfiability of word equations with constants is in pspace. *J. ACM*, 51(3):483–496, May 2004.
25. Grigore Rosu and Mahesh Viswanathan. Testing extended regular language membership incrementally by rewriting. In Robert Nieuwenhuis, editor, *Rewriting Techniques and Applications*, volume 2706 of *Lecture Notes in Computer Science*, pages 499–514. Springer Berlin Heidelberg, 2003.
26. K.U. Schulz, editor. *Word Equations and Related Topics.* Springer-Verlag New York, Inc., New York, NY, USA, 1990.
27. Takaaki Tateishi, Marco Pistoia, and Omer Tripp. Path- and index-sensitive string analysis based on monadic second-order logic. *ACM Trans. Softw. Eng. Methodol.*, 22(4):33:1–33:33, October 2013.
28. Nikolai Tillmann and Jonathan Halleux. Pex - white box test generation for .net. In Bernhard Beckert and Reiner Hähnle, editors, *Tests and Proofs*, volume 4966 of *Lecture Notes in Computer Science*, pages 134–153. Springer Berlin Heidelberg, 2008.
29. Minh-Thai Trinh, Duc-Hiep Chu, and Joxan Jaffar. S3: A symbolic string solver for vulnerability detection in web applications. In Moti Yung and Ninghui Li, editors, *Proceedings of the 21st ACM Conference on Computer and Communications Security*, 2014.
30. Margus Veanes. Applications of symbolic finite automata. In *Proceedings of the 18th International Conference on Implementation and Application of Automata*, CIAA'13, pages 16–23, Berlin, Heidelberg, 2013. Springer-Verlag.
31. Margus Veanes, Nikolaj Bjørner, and Leonardo De Moura. Symbolic automata constraint solving. In *Proceedings of the 17th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, LPAR'10, pages 640–654, Berlin, Heidelberg, 2010. Springer-Verlag.
32. Fang Yu, Muath Alkhalaf, and Tevfik Bultan. Stranger: An automata-based string analysis tool for php. In Javier Esparza and Rupak Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 6015 of *Lecture Notes in Computer Science*, pages 154–157. Springer Berlin Heidelberg, 2010.
33. Yunhui Zheng, Xiangyu Zhang, and Vijay Ganesh. Z3-str: A z3-based string solver for web application analysis. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, pages 114–124, New York, NY, USA, 2013. ACM.

A Proofs

A.1 Proofs for Lemma 3

Lemma 7 (Correctness of ε). *Let r be a $\sqcap$-free regular expression and $c \in \mathcal{A}$. $\varepsilon(r)$ iff $\epsilon \in \mathcal{L}(r)$.*

Proof. Termination of the predicate ε follows from its inductive definition and the fact that after each computation step ε either terminates, or calls on finite number of regular expressions, each with a strictly fewer symbols.

By definition, if a regular expression is with top symbol either ϵ or $*$, its language contains the empty string. If $r = r_1 \cdot r_2$, we need check both r_1 and r_2. If $r = r_1 \sqcup r_2$, r contains the empty string iff either r_1 or r_2 contains.

Lemma 8 (Correctness of ∂_c). *Let r be a $\sqcap$-free regular expression and $c \in \mathcal{A}$. $\mathcal{L}(\partial_c r) = \{s \mid cs \in \mathcal{L}(r)\}$.*

Proof. Termination of the function ∂_c follows from the termination of the predicate ε and the fact that a computation step either induces calls to the function ∂_c on (at most two) regular expressions (each with strictly fewer symbols), or terminates on atomic expressions ($\varnothing$, l, and Ch).

We prove the correctness by induction. The base cases hold on atomic expressions by definition.

If $r = r_1 \cdot r_2$ and $\neg\varepsilon(r_1)$,

$$\begin{aligned}\mathcal{L}(\partial_c(r_1 \cdot r_2)) &= \mathcal{L}(\partial_c r_1 \cdot r_2)\\ &= \{s \mid cs \in \mathcal{L}(r_1)\} \cdot \mathcal{L}(r_2)\\ &= \{st \mid cs \in \mathcal{L}(r_1), t \in \mathcal{L}(r_2)\}\\ &= \{s \mid cs \in \mathcal{L}(r_1 \cdot r_2)\}.\end{aligned}$$

If $r = r_1 \cdot r_2$ and $\varepsilon(r_1)$,

$$\begin{aligned}\mathcal{L}(\partial_c(r_1 \cdot r_2)) &= \mathcal{L}(\partial_c r_1 \cdot r_2 \sqcup \partial_c r_2)\\ &= \mathcal{L}(\partial_c r_1 \cdot r_2) \cup \mathcal{L}(\partial_c r_2)\\ &= (\{s \mid cs \in \mathcal{L}(r_1)\} \cdot \mathcal{L}(r_2)) \cup \mathcal{L}(\partial_c \epsilon \cdot r_2)\\ &= \{st \mid cs \in \mathcal{L}(r_1), t \in \mathcal{L}(r_2)\} \cup \{s \mid cs \in \mathcal{L}(\epsilon \cdot r_2)\}\\ &= \{s \mid cs \in \mathcal{L}(r_1 \cdot r_2)\}.\end{aligned}$$

If $r = r_1 \sqcup r_2$,

$$\begin{aligned}\mathcal{L}(\partial_c(r_1 \sqcup r_2)) &= \mathcal{L}(\partial_c r_1 \sqcup \partial_c r_2)\\ &= \{s \mid cs \in \mathcal{L}(r_1)\} \cup \{s \mid cs \in \mathcal{L}(r_2)\}\\ &= \{s \mid cs \in \mathcal{L}(r_1) \cup \mathcal{L}(r_2)\}\\ &= \{s \mid cs \in \mathcal{L}(r_1 \sqcup r_2)\}.\end{aligned}$$

If the regular expression is r^*, $\mathcal{L}(\partial_c r^*) = \mathcal{L}(\partial_c r \cdot r^*)$, we have two sub-cases:

(i) when $\neg\varepsilon(r)$, $\mathcal{L}(\partial_c r, r^*) = \emptyset \,\cup\, \{s \mid cs \in \mathcal{L}(r \cdot r^*)\}$;
(ii) when $\varepsilon(r)$,

$$\begin{aligned}\mathcal{L}(\partial_c r \cdot r^*) &= (\mathcal{L}(\partial_c r) \,\cup\, \emptyset) \,\cup\, (\mathcal{L}(\partial_c r \cdot r) \,\cup\, \partial_c r) \cup \\ &\quad (\mathcal{L}(\partial_c r \cdot r^2) \,\cup\, \mathcal{L}(\partial_c r \cdot r)) \,\cup \ldots \\ &= \emptyset \,\cup\, \{s \mid cs \in \mathcal{L}(r \cdot r^*)\}.\end{aligned}$$

Thus, we complete with:

$$\begin{aligned}&\emptyset \,\cup\, \{s \mid c \cdot s \in \mathcal{L}(r \cdot r^*)\} \\ &\quad = \{s \mid cs = \epsilon\} \,\cup\, \{s \mid cs \in \mathcal{L}(r \cdot r^*)\} \\ &\quad = \{s \mid cs \in \mathcal{L}(\epsilon \sqcup r \sqcup r^2, \cdots))\} \\ &\quad = \{s \mid cs \in \mathcal{L}(r^*)\}.\end{aligned}$$

Lemma 9 (Correctness for β-1). *Let r be a $\sqcap$-free regular expression. For all $(r_1, r_2) \in \beta(r)$, $\mathcal{L}(r_1 \cdot r_2) = \mathcal{L}(r)$;*

Proof. Prove by induction on the structure of a regular expressions. Base cases are for atomic regular expressions ($\varnothing$, l, and Ch). The property holds by definition.

If $r = r_1 \sqcup r_2$, $\beta(r) = \beta(r_1) \cup \beta(r_2)$. The property holds by hypothesis.

If $r = r_1 \cdot r_2$, by hypothesis, $\beta(b_1)$ holds the property. Thus, $\mathcal{L}(r_{11} \cdot r_{12} \cdot r_2) = \mathcal{L}(r)$. Similarly, we have the other branch.

If the regular expression is r^*, $r^* = \epsilon \sqcup (r^* \cdot r \cdot r^*)$. Since r is smaller than r^*, by hypothesis, $\forall (r_1, r_2) \in \beta(r).\mathcal{L}(r_1 \cdot r_2) = \mathcal{L}(r)$. Thus, $\mathcal{L}(r^* \cdot r_1 \cdot r_2 \cdot r^*) = \mathcal{L}(r^* \cdot r \cdot r^*)$, for all pairs in $\beta(r)$.

Lemma 10 (Correctness for β-2). *Let r be a $\sqcap$-free regular expression. For all $w_1 w_2 \in \mathcal{L}(r)$, there is a $(r_1, r_2) \in \beta(r)$ s.t. $w_1 \in \mathcal{L}(r_1)$ and $w_2 \in \mathcal{L}(r_2)$.*

Proof. The proof is by induction on the structure of a regular expression. Because the algorithm works from top down, every recursive call is made on a subexpression, and the number of symbols in a regular expression is finite, it always terminates.

The base cases are for atomic regular expressions, the correctness follows from the definition of regular expressions.

Let w_1, w_2 be two arbitrary words and $w_1 w_2 \in \mathcal{L}(r_1 \cdot r_2)$. By definition, we have (i) $w_1 \in \mathcal{L}(r_1)$ and $w_2 \in \mathcal{L}(r_2)$; (ii) $w_{11} \in \mathcal{L}(r_1)$, $w_{12} w_2 \in \mathcal{L}(r_2)$, and $w_1 = w_{11} w_{12}$; (iii) $w_1 w_{21} \in \mathcal{L}(r_1)$, $w_{22} \in \mathcal{L}(r_2)$, and $w_2 = w_{21} w_{22}$. In case (i), $(r_1, r_2) \in \beta(r_1 \cdot r_2)$. In case (ii), by hypothesis, $\exists (r_3, r_4) \in \beta(r_2)$, such that, $w_{12} \in \mathcal{L}(r_3)$ and $w_2 \in \mathcal{L}(r_4)$. Thus, $w_1 \in \mathcal{L}(r_1 \cdot r_3)$. By the definition of β, $(r_1 \cdot r_3, r_4)$ is in $\beta(r_1 \cdot r_2)$. Similarly, we can prove the property for case (iii).

If $r = r_1 \sqcup r_2$, it holds because of the semantics of union. It holds for r^*, since $r^* = \epsilon \sqcup (r \cdot r^*)$ and by a similar reason for union and concatenation.

A.2 Proofs for Lemma 4

Lemma 11 (Model Preservation of γ'). *Let A be an arbitrary set of linear arithmetic constraints, let x be a variable of sort String, let r be a normalized regular expression with $\mathrm{top}(r) \notin \{\varnothing, \sqcup\}$, and let $\gamma'(r, A) = (q, u, A')$. For all models $\mathcal{I}$ of both A and $x \,\mathsf{in}\, r$, there is a model $\mathcal{I}'$ that satisfies A' and $x \,\mathsf{in}\, q$ and $|x| \approx u$, $x^{\mathcal{I}} = x^{\mathcal{I}'}$, and forall $v \in \mathcal{V}(A)$, $v^{\mathcal{I}} = v^{\mathcal{I}'}$.*

Proof. Prove by induction on the structure of a regular expression. Base cases (when r is either l or Ch) hold by the definition of a regular expression. It is clear that the property holds for r^*, l^z and Ch^z. Let $\mathcal{I}$ be an arbitrary model that satisfies A and $x \,\mathsf{in}\, r$, and $x^{\mathcal{I}} = w$.

Assume $r = r_1 \cdot r_2$. By the definition of a regular expression, there exist w_1 and w_2, such that $w = w_1 w_2$, $w_1 \in \mathcal{L}(r_1)$, and $w_2 \in \mathcal{L}(r_2)$. We expand $\mathcal{I}$ to $\mathcal{I}'$ by adding $u_1 \mapsto |w_1|$ and $u_2 \mapsto |w_2|$. Thus, $\mathcal{I}'$ satisfies $|x| \approx u_1 + u_2$. Since γ introduces constraints only over fresh variables to A, $\mathcal{V}(A) = \mathcal{V}(A_1) \cap \mathcal{V}(A_2)$, and by hypothesis, $\mathcal{I}'$ holds the property for both r_1 and r_2.

Assume $r = (r_1 \sqcup r_2)^z$. By the definition of sh, there exist w_1 and w_2, such that $w \in \mathsf{sh}(w_1, w_2, r)$, $w_1 \in \mathcal{L}(r_1^{z^{\mathcal{I}}})$, and $w_2 \in \mathcal{L}(r_2^{z^{\mathcal{I}}})$. We can construct $\mathcal{I}'$ by adding $u_1 \mapsto |w_1|$ and $u_2 \mapsto |w_2|$. By hypothesis, $\mathcal{I}'$ holds the property for the sub-expressions. Similarly, the property holds for $r = (r_1 \cdot r_2)^z$.

Assume $r = (r_1^*)^z$. If $z^{\mathcal{I}} = 0$, $x^{\mathcal{I}} = \epsilon$ and $z_1^{\mathcal{I}'} = 0$. If $z^{\mathcal{I}} > 0$, assume $w \in \mathcal{L}((r_1)^n)$ (where n is a natural number), then n is a model of z_1 in $\mathcal{I}'$.

Lemma 12 (Model Generation of γ'). *Let A be an arbitrary set of linear arithmetic constraints, let x be a variable of sort String, let r be an arbitrary normalized regular expression with $\mathrm{top}(r) \notin \{\varnothing, \sqcup\}$, and let $\gamma'(r, A) = (q, u, A')$. For all models $\mathcal{I}$ of the constraints (A', $x \,\mathsf{in}\, q$ and $|x| \approx u$), indeed it also satisfies A and $x \,\mathsf{in}\, r$.*

Proof. Since $A \subseteq A'$, any model of A' is a model of A. Prove by induction on the structure of a regular expression. Base cases (when r is either l or Ch) hold by the definition of a regular expression. It is clear that the property holds for r^*, l^z and Ch^z. Let $\mathcal{I}$ be an arbitrary model that satisfies A' and $x \,\mathsf{in}\, q$. $x^{\mathcal{I}} = w$.

Assume $r = r_1 \cdot r_2$. Since γ only introduces constraints over fresh variables to A, $\mathcal{V}(A) = \mathcal{V}(A_1) \cap \mathcal{V}(A_2)$. Since $w \in (q_1 \cdot q_2)^{\mathcal{I}}$, there exist w_1 and w_2 such that $w_1 \in q_1^{\mathcal{I}}$ and $w_2 \in q_2^{\mathcal{I}}$. By hypothesis, $w_1 \in \mathcal{L}(r_1)$ and $w_2 \in \mathcal{L}(r_2)$. Thus, $w \in \mathcal{L}(r_1 \cdot r_2)$.

If $r = (r_1 \sqcup r_2)^z$ or $r = (r_1 \cdot r_2)^z$, by the definition of sh, $w \in \mathcal{L}(r)$.

Assume $r = (r_1^*)^z$. If $z^{\mathcal{I}} = 0$, $w = \epsilon$ and $\epsilon \in r$. If $z^{\mathcal{I}} > 0$, $w \in \mathcal{L}(r_1^{z_1^{\mathcal{I}}})$, and thus $w \in \mathcal{L}(r_1^{z_1^{\mathcal{I}}} \cdot (r_1^*)^{z^{\mathcal{I}}-1})$ which is $(r_1^*)^{z^{\mathcal{I}}}$.

Lemma 13. *Let $K = \langle A, R, V \rangle$ be a configuration in a derivation tree rooted by $\langle A_0, R_0, \emptyset \rangle$. If $x \,\mathsf{in}\, q \in V$, then for every model $\mathcal{I}$ of A, $q^{\mathcal{I}}$ is not empty, and $\forall w \in q^{\mathcal{I}}.\ |w| = |x|^{\mathcal{I}}$.*

Proof. If $x \,\mathsf{in}\, q \in V$, the constraint is added either by $\mathsf{Assign\text{-}1}$ or $\mathsf{Assign\text{-}2}$. It is obvious to show that the property holds for constraints added by $\mathsf{Assign\text{-}1}$. Thus, we focus on the constraints added by $\mathsf{Assign\text{-}2}$.

First, for every model $\mathcal{I}$ of A, $q^{\mathcal{I}}$ is not empty, because: (i) q only contains non-negative coefficients, since when z_i is introduced, $z_i \geq 0$ is added. $\mathcal{I}$ has a value for z_i, since they are consistent with A. (ii) q does not contain $\sqcap$ nor $\varnothing$. (iii) $\mathsf{sh}(q_1^{\mathcal{I}}, q_2^{\mathcal{I}}, r^{\mathcal{I}})$ is not empty — we only have two cases to consider: Case 1: $r = (r_1 \sqcup r_2)^z$: In this case: $q_1^{\mathcal{I}} = r_1^{z_1^{\mathcal{I}}}, q_2^{\mathcal{I}} = r_2^{z_2^{\mathcal{I}}}, z^{\mathcal{I}} = z_1^{\mathcal{I}} + z_2^{\mathcal{I}}$. By induction $q_1^{\mathcal{I}}$ and $q_2^{\mathcal{I}}$ are not empty, so $q_1^{\mathcal{I}} \cdot q_2^{\mathcal{I}}$ is not empty. Note that $(q_1 \cdot q_2)^{\mathcal{I}}$ is a subset of $\mathsf{sh}(q_1^{\mathcal{I}}, q_2^{\mathcal{I}}, r^{\mathcal{I}})$. Thus, it is not empty. Case 2: $r = (r_1 \cdot r_2)^z$: In this case: $q_1^{\mathcal{I}} = r_1^{z^{\mathcal{I}}}, q_2^{\mathcal{I}} = r_2^{z^{\mathcal{I}}}$. By induction $q_1^{\mathcal{I}}$ and $q_2^{\mathcal{I}}$ are not empty. Let $n = z^{\mathcal{I}}$, $n \geq 0$, and let $w_1 \in q_1^{\mathcal{I}}$, $w_2 \in q_2^{\mathcal{I}}$. Because $w_1 \in r_1^n$, we can divide w_1 into n pieces and each piece is in r_1, i.e., $w_1 = a_1 a_2 \ldots a_n$, $\forall i.\ a_i \in r_1$. Similarly, we can break w_2 into n pieces, i.e., $w_2 = b_1 b_2 \ldots b_n$, $\forall i.\ b_i \in r_2$. Thus, $w_3 = a_1 b_1 a_2 b_2 \ldots a_n b_n \in (r_1 \cdot r_2)^n$. Therefore, it is not empty.

Now, we show by induction that $\forall w \in q^{\mathcal{I}}.\ |w| = |x|^{\mathcal{I}}$. Base cases ($l$ and Ch) trivially hold. It is easy to show it holds for $r_1 \cdot r_2$, r^*, l^z and Ch^z. When $r = (r_1 \sqcup r_2)^z$, by inductive hypothesis, we have $\forall w_1 \in q_1^{\mathcal{I}}.\ |w_1| = u_1^{\mathcal{I}}$ and $\forall w_2 \in q_2^{\mathcal{I}}.\ |w_2| = u_2^{\mathcal{I}}$. Since $\mathcal{L}(\mathsf{sh}(q_1, q_2, r)^{\mathcal{I}}) = \{a_1 b_1 \ldots a_{z^{\mathcal{I}}} b_{z^{\mathcal{I}}} \in \mathcal{L}(r^{\mathcal{I}}) \mid a_1 \ldots a_{z^{\mathcal{I}}} \in \mathcal{L}(q_1^{\mathcal{I}}), b_1 \ldots b_{z^{\mathcal{I}}} \in \mathcal{L}(q_2^{\mathcal{I}})\}$, Thus, we have $\forall w \in r^{\mathcal{I}}.\ |w| = (u_1 + u_2)^{\mathcal{I}}$. The case $r = (r_1 \cdot r_2)^z$ can be proved similarly. When $r = (r_1^*)^z$, if $z^{\mathcal{I}} = 0$, then $z'^{\mathcal{I}} = u^{\mathcal{I}} = 0$ and the property holds; if $z^{\mathcal{I}} > 0$, then the property holds by induction.

Lemma 14 (Correctness of γ). *Let x be a string variable, let r be a normalized regular expression with* $\mathrm{top}(r) \notin \{\varnothing, \sqcup\}$*, let A be a set of arithmetic constraints, and let $(r_\gamma, u_\gamma, A_\gamma) = \gamma(r)$. Then (i) the constraint set $S := \{x \,\mathsf{in}\, r\} \cup A$ is satisfied by a model $\mathcal{I}$ of T_{LR} iff the set $S_\gamma := \{x \,\mathsf{in}\, r_\gamma, |x| \approx u_\gamma\} \cup A \cup A_\gamma$ is satisfied by a model $\mathcal{I}_\gamma$ of T_{LR} where $\mathcal{I}$ and $\mathcal{I}_\gamma$ agree on the variables of S, and (ii) all interpretations $\mathcal{J}$ satisfying A_γ are such that for all $w \in r_\gamma^{\mathcal{J}}$, $|x| = u_\gamma^{\mathcal{J}}$.*

Proof. Point (i) is proved by Lemma 11 and 12, and Point (ii) is proved by Lemma 13.

A.3 Proofs for Lemma 5

Lemma 15. *For every rule of the calculus, the premise configuration is satisfied by a model $\mathcal{I}_p$ of T_{LR} iff one of its conclusion configurations is satisfied by a model $\mathcal{I}_c$ of T_{LR} where $\mathcal{I}_p$ and $\mathcal{I}_c$ agree on the variables shared by the two configurations.*

Proof. For each rule, let $K_p := \langle A_p, R_p, V_p \rangle$ be a premise configuration, and let $K_c := \langle A_c, R_c, V_c \rangle$ be a conclusion configuration. If K_p is satisfiable, we use $\mathcal{I}_p$ to denote an interpretation of T_{LR} satisfying K_p; if K_c is satisfiable, we use $\mathcal{I}_c$ to denote an interpretation of T_{LR} satisfying K_c. In the proof we rely on Lemma 1, i.e., normalization maintains term equivalence.

– Assign-1

$$\frac{\mathsf{R} = R,\ x \,\mathsf{in}\, l}{\mathsf{A} := \mathsf{A},\ |x| \approx |l|{\downarrow} \quad \mathsf{R} := (R\{x \mapsto l\}){\downarrow} \quad \mathsf{V} := \mathsf{V}, x \,\mathsf{in}\, l}$$

Only If: The constraint $x \,\mathsf{in}\, l$ is satisfied in $\mathcal{I}_p$ iff $x^{\mathcal{I}_p} = l$. We will show that $\mathcal{I}_p$ is an interpretation that also satisfies the conclusion configuration K_c. Since $x^{\mathcal{I}_p} = l$, $x \approx l$ in V_c and $|x| \approx |l|{\downarrow}$ are satisfiable by $\mathcal{I}_p$. Since $x^{\mathcal{I}_p} = l$, it follows that $(s\{x \mapsto l\})^{\mathcal{I}_p} = s^{\mathcal{I}_p}$, $\forall s \,\mathsf{in}\, r \in R$. Thus, $(R\{x \mapsto l\})^{\mathcal{I}_p} = R^{\mathcal{I}_p}$. Therefore, $\mathcal{I}_p$ is an interpretation that satisfies the conclusion configuration K_c of the rule.
If: Since $\mathcal{I}_c$ is a model of $x \,\mathsf{in}\, l$ in V_c, it is a model of the same constraint in R_p. Since $\forall s \,\mathsf{in}\, r \in R_p$. $(s\{x \mapsto l\})^{\mathcal{I}_c} = s^{\mathcal{I}_c}$, $\mathcal{I}_c$ is a model of R_p.

– Assign-2

$$\frac{\mathsf{R} = R,\ x \,\mathsf{in}\, r \quad \mathrm{top}(r) \notin \{\sqcup, \varnothing\} \quad x \notin \mathcal{V}(R) \quad \gamma(r) = (q, u, A)}{\mathsf{A} := \mathsf{A},\ |x| \approx u{\downarrow},\ A{\downarrow} \quad \mathsf{V} := \mathsf{V},\ x \,\mathsf{in}\, q \quad \mathsf{R} = R}$$

Only If: From the definition of the γ function (Fig. 9) follows that A, q and u may contain only fresh variables. Thus, variables in $\mathcal{V}(A) \cup \mathcal{V}(q) \cup \mathcal{V}(u)$ do not occur among the variables of the premise configuration K_p. Consider a subset of the premise configuration constraints $S := \{x \,\mathsf{in}\, r\} \cup A_p$. Obviously, $\mathcal{I}_p$ satisfies S. Then, by Lemma 4 there exists an interpretation $\mathcal{I}'$ of T_{LR} which satisfies $S' := \{x \,\mathsf{in}\, q, |x| \approx u\} \cup A_p \cup A$ and agrees with $\mathcal{I}_p$ on the variables shared by S and S'. Since A, q and u may contain only fresh variables and $A_p \subset S'$, the only shared variables S' can have with $A_p \cup R_p \cup V_p$ are the variables shared with A_p. At the same time $A_p \subset S$, thus every variable of A_p is a variable shared by S and S' and the interpretations $\mathcal{I}'$ and $\mathcal{I}_p$ agree on it. Also, x is a variable shared by S and S', thus $x^{\mathcal{I}'} = x^{\mathcal{I}_p}$.Therefore, we can build an interpretation $\mathcal{I}_c$, such that $z^{\mathcal{I}_c} = z^{\mathcal{I}_p}$ if $z \,\mathsf{in}\, (x \cup \mathcal{V}(A_p) \cup \mathcal{V}(R_p) \cup \mathcal{V}(V_p))$, and $z^{\mathcal{I}_c} = z^{\mathcal{I}'}$ otherwise. Clearly, $\mathcal{I}_c$ is an interpretation that satisfies S', R_p and V_p. Thus, $\mathcal{I}_c$ satisfies the conclusion configuration K_c of the rule and agrees with $\mathcal{I}_p$ on all variables shared by K_p and K_c.
If: By assumption, $x \notin \mathcal{V}(R)$. For a normalized regular expression r with $\mathrm{top}(r) \notin \{\varnothing, \sqcup\}$, such that $\gamma(r) = (q, u, A)$ we will show that $\mathcal{I}_c$ also satisfies $x \,\mathsf{in}\, r$. This will automatically imply that $\mathcal{I}_c$ satisfies the premise configuration K_p of the rule. Consider a set of constraints $S' := \{x \,\mathsf{in}\, q, |x| \approx u\} \cup A_p \cup A$, which is a subset of the conclusion configuration constraints. By Lemma 4 there exists an interpretation $\mathcal{I}$ of T_{LR} which satisfies constraints $S := \{x \,\mathsf{in}\, r\} \cup A_p$ and agrees with $\mathcal{I}_c$ on the variables shared by S and S'. As noticed, by the definition of the γ function the variables in A, q and u are freshly introduced and thus are not occurring in $\mathcal{V}(A_p) \cup \mathcal{V}(R_p) \cup \mathcal{V}(V_p)$. Since $A_p \subset S'$ the only shared variables S' can have with $A_p \cup R_p \cup V_p$ are the variables shared with A_p. At the same time $A_p \subset S$, thus every variable of A_p is a variable shared by S and S' and the interpretations $\mathcal{I}_c$ and $\mathcal{I}$

agree on it. Also, x is a variable shared by $\mathcal{I}_c$ and $\mathcal{I}$, thus $x^{\mathcal{I}_c} = x^{\mathcal{I}}$. Clearly, $\mathcal{V}(S) = \mathcal{V}(A_p) \cup x$ and, therefore, the interpretation $\mathcal{I}_c$ satisfies S, too. Thus $\mathcal{I}_c$ satisfies the premise configuration K_p of the rule.

– Drop

$$\frac{\mathsf{R} = R,\ c \cdot s\ \mathsf{in}\ r}{\mathsf{R} := R,\ s\ \mathsf{in}\ \partial_c(r){\downarrow}}$$

Only If: Let l be a string constant, such that $(c \cdot s)^{\mathcal{I}_p} = l$. Since $(c \cdot s)^{\mathcal{I}_p} = cs^{\mathcal{I}_p}$, by the definition of the derivative function $l = c\partial_c(l)$, and thus $s^{\mathcal{I}_p} = \partial_c(l)$. The constraint $c{\cdot}s\mathsf{in}r$ is satisfied in $\mathcal{I}_p$ iff $l \in \mathcal{L}(r)$, that is $c\partial_c(l) \in \mathcal{L}(r)$ – which by the definition of the derivative means $\partial_c(l) \in \mathcal{L}(\partial_c(r))$. Thus, since $s^{\mathcal{I}_p} = \partial_c(l)$ then $s^{\mathcal{I}_p} \in \mathcal{L}(\partial_c(r))$ and the constraint $s\ \mathsf{in}\ \partial_c(r)$ is satisfiable in $\mathcal{I}_p$. So is satisfiable in $\mathcal{I}_p$ the new constraint $s\ \mathsf{in}\ \partial_c(r){\downarrow}$ in R_c. Thus $\mathcal{I}_p$ satisfies the conclusion configuration K_c of the rule.
If: We will show that $\mathcal{I}_c$ also satisfies $c \cdot s\ \mathsf{in}\ r$, then $\mathcal{I}_c$ will automatically satisfy the premise configuration K_p of the rule. Let l be a string constant, such that $s^{\mathcal{I}_c} = l$. The constraint $s\ \mathsf{in}\ \partial_c(r)$ is satisfied in $\mathcal{I}_c$ iff $l \in \mathcal{L}(\partial_c(r))$. By the definition of the derivative function $l \in \mathcal{L}(\partial_c(r))$ iff $cl \in \mathcal{L}(r)$. Since $s^{\mathcal{I}_c} = l$, then $(c \cdot s)^{\mathcal{I}_c} = cl$ and thus $(c \cdot s)^{\mathcal{I}_c} \in \mathcal{L}(r)$. Which means $\mathcal{I}_c$ satisfies $c \cdot s\ \mathsf{in}\ r$ and consequently satisfies the premise configuration of the rule, too.

– Split

$$\frac{\mathsf{R} := R,\ x \cdot s\ \mathsf{in}\ r}{\|_{(r_1,r_2)\in\beta(r)}\ \mathsf{R} := R,\ x\ \mathsf{in}\ r_1{\downarrow}\,,\ s\ \mathsf{in}\ r_2{\downarrow}}$$

Only If: Let l_1 and l_2 be string constants, such that $x^{\mathcal{I}_p} = l_1$ and $s^{\mathcal{I}_p} = l_2$. Then $(x \cdot s)^{\mathcal{I}_p} = x^{\mathcal{I}_p} s^{\mathcal{I}_p} = l_1 l_2$. The constraint $x \cdot s\ \mathsf{in}\ r$ is satisfied in $\mathcal{I}_p$ iff $l_1 l_2 \in \mathcal{L}(r)$. By Lemma 10 about the β function, for l_1 and l_2 there exists a pair of regular expressions $(r_1, r_2) \in \beta$, such that $l_1 \in \mathcal{L}(r_1)$ and $l_2 \in \mathcal{L}(r_2)$. Since $x^{\mathcal{I}_p} = l_1$ and $s^{\mathcal{I}_p} = l_2$, then $\mathcal{I}_p$ satisfies $x \in r_1$ and $s \in r_2$ and thus satisfies $x \in r_1{\downarrow}$ and $s \in r_2{\downarrow}$. Therefore, $\mathcal{I}_p$ is an interpretation that satisfies on of the conclusion configurations of the rule.
If: Let this branch be $(r_1, r_2) \in \beta(r)$, R, $x\,\mathsf{in}\,r_1$, and $s\,\mathsf{in}\,r_2$. By the definition of the β function, the pair (r_1, r_2) is such such $\mathcal{L}(r_1 \cdot r_2) = \mathcal{L}(r)$. Let l_1 and l_2 be string constants, such that $x^{\mathcal{I}_c} = l_1$ and $s^{\mathcal{I}_c} = l_2$. Then, $x\ \mathsf{in}\ r_1$ and $s\ \mathsf{in}\ r_2$ are satisfied in $\mathcal{I}_c$ iff $l_1 \in \mathcal{L}(r_1)$ and $l_2 \in \mathcal{L}(r_2)$. From here, $l_1 l_2 \in \mathcal{L}(r_1 \cdot r_2) = \mathcal{L}(r)$. At the same time, $(x \cdot s)^{\mathcal{I}_c} = x^{\mathcal{I}_c} s^{\mathcal{I}_c} = l_1 l_2$. Which means $\mathcal{I}_c$ satisfies the constraint $x \cdot s\,\mathsf{in}\,r$ and thus $\mathcal{I}_c$ automatically satisfies the premise configuration of the rule.

– Inter

$$\frac{\mathsf{R} := R,\ s\ \mathsf{in}\ r_1,\ s\ \mathsf{in}\ r_2}{\mathsf{R} := R,\ s\ \mathsf{in}\ (r_1 \sqcap r_2){\downarrow}}$$

Only If: $s\,\mathsf{in}\,r_1$ and $s\,\mathsf{in}\,r_2$ are satisfied in $\mathcal{I}_p$ iff $s^{\mathcal{I}_p} \in \mathcal{L}(r_1)$ and $s^{\mathcal{I}_p} \in \mathcal{L}(r_2)$, i.e., $s^{\mathcal{I}_p} \in \mathcal{L}(r_1 \cap r_2)$. By Lemma 1 and 2, $\mathcal{I}_p$ is an interpretation satisfying K_c.

If: By Lemma 1 and 2, $\mathcal{I}_c$ is an interpretation satisfying K_p.

- Union

$$\frac{\mathsf{R} := R,\ s\ \mathsf{in}\ r_1 \sqcup r_2}{\mathsf{R} := R,\ s\ \mathsf{in}\ r_1 \quad \| \quad \mathsf{R} := R,\ s\ \mathsf{in}\ r_2}$$

  Only If: s in $r_1 \sqcup r_2$ is satisfied in $\mathcal{I}_p$ iff $s^{\mathcal{I}_p} \in \mathcal{L}(r_1 \cup r_2)$. Thus, either $s^{\mathcal{I}_p} \in \mathcal{L}(r_1)$ or $s^{\mathcal{I}_p} \in \mathcal{L}(r_2)$ holds.
  If: If $s^{\mathcal{I}_c} \in \mathcal{L}(r_1)$, $s\,\mathsf{in}\,r_1 \sqcup r_2$ is satisfiable by $\mathcal{I}_c$, similarly to the other branch.

- For A-Conflict, EmptyS, and EmptyR, it is sufficient to show that premise configurations are unsatisfiable.

A.4 Proofs for Lemma 6

Lemma 16. *If $\langle A, R, V\rangle$ is a saturated leaf of a derivation tree with root $\langle A_0, R_0, \emptyset\rangle$ then for every string variable $x \in \mathcal{V}(R_0)$ there is a constraint of the form x in q in V.*

Proof. The only rules that remove a variable from $\mathcal{V}(\mathsf{R})$ are Assign-1 and Assign-2. Once a variable, say x, is removed from $\mathcal{V}(\mathsf{R})$ by either rule, a constraint of the form x in q is added to V.