Bounded Strings for Constraint Programming

Joseph D. Scott, Pierre Flener, and Justin Pearson
Department of Information Technology
Uppsala University, Sweden
Email: *first.last*@it.uu.se

***Abstract*—We present a domain for string decision variables of bounded length, combining features from fixed-length and unbounded-length string solvers to reason on an interval defined by languages of prefixes and suffixes. We provide a theoretical groundwork for constraint solving on this domain and describe propagation techniques for several common constraints.**

***Index Terms*—constraint programming; string constraints**

I. Introduction

Constraints over strings occur in a wide variety of real-world problems, especially in fields such as test case generation [1], program analysis [2], model checking [3], and web security [4]. In recent years several methods for solving string constraints have appeared, which may be broadly classified based on their treatment of string length.

For constraints on unbounded-length strings, solvers typically work by reasoning on automaton representations; see for example the regular domains of [5]. To avoid exponential blowup, these solvers are frequently restricted to a small set of constraints, such as the tools SUSHI [6] and DPRLE [7]. Recent work also includes tools such as REVENANT [3], which uses SAT and interpolation to check intersections of regular languages defined as symbolic automata; and STRSOLVE [8], a dedicated solver that handles expensive automata intersection operations by lazily constructing cross-products. These tools rely exclusively on automata operations; constraints that would be handled using simpler methods in other solvers (such as integer arithmetic constraints on string lengths) are encoded into automata, resulting in increased computational complexity.

Another approach is to solve constraints on strings of a fixed length. For example, the HAMPI tool provides a theory of fixed-length strings for satisfaction modulo theories (SMT) solvers, using the bit-vector solver STP to solve problems of a single string variable [9]. The bit-vector approach has also been explored in constraint programming (CP) [10]. More commonly, however, fixed-length approaches in CP reason at the level of symbols, providing constraints for regular [11], [12] or context-free [13], [14] languages that operate on a sequence of symbol variables. In this approach, string constraints are implemented in existing CP frameworks, allowing solvers to easily combine constraints on strings with constraints on individual symbols and other non-string variables.

A possibility lying between these two extremes is to handle strings with lengths that are not fixed, but bounded by an integer interval. Probably the best known bounded-length solver is KALUZA [15], which solves constraints in two stages. In the first, an SMT solver is used to find possible lengths for strings which satisfy explicit length constraints, length constraints implied by the string constraints of the problem, and any other integer constraints present in the problem. In the second stage, these lengths are applied to create a fixed-length bit-vector problem, solved with STP in the same manner as HAMPI. If the problem in the second stage is unsatisfiable, the first stage is repeated, with the addition of new constraints to avoid previously tried lengths. Other work, such as [16], [17], handles bounded-length strings by adding a string length reasoning component to unbounded automata approaches. In CP, [18] illustrates propagation for bounded-length versions of the fixed-length REGULAR and CFG constraints.

We propose a new bounded-length approach in CP. We begin by defining variables and constraints for bounded-length strings. We then describe the *affix domain* for bounded-length strings, which is based on languages of prefixes and suffixes. This domain allows us to reason about the content of string suffixes, even when the length of the string is unknown. We then describe propagation techniques for affix domains over constraints of string equality, concatenation, indexing, substring, reversal, and regular language membership.

This new domain fits into the framework of existing CP solvers, thereby making available the large number of constraints over scalar and set variables which already exist. Combined with these pre-existing non-string constraints, this new domain and its attendant constraints yield a solver at least as expressive as that handled by other bounded-length solvers. Indeed, it is generally more expressive, as it freely combines string and numeric reasoning, without restrictions on the number of variables or types of arithmetic constraint.

II. Preliminaries

A constraint satisfaction problem (CSP) [19] is a tuple $\langle X, D, C\rangle$ of *decision variables* $X = \langle X_1, \dots, X_n\rangle$, *domains* $D = \langle D_1, \dots, D_n\rangle$ such that $X_i \in D_i$, and *constraints* C. A constraint C_j is a pair $\langle R_j, S_j\rangle$ where R_j is a relation on the variables $S_j \subseteq X$, called the *scope* of C_j. An *assignment* is a tuple $A = \langle a_1, \dots, a_n\rangle$; A is a *solution* of the CSP if all $a_i \in D_i$, and every C_j is *satisfied*, i.e., the projection of A onto S_j is in R_j. $A(X_i)$ denotes the projection of A onto X_i.

In sequel, we denote decision variables in uppercase (N) to distinguish them from mathematical variables in lowercase (i). We refer to the current domain of a decision variable N as $\mathcal{D}(N)$, and, for an integer decision variable N, to the lower and upper bounds of the domain as $\underline{N}$ and $\overline{N}$, respectively, and denote the *original* upper bound as N_{orig}. We indicate

1082-3409/13 $31.00 © 2013 IEEE
DOI 10.1109/ICTAI.2013.155

arrays of mathematical variables or scalar decision variables with vector notation ($\vec{c}$, $\vec{A}$), and string decision variables with bold-face ($\mathbf{X}$). We denote an element of an array with the index in brackets ($\vec{A}[i]$). We use the notation $[\ell, u]$ to refer to the set of integers $\{\, \ell, \ell+1, \dots, u-1, u \,\}$.

An alphabet Σ is a finite set of symbols. A string w of length $|w| = n$ over the alphabet Σ is a finite sequence of elements of Σ, denoted $w_1 w_2 \cdots w_n$, where $w_i \in \Sigma$ for all $1 \leq i \leq n$. The set of all strings over Σ is denoted by Σ^*. A possibly infinite subset of $\mathcal{L} \subseteq \Sigma^*$ is called a *language* over Σ. Given a language $\mathcal{L}$ over Σ, for a string $w \in \mathcal{L}$ we say p is a *prefix* of w if there exists a string $x \in \Sigma^*$ such that $w = px$ (where px denotes the concatenation of p and x). Similarly, s is a *suffix* of w if there exists a string $y \in \Sigma^*$ such that $w = ys$; we refer to prefixes and suffixes collectively as *affixes*. The *reversal* of a word w is the word $w^{\text{rev}} = w_n \cdots w_2 w_1$. The reversal of a language $\mathcal{L}$ is the set $\mathcal{L}^{\text{rev}} = \{\, w^{\text{rev}} \mid w \in \mathcal{L} \,\}$.

III. Bounded-Length Strings

In this section, we describe a representation for string decision variables of bounded length, and discuss constraints on them. In Section IV we introduce a representation which we argue is more useful for propagation; however, it is convenient to specify constraints on bounded-length string decision variables in terms of the simpler representation introduced here.

A. Decision Variables

We will refer to unknown strings, over an arbitrary alphabet Σ, which occur in the model of a constraint satisfaction problem, as *string decision variables*. Abstractly, the domain of a string decision variable is a subset of Σ^*; however, we require a concrete representation. In [5], automata serve as a concrete representation, allowing for domains that are possibly infinite sets of strings; propagation is defined in terms of automata operations. Another choice of representation is a fixed-length sequence of scalar decision variables over Σ [11], [12]. The latter representation has been more widely adopted, primarily because of tighter integration with existing finite-domain solvers; additional constraints are stated over individual decision variables from the sequence in a natural way. In contrast, a value of a decision variable in [5] is a string, which does not readily facilitate interaction between individual symbols and other types of decision variables.

As we propose to treat bounded-length strings, it is natural to consider a representation based on the one used for bounded open constraints [18]. A constraint is global [20] if the cardinality of its scope is not determined *a priori*. In a *closed* global constraint, the cardinality of the scope is determined by the model, and remains constant throughout the solution process; however, in an *open* global constraint, the cardinality of the scope is determined as the solution procedure progresses [21]. In [18], the scope of an open global constraint is a sequence of scalar decision variables with a length that is bounded by an integer decision variable. This formulation leads to a string representation we will call the *open sequence representation*: $\langle \vec{A}, N \rangle$ consists of an array $\vec{A}$ of scalar decision variables over Σ, and an integer decision variable N representing the length of the string. In contrast to [18], we treat the sequence and length together as a representation of a single, bounded-length string decision variable, a syntactical difference that will be convenient when we define constraints over multiple variables in the next subsection. The domain of $\langle \vec{A}, N \rangle$ is defined in terms of the domains of the pair:

$$\mathcal{D}\Big(\Big\langle \vec{A}, N \Big\rangle\Big) = \bigcup_{n \in \mathcal{D}(N)} \Big\{\, a_1 \dots a_n \;\Big|\; \forall i \in [1, n]\colon a_i \in \mathcal{D}\big(\vec{A}[i]\big) \Big\}$$

B. Simple Constraints

Semantically, the relation of a constraint may be defined as a set of tuples over the scope of the constraint. We define string constraints in the open sequence representation.

The constraint $\textsc{Equal}(\mathbf{X}, \mathbf{Y})$ enforces string equality for string decision variables $\mathbf{X}$ and $\mathbf{Y}$, defined as strings having equal length and the same symbol at each index:

$$\begin{gathered}\textsc{Equal}\left(\Big\langle \vec{A}^x, N^x \Big\rangle, \Big\langle \vec{A}^y, N^y \Big\rangle\right) = \Big\{ \Big(\langle \vec{b}, p \rangle, \langle \vec{c}, q \rangle \Big) \Big| \\ p \in \mathcal{D}(N^x) \wedge q \in \mathcal{D}(N^y) \wedge p = q \wedge \forall i \in [1, p]\colon \\ \Big(\vec{b}[i] \in \mathcal{D}\big(\vec{A}^x[i]\big) \wedge \vec{c}[i] \in \mathcal{D}\big(\vec{A}^y[i]\big) \wedge \vec{b}[i] = \vec{c}[i] \Big) \Big\}\end{gathered}$$

The constraint $\textsc{Reverse}(\mathbf{X}, \mathbf{Y})$, for string decision variables $\mathbf{X}$ and $\mathbf{Y}$, states that $\mathbf{X}$ is equal to the reverse of $\mathbf{Y}$ (the specification differs from $\textsc{Equal}$ only in the indexing of $\mathbf{Y}$):

$$\begin{gathered}\textsc{Reverse}\left(\Big\langle \vec{A}^x, N^x \Big\rangle, \Big\langle \vec{A}^y, N^y \Big\rangle\right) = \Big\{ \Big(\langle \vec{b}, p \rangle, \langle \vec{c}, q \rangle \Big) \Big| \\ p \in \mathcal{D}(N^x) \wedge q \in \mathcal{D}(N^y) \wedge p = q \wedge \forall i \in [1, p]\colon \\ \Big(\vec{b}[i] \in \mathcal{D}\big(\vec{A}^x[i]\big) \wedge \vec{c}[i] \in \mathcal{D}\big(\vec{A}^y[i]\big) \wedge \vec{b}[i] = \vec{c}[p - i + 1] \Big) \Big\}\end{gathered}$$

$\textsc{Concat}(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$ states that string decision variable $\mathbf{Z}$ equals the concatenation of string decision variables $\mathbf{X}$ and $\mathbf{Y}$:

$$\begin{gathered}\textsc{Concat}\left(\Big\langle \vec{A}^x, N^x \Big\rangle, \Big\langle \vec{A}^y, N^y \Big\rangle, \Big\langle \vec{A}^z, N^z \Big\rangle\right) = \\ \Big\{ \Big(\langle \vec{b}, p \rangle, \langle \vec{c}, q \rangle, \langle \vec{d}, r \rangle \Big) \Big| \, p \in \mathcal{D}(N^x) \wedge q \in \mathcal{D}(N^y) \\ \wedge\, r \in \mathcal{D}(N^z) \wedge p + q = r \wedge \forall i \in [1, p]\colon \\ \Big(\vec{b}[i] = \vec{d}[i] \wedge \vec{b}[i] \in \mathcal{D}\big(\vec{A}^x[i]\big) \wedge \vec{d}[i] \in \mathcal{D}\big(\vec{A}^z[i]\big) \Big) \\ \wedge\, \forall j \in [1, q]\colon \Big(\vec{c}[j] = \vec{d}[j{+}p{-}1] \wedge \vec{c}[j] \in \mathcal{D}\big(\vec{A}^y[j]\big) \\ \wedge\, \vec{d}[j{+}p{-}1] \in \mathcal{D}\big(\vec{A}^z[j{+}p{-}1]\big) \Big) \Big\}\end{gathered}$$

$\textsc{Substring}(\mathbf{X}, \mathbf{Y}, I)$ states that string decision variable $\mathbf{Y}$ is a contiguous substring of string decision variable $\mathbf{X}$, starting at the index given by the integer decision variable I. The special case when $\mathbf{Y}$ has a length of one (i.e., where the second decision variable is scalar) is defined as $\textsc{CharacterAt}(\mathbf{X}, C, I)$:

$$\begin{gathered}\textsc{Substring}\left(\Big\langle \vec{A}^x, N^x \Big\rangle, \Big\langle \vec{A}^y, N^y \Big\rangle, I\right) = \\ \Big\{ \Big(\langle \vec{b}, p \rangle, \langle \vec{c}, q \rangle, r \Big) \Big| \, p \in \mathcal{D}(N^x) \wedge q \in \mathcal{D}(N^y) \wedge r \in \mathcal{D}(I) \\ \wedge\, p \geq q + r - 1 \wedge \forall i \in [1, p]\colon \vec{b}[i] \in \mathcal{D}\big(\vec{A}^x[i]\big) \wedge \\ \forall j \in [1, q]\colon \Big(\vec{c}[j] \in \mathcal{D}\big(\vec{A}^y[j]\big) \wedge \vec{b}[r + j - 1] = \vec{c}[j] \Big) \Big\}\end{gathered}$$

$$\textsc{CharacterAt}\left(\left\langle \vec{A}^x, N^x \right\rangle, C, I\right) = \\ \left\{ \left(\langle \vec{b}, p\rangle, q, r\right) \middle| \, p \in \mathcal{D}(N^x) \wedge q \in \mathcal{D}(C) \\ \wedge\, r \in \mathcal{D}(I) \wedge \forall i \in [1,p] \colon \vec{b}[i] \in \mathcal{D}\big(\vec{A}^x[i]\big) \wedge \vec{b}[r] = q \right\}$$

C. Language Membership Constraints

$\textsc{Regular}(\mathcal{L}, \mathbf{X})$ states that string decision variable $\mathbf{X}$ must belong to the given regular language $\mathcal{L}$. The specification would be valid for any language, irrespective of regularity; however, for bounded-length strings we only know of propagation techniques when $\mathcal{L}$ is regular or context-free [18].

$$\textsc{Regular}\left(\mathcal{L}, \left\langle \vec{A}^x, N^x \right\rangle\right) = \left\{ \left(\mathcal{L}, \langle \vec{b}, p\rangle\right) \middle| \, p \in \mathcal{D}(N^x) \\ \wedge\, \forall i \in [1,p] \colon \vec{b}[i] \in \mathcal{D}\big(\vec{A}[i]\big) \wedge \vec{b}[1]\vec{b}[2] \ldots \vec{b}[p] \in \mathcal{L} \right\}$$

IV. Affix Domains

When searching for strings in the languages for which reversal may be efficiently computed (i.e., the regular languages), it is possible to make strong inferences not just about the beginnings of strings, but also about their ends. Propagation over fixed-length string decision variables takes advantage of this fact, for example in the backwards pruning phase of [11]. Extending fixed-length methods to bounded length using the open sequence representation [18], this source of inference is lost: the last symbols of a string decision variable represented as $\langle \vec{A}, N\rangle$ have unknown indices in $\vec{A}$ until N is fixed.

In order to make better use of this source of inference, we propose a domain based on the concatenation of two languages, determining a set of strings based on both their beginning and their end. We call these languages the *prefix language* and *suffix language* (collectively, the *affix languages*) of the domain. This approach could be extended to a concatenation over arbitrarily many languages, thereby allowing for reasoning over several discontinuous regions of satisfying strings; however, we believe that reasoning starting from the outside of the string and moving towards the inside should result in stronger pruning, as the beginning and end of the strings are usually more clearly determined by the constraints.

A. Domains

The *affix representation* of a string decision variable $\mathbf{X}$ is given by a tuple $\langle \vec{P}, P, \vec{S}, S\rangle$, where the components P and S are integer decision variables, $\vec{P}$ is an array of scalar decision variables $\langle \vec{P}[1], \ldots, \vec{P}[\overline{P}]\rangle$ over Σ, and $\vec{S}$ is an array of scalar decision variables $\langle \vec{S}[1], \ldots, \vec{S}[\overline{S}]\rangle$ over Σ. When required for clarity, we will distinguish the components of $\mathbf{X}$ from those of other string decision variables by denoting them P^x, etc.

We construct the following bounded-length affix languages:

$$\mathcal{L}(\vec{P}, P) = \bigcup_{k \in \mathcal{D}(P)} \left\{ p_1 \ldots p_k \,\middle|\, \forall i \in [1,k] \colon p_i \in \mathcal{D}\big(\vec{P}[i]\big) \right\}$$

$$\mathcal{L}(\vec{S}, S) = \bigcup_{k \in \mathcal{D}(S)} \left\{ s_1 \ldots s_k \,\middle|\, \forall i \in [1,k] \colon s_i \in \mathcal{D}\big(\vec{S}[k-i+1]\big) \right\}$$

Definition 1. The *affix domain* of $\mathbf{X}$ is the concatenation of the affix languages:

$$\mathcal{D}\left(\left\langle \vec{P}, P, \vec{S}, S\right\rangle\right) = \left\{ ps \,\middle|\, p \in \mathcal{L}(\vec{P}, P), s \in \mathcal{L}(\vec{S}, S) \right\}$$

As strings in $\mathcal{L}(\vec{S}, S)$ are constructed from $\vec{S}$ by *decreasing* index, the first symbol in $\vec{S}$ is the last symbol of any word in $\mathcal{D}(\mathbf{X})$. Each assignment of the affix representation is mapped to a unique open sequence representation assignment via the semantic function:

$$\left[\!\!\left[\left\langle \vec{P}, P, \vec{S}, S \right\rangle \right]\!\!\right] = \left\langle \langle \vec{P}[1], \ldots, \vec{P}[P], \vec{S}[S], \ldots, \vec{S}[1] \rangle, P + S \right\rangle$$

In the examples which follow, we take as our alphabet the set of symbols corresponding to the lowercase, English characters, denoted `a`, `b`, etc. We write [`p`–`s`] to represent the lexicographic interval $\{$ `p`, `q`, `r`, `s` $\}$. A singleton lexicographic interval $\{$ `p` $\}$ or integer interval $\{ 1 \}$ we denote as a bare element, i.e., `p` or 1. We denote a concrete string as '`example`'.

Example 1 Consider a concrete affix representation:

$$\alpha = \langle \langle [\texttt{p–s}], [\texttt{a–z}], [\texttt{a–z}], [\texttt{f–x}] \rangle, [1,3], \langle \texttt{x}, \texttt{i}, \texttt{f}, [\texttt{f–x}] \rangle, [2,4] \rangle$$

Let $\mathbf{A}$ be a string decision variable, where the current domains of the components of the affix representation are the components of α. Here $\vec{P^a}$ has 4 elements, corresponding to the *original* upper bound on the length of strings in the prefix language, as opposed to $\overline{P^a} = 3$, which is the *current* upper bound on that length. The strings '`prefix`' and '`suffix`' are both in $\mathcal{D}(\mathbf{A})$, while '`affix`' is not (since `a` $\notin$ [`p`–`s`]). Also, the prefix language of $\mathbf{A}$ does not include the string '`post`', because its length exceeds $\overline{P^a} = 3$; nevertheless, '`postfix`' is in $\mathcal{D}(\mathbf{A})$, as a concatenation of '`pos`' and '`tfix`'. ◇

The semantic function is surjective, but *not* injective: a string defined in the open sequence representation may have several equivalent definitions in the affix representation. This is, however, sufficient for the purposes at hand; namely, propagators which act on the underlying affix representation may, by means of the semantic function, implement constraints which are defined in terms of the open sequence representation.

The affix representation $\langle \vec{P}, P, \vec{S}, S\rangle$ of a string decision variable which is not yet fixed may be mapped to an open sequence representation $\langle \vec{A}, N\rangle$, where $N = P + S$ and for every index i in the array $\vec{A}$:

$$\mathcal{D}\big(\vec{A}[i]\big) = \begin{cases} \mathcal{D}\big(\vec{P}[i]\big) & \text{if } i \in [1, \underline{P}\,] \\ \mathcal{D}\big(\vec{P}[i]\big) \cup \displaystyle\bigcup_{j=\max(\underline{P}+\underline{S}-i+1,1)}^{\overline{S}} \mathcal{D}\big(\vec{S}[j]\big) & \text{if } i \in \big[\underline{P}+1, \overline{P}\,\big] \\ \displaystyle\bigcup_{k=\max(\underline{P}+\underline{S}-i,1)}^{\overline{P}+\overline{S}-i+1} \mathcal{D}\big(\vec{S}[k]\big) & \text{if } i \in \big[\,\overline{P}+1, \overline{P}+\overline{S}\,\big] \end{cases}$$

Example 2 The affix representation

$$\langle \langle [\texttt{p–s}], [\texttt{a–z}], [\texttt{a–z}], [\texttt{f–x}] \rangle, [1,4], \langle \texttt{x}, \texttt{i}, \texttt{f}, [\texttt{f–x}] \rangle, [2,3] \rangle$$

and the representation α from Example 1 both map to the open sequence representation

$$\big\langle \langle \texttt{[p–s]}, \texttt{[a–z]}, \texttt{[a–z]}, \texttt{[f–x]}, \{\, \texttt{f}, \texttt{i}, \texttt{x} \,\}, \{\, \texttt{i}, \texttt{x} \,\}, \texttt{x} \rangle, [3,7] \big\rangle$$

$\diamond$

B. Consistency

We extend open domain consistency [18] to account for the interaction between affix languages, and for the presence of multiple string decision variables in the scope of a single constraint. Informally, every symbol in the domain of every affix element variable participates at its index in a solution string, and there is a solution for every length in the domain of each affix length variable. Formally:

Definition 2 (PS-consistency)**.** Given a constraint C and a bounded-length string decision variable $\mathbf{X} = \langle \vec{P}, P, \vec{S}, S \rangle$ in its scope, the domain of $\mathbf{X}$ is *PS-consistent* if

- for every $\vec{P}[i] \in \vec{P}$ and every $v \in \mathcal{D}(\vec{P}[i])$, there exists a string $p = p_1 \ldots p_n$ such that $n \in \mathcal{D}(P)$, $p_i = v$, $p_j \in \mathcal{D}(\vec{P}[j])$ for $j \in [1, n]$, a string $s \in \mathcal{L}(\vec{S}, S)$, and an assignment A satisfying C such that $A(\mathbf{X}) = ps$, and
- for every $n \in \mathcal{D}(P)$ there exists a string $p \in \mathcal{L}(\vec{P}, P)$ of length $|p| = n$, a string $s \in \mathcal{L}(\vec{S}, S)$, and an assignment A satisfying C such that $A(\mathbf{X}) = ps$, and
- for every $\vec{S}[i] \in \vec{S}$ and every $v \in \mathcal{D}(\vec{S}[i])$, there exists a string $s = s_1 \ldots s_n$ such that $n \in \mathcal{D}(S)$, $s_i = v$, $s_j \in \mathcal{D}(\vec{S}[j])$ for $j \in [1, n]$, a string $p \in \mathcal{L}(\vec{P}, P)$, and an assignment A satisfying C such that $A(\mathbf{X}) = ps$, and
- for every $n \in \mathcal{D}(S)$ there exists a string $s \in \mathcal{L}(\vec{S}, S)$ of length $|s| = n$, a string $p \in \mathcal{L}(\vec{P}, P)$, and an assignment A satisfying C such that $A(\mathbf{X}) = ps$.

We define *PSL-consistency* as a weakening of PS-consistency that considers only prefixes of length $\underline{P}$ and suffixes of length $\underline{S}$; *PSU-consistency* is the same for maximal-length affixes, and *PSLU-consistency* for both minimal- and maximal-length affixes.

C. Domain Strength

Affix domains are sets of strings; for a pair of affix domains $\mathcal{D}$ and $\mathcal{D}'$ we say that $\mathcal{D}'$ is *stronger* (resp. *strictly stronger*) than $\mathcal{D}$ iff $\mathcal{D}' \subseteq \mathcal{D}$ (resp. $\mathcal{D}' \subset \mathcal{D}$). From Definition 1 it is clear that strengthening the domain of an affix representation $\langle \vec{P}, P, \vec{S}, S \rangle$ may be accomplished by removing strings from either $\mathcal{L}(\vec{P}, P)$ or $\mathcal{L}(\vec{S}, S)$. We now describe the removal of strings from the prefix language; the description applies equally to the suffix language.

The prefix length P may be used to partition the indices of $\vec{P}$ into three ranges: the *required range* of the indices up to the current lower bound of P, the *optional range* of the indices beyond the current lower and until the current upper bound, and the *discarded range* of the indices beyond the current upper bound:

$$\vec{P}_{\text{req}} = [1, \underline{P}] \qquad \vec{P}_{\text{opt}} = [\underline{P}+1, \overline{P}] \qquad \vec{P}_{\text{dis}} = [\overline{P}+1, P_{\text{orig}}]$$

The prefix language may be strengthened by tightening the bounds on P, thereby increasing the required and/or discarded ranges, and decreasing the optional range. In addition, the prefix language may be strengthened by removing values from the domains of the scalar decision variables with indices in the required or optional range.

Example 3 Let the components of the affix representations of string decision variables $\mathbf{B}$ and $\mathbf{C}$ be the components of, respectively:

$$\beta = \big\langle \langle \texttt{p}, \texttt{r}, \texttt{[a–z]}, \texttt{[a–z]} \rangle, [1,3], \langle \texttt{x}, \texttt{i}, \texttt{f}, \texttt{[f–x]} \rangle, [2,4] \big\rangle$$
$$\gamma = \big\langle \langle \texttt{[p–s]}, \texttt{[a–z]}, \texttt{[a–z]}, \texttt{[a–z]} \rangle, [1,3], \langle \texttt{x}, \texttt{i}, \texttt{f}, \texttt{[f–x]} \rangle, 4 \big\rangle$$

Comparing with $\mathbf{A}$ from Example 1, we have $\mathcal{D}(P^b) = \mathcal{D}(P^a)$, $\mathcal{D}(\vec{P}^b[i]) \subseteq \mathcal{D}(\vec{P}^a[i])$ for all $i \in [1, \overline{P^b}]$, and $\mathcal{D}(\vec{P}^b[1]) \subset \mathcal{D}(\vec{P}^a[1])$; therefore, $\mathcal{L}(\vec{P}^b, P^b) \subset \mathcal{L}(\vec{P}^a, P^a)$. Since $\mathcal{L}(\vec{S}^b, S^b) = \mathcal{L}(\vec{S}^a, S^a)$, this results in $\mathcal{D}(\mathbf{B}) \subset \mathcal{D}(\mathbf{A})$. On the other hand, we have $\mathcal{D}(\vec{P}^c[i]) \subseteq \mathcal{D}(\vec{P}^a[i])$ for all $i \in [1, \overline{P^c}]$; yet we still have $\mathcal{L}(\vec{S}^c, S^c) \subset \mathcal{L}(\vec{S}^a, S^a)$ due to $\mathcal{D}(S^c) \subset \mathcal{D}(S^a)$. Since $\mathcal{L}(\vec{P}^c, P^c) = \mathcal{L}(\vec{P}^a, P^a)$, $\mathcal{D}(\mathbf{C}) \subset \mathcal{D}(\mathbf{A})$. The domains $\mathcal{D}(\mathbf{B})$ and $\mathcal{D}(\mathbf{C})$ are not comparable, as illustrated by the strings ‘`prefix`’ and ‘`suffix`’; while both are elements of $\mathcal{D}(\mathbf{A})$, ‘`prefix`’ is an element of $\mathcal{D}(\mathbf{B})$ but not $\mathcal{D}(\mathbf{C})$, and ‘`suffix`’ is an element of $\mathcal{D}(\mathbf{C})$ but not $\mathcal{D}(\mathbf{B})$. $\diamond$

Any propagator which only removes values from domains of the elements of the affix arrays, or tightens the integer bounds on an affix length, must be contracting. We further note that each of these operations is finitely bounded: P and S are finitely bounded integers, which cannot be tightened indefinitely, while the finity of Σ guarantees that $\vec{P}_{\text{req}}$ and $\vec{S}_{\text{req}}$ must eventually converge to concrete strings.

V. Propagation

Next, we describe propagators for constraints over affix domains. The constraints given in Section III-B state relationships between the components of the open sequence representations of bounded-length string decision variables, but do not imply the same relationships among the components of their affix representations. For example, in the open sequence representation, $\textsc{Equal}(\langle \vec{A}^x, N^x \rangle, \langle \vec{A}^y, N^y \rangle)$ implies both $\vec{A}^x = \vec{A}^y$ and $N^x = N^y$, but it does not imply either $\vec{P}^x = \vec{P}^y$ or $P^x = P^y$ in a corresponding affix representation. Propagation of $\textsc{Equal}$ over affix domains must account for cases where a region of $\vec{P}^x$ must be equal to a region of $\vec{P}^y$, but also cases where a region of $\vec{P}^x$ must be equal to the reverse of a region of $\vec{S}^y$; for $\vec{S}^x$ there exist symmetric cases.

Example 4 Continuing from Example 3, consider the constraint $\textsc{Equal}(\mathbf{B}, \mathbf{C})$. In any satisfying assignment, the symbols represented by $\vec{P}^b[1]$ and $\vec{P}^c[1]$ must be the same; no similar deduction is possible for $\vec{P}^b[2]$ and $\vec{P}^c[2]$, as $\underline{P^b} = \underline{P^c} = 1$. Pruning the domain of $\mathbf{C}$ may be accomplished by giving it the affix representation:

$$\gamma' = \big\langle \langle \texttt{p}, \texttt{[a–z]}, \texttt{[a–z]}, \texttt{[a–z]} \rangle, [1,3], \langle \texttt{x}, \texttt{i}, \texttt{f}, \texttt{[f–t]} \rangle, 4 \big\rangle$$

which is strictly stronger than γ. The domain of $\mathbf{B}$ may also be pruned, but only by reducing the domains of the affix lengths to reflect $P^b + S^b \in [5, 7]$; i.e., the current interval of $P^c + S^c$. Pruning of affix lengths is discussed in Section V-B1. $\diamond$

Example 5 If $\mathbf{B}$ has an affix representation stronger than β, however, then an additional region of alignment may be reasoned upon. For example:

$$\beta' = \big\langle \langle \mathtt{p}, \mathtt{r}, [\mathtt{a–z}], [\mathtt{a–z}] \rangle, [1, 3], \langle \mathtt{x}, \mathtt{i}, \mathtt{f}, [\mathtt{f–t}] \rangle, 2 \big\rangle$$

If the components of the affix representations of $\mathbf{B}$ and $\mathbf{C}$ are β' and γ' (of Example 4), respectively, then the length of any satisfying string must be 5 (being both the minimum of $P^b + S^b$ and the maximum of $P^c + S^c$); furthermore, in any solution the region $\vec{P}^b[2]$ through $\vec{P}^b[3]$ must align with the region $\vec{S}^c[4]$ through $\vec{S}^c[3]$. As $\mathcal{D}(\vec{P}^b[3]) \cap \mathcal{D}(\vec{S}^c[3]) = \emptyset$, there can be no satisfying solution. $\diamond$

In Section V-A we define a pair of primitive pruning operations that filter the domains of symbols in overlapping regions of two affixes at a time. Then, in Section V-B we use these operations to construct propagators for the constraints specified in Section III-B. In Section V-C we discuss propagating a regular language membership constraint.

A. Pruning Regions

A large portion of the filtering required for the propagation of the constraints in Section III-B may be generally described as a sequence of successive equalities between the symbols occurring in a range of indices in an affix of one variable, and a same-sized range in an affix of another variable. We define this operation as $\textsc{EqRegion}(\vec{Q}, Q, \vec{R}, R, D)$, where $\vec{Q}$ and $\vec{R}$ are arrays of scalar decision variables, and Q, R, and D are integer decision variables, where $\mathcal{L}(\vec{Q}, Q)$ and $\mathcal{L}(\vec{R}, R)$ are affix languages of two distinct string decision variables. The integer D represents the distance (in number of symbols) between the position of $\vec{Q}[1]$ and $\vec{R}[1]$; in other words, when D is fixed, $\mathcal{D}(\vec{Q}[D+1])$ must be equal to $\mathcal{D}(\vec{R}[1])$. In some cases, this distance will be a fixed quantity; however, for the sake of generality, we need to treat the distance as an integer decision variable. The domain of the distance defines the possible relative alignments of the two affix bounds.

To cover all the cases of aligned regions between the arguments of the constraints, two versions of $\textsc{EqRegion}$ are required. The versions differ only in the relative *direction* of these two affixes: a pair of prefixes or pair of suffixes is said to be *unidirectional*, while a mixed prefix-suffix pair is said to be *bidirectional*.

1) Unidirectional Region: The unidirectional pruning operation $\textsc{EqRegion}^{\mathrm{U}}(\vec{Q}, Q, \vec{R}, R, D)$ makes two types of inferences. The first considers elements of $\vec{Q}$ with an index in $\vec{Q}_{\mathrm{req}}$ that align with some element of $\vec{R}$ with an index in $\vec{R}_{\mathrm{req}}$, for every distance in $\mathcal{D}(D)$. Each value in the domains of these elements of $\vec{Q}$ requires a witness in the union of the domains of the elements of $\vec{R}$ with which it aligns. There is a symmetric requirement for elements of $\vec{R}$ with index in $\vec{R}_{\mathrm{req}}$ that align with some element of $\vec{Q}$ with an index in $\vec{Q}_{\mathrm{req}}$.

The second type of inference concerns the possible values of D: for each distance, the domains of the elements of $\vec{Q}$ and $\vec{R}$ that would be aligned must have some value in common, or the distance can be removed from $\mathcal{D}(D)$.

$$\begin{aligned}
&\textsc{EqRegion}^{\mathrm{U}}\Big(\vec{Q}, Q, \vec{R}, R, D\Big) \equiv \\
&\forall i \in \vec{Q}_{\mathrm{req}} \cap \big[\,\overline{D}, \underline{R} + \underline{D} - 1\big] : \Big(\vec{Q}[i] \in \bigcup_{k \in \mathcal{D}(D)} \mathcal{D}\Big(\vec{R}[i-k]\Big) \\
&\qquad \wedge D \in \Big\{\, k \;\Big|\; \mathcal{D}\Big(\vec{R}[i-k]\Big) \cap \mathcal{D}\Big(\vec{Q}[i]\Big) \neq \emptyset \,\Big\}\Big) \\
&\wedge \forall j \in \vec{R}_{\mathrm{req}} \cap \big[-\underline{D}, \underline{Q} - \overline{D} - 1\big] : \Big(\vec{R}[j] \in \bigcup_{k \in \mathcal{D}(D)} \mathcal{D}\Big(\vec{Q}[k+j]\Big) \\
&\qquad \wedge D \in \Big\{\, k \;\Big|\; \mathcal{D}\Big(\vec{Q}[k+j]\Big) \cap \mathcal{D}\Big(\vec{R}[j]\Big) \neq \emptyset \,\Big\}\Big)
\end{aligned}$$

2) Bidirectional Region: For prefix-suffix or suffix-prefix alignments, the two affixes have opposite directions. So while $\vec{R}[1]$ and $\vec{Q}[D+1]$ align in both cases, the alignments between subsequent elements will differ. Otherwise, the reasoning performed is similar to that of $\textsc{EqRegion}^{\mathrm{U}}$: elements of $\vec{Q}$ that always align with some element of $\vec{R}$ have their domains filtered, and vice versa; while the domain of possible distances is filtered based on the domains of the elements that would become equal in that alignment.

$$\begin{aligned}
&\textsc{EqRegion}^{\mathrm{B}}\Big(\vec{Q}, Q, \vec{R}, R, D\Big) \equiv \\
&\forall i \in \vec{Q}_{\mathrm{req}} \cap \big[\,\overline{D} - \underline{R}, \underline{D} - 1\big] : \Big(\vec{Q}[i] \in \bigcup_{k \in \mathcal{D}(D)} \mathcal{D}\Big(\vec{R}[k{-}i{-}1]\Big) \\
&\qquad \wedge D \in \Big\{\, k \;\Big|\; \mathcal{D}\Big(\vec{R}[k{-}i{-}1]\Big) \cap \mathcal{D}\Big(\vec{Q}[i]\Big) \neq \emptyset \,\Big\}\Big) \\
&\wedge \forall j \in \vec{R}_{\mathrm{req}} \cap \big[\,\overline{D} - \underline{Q}, \underline{D} - 1\big] : \Big(\vec{R}[j] \in \bigcup_{k \in \mathcal{D}(D)} \mathcal{D}\Big(\vec{Q}[k{-}j{-}1]\Big) \\
&\qquad \wedge D \in \Big\{\, k \;\Big|\; \mathcal{D}\Big(\vec{Q}[k{-}i{-}1]\Big) \cap \mathcal{D}\Big(\vec{R}[j]\Big) \neq \emptyset \,\Big\}\Big)
\end{aligned}$$

B. Propagators

We proceed to describe propagators for the constraints of Section III-B in terms of the two primitive pruning operations defined in the previous subsection, with the addition of a few integer equalities on affix length bounds.

1) Equality: It seems likely that unifying $\mathbf{X}$ and $\mathbf{Y}$ in the model prior to constraint posting will generally be preferable to maintaining $\textsc{Equal}(\mathbf{X}, \mathbf{Y})$; however, this is not always possible, for example if one of the decision variables is used in a reified context. Propagation of equality over affix domains is significantly more complicated than for more familiar domains, as we must account for interactions between $\vec{P}^x$ and $\vec{S}^y$, and between $\vec{S}^x$ and $\vec{P}^y$. It is, nevertheless, simpler than propagation for the other constraints specified here; we therefore describe propagation of $\textsc{Equal}$ in some detail, to serve as an illustrative example of how propagation over affix representations functions.

Figure 1 provides a propagator for the $\textsc{Equal}$ constraint. Lines 2 to 12 correspond to $N^x = N^y$ in the open sequence

```
1  $\textsc{Equal}(\langle \vec{P}^x, P^x, \vec{S}^x, S^x \rangle, \langle \vec{P}^y, P^y, \vec{S}^y, S^y \rangle)$:
2      if $\underline{P^x} + \underline{S^x} < \underline{P^y} + \underline{S^y}$ then
3          $\underline{P^x} \leftarrow \max(\underline{P^x}, \min(\underline{P^y}, \underline{P^y} + \underline{S^y} - \underline{S^x}))$
4          $\underline{S^x} \leftarrow \max(\underline{S^x}, \min(\underline{S^y}, \underline{P^y} + \underline{S^y} - \underline{P^x}))$
5      else
6          $\underline{P^y} \leftarrow \max(\underline{P^y}, \min(\underline{P^x}, \underline{P^x} + \underline{S^x} - \underline{S^y}))$
7          $\underline{S^y} \leftarrow \max(\underline{S^y}, \min(\underline{S^x}, \underline{P^x} + \underline{S^x} - \underline{P^y}))$
8      $n \leftarrow \min(\overline{P^x} + \overline{S^x}, \overline{P^y} + \overline{S^y})$
9      $\overline{P^x} \leftarrow \min(\overline{P^x}, n - \underline{S^x})$
10     $\overline{S^x} \leftarrow \min(\overline{S^x}, n - \underline{P^x})$
11     $\overline{P^y} \leftarrow \min(\overline{P^y}, n - \underline{S^y})$
12     $\overline{S^y} \leftarrow \min(\overline{S^y}, n - \underline{P^y})$
13     $\textsc{EqRegion}^{\mathrm{U}}(\vec{P}^x, P^x, \vec{P}^y, P^y, 0)$
14     $\textsc{EqRegion}^{\mathrm{B}}(\vec{P}^x, P^x, \vec{S}^y, S^y, P^x + S^x)$
15     $\textsc{EqRegion}^{\mathrm{B}}(\vec{S}^x, S^x, \vec{P}^y, P^y, P^x + S^x)$
16     $\textsc{EqRegion}^{\mathrm{U}}(\vec{S}^x, S^x, \vec{S}^y, S^y, 0)$
```

Fig. 1. Propagation of the constraint Equal

representation. The naive translation into $P^x + S^x = P^y + S^y$ would fail to take advantage of all available information. Consider that an increase to $\underline{N^x}$ due to propagation could be achieved with an increase to $\underline{P^x}$, $\underline{S^x}$, or both; if, however, the required region of the prefix of $\mathbf{Y}$ is larger than that of $\mathbf{X}$, then increasing $\underline{P^x}$ is likely to result in filtering of domains for symbols in $\vec{P}^x$. In contrast, increasing $\underline{S^x}$ under those same circumstances will yield little, if any, filtering of symbol domains, as shown in the following example.

Example 6 Let the components of the affix representations of string decision variables $\mathbf{X}$ and $\mathbf{Y}$ be, respectively:

$$\big\langle \langle [\texttt{p–s}], [\texttt{a–z}], [\texttt{a–z}], \texttt{f} \rangle, [1,3], \langle \texttt{x}, \texttt{i}, [\texttt{a–z}], [\texttt{a–z}] \rangle, [2,4] \big\rangle$$
$$\big\langle \langle [\texttt{p–s}], [\texttt{r–u}], [\texttt{f–s}], [\texttt{f–x}] \rangle, 3, \langle \texttt{x}, \texttt{i}, \texttt{f}, [\texttt{a–f}] \rangle, [2,4] \big\rangle$$

The minimum length of any string in $\mathcal{D}(\mathbf{Y})$ is $\underline{P^y} + \underline{S^y} = 5$, so clearly one or both of $\underline{P^x} = 1$ and $\underline{S^x} = 2$ should be increased accordingly. Choosing $\underline{S^x} \leftarrow 4$ satisfies the length constraints, but results in rather limited pruning; i.e., it would not be valid to prune $\mathcal{D}(\vec{S}^x[3]) \leftarrow \{\, \texttt{f} \,\}$ based on $\mathcal{D}(\vec{S}^y[3])$, as depending on the value of P^x, $\vec{S}^x[3]$ might also align with $\vec{P}^y[2]$ or $\vec{P}^y[3]$. In contrast, the choice of $\underline{P^x} \leftarrow 3$ ensures that in all satisfying assignments $\vec{P}^x[2]$ aligns with $\vec{P}^x[2]$. ◇

Every domain of N may be represented by several P and S combinations, so we choose the most advantageous of these combinations. The best choice can be determined in constant time (see line 2) by comparing the length bounds of the two affix representations. For the pruning of the lower bounds of lengths of strings in the affixes (following the conditional at line 2), this means that the required region of each prefix (resp. suffix) may be enlarged to take advantage of information from the other prefix (resp. suffix), but only so long as the operation would not cause the minimum length of either string to exceed the minimum length of the other string. Adjustment to the maximum lengths of the affixes is then made based on the combination of these new affix minimum lengths, and the prior maximum bound of the string lengths (computed at line 8).

As each affix is anchored to one or the other end of its string, the prefix-prefix and suffix-suffix pairs each have a fixed distance of zero; as a result, the EqRegion$^{\mathrm{U}}$ operations at lines 13 and 16 perform quite strong pruning. For the bidirectional pairs, the distance is the length of the two strings; as long as this length is not fixed, EqRegion$^{\mathrm{B}}$ will be relatively weaker. This is exactly as we would expect: with an unknown string length, the precise alignment between prefix symbols of one string and suffix symbols of the other may only be known very late in the solution process, and it is only the domains of those symbols that must align with some symbol from the opposing affix for every feasible value of the string length which might be pruned.

The Equal propagator in Figure 1 is contracting; it is also checking, as any assignment must have a fixed length, at which point the relative positions of the symbols in the two strings are known. However, the reasoning on the required regions of $\mathbf{X}$ and $\mathbf{Y}$ is not quite sufficient to maintain PSL-consistency. This is due to a gap between the affix regions which are covered by the two primitive pruning operations described in the previous subsection. For example, when $\underline{P^y} < \underline{P^x} < \overline{P^y}$, then there may be elements of $\vec{P}^x$ that align with either one element from $\vec{P}^y$ or a region of elements from $\vec{S}^y$. The Equal propagator in Figure 1 is correct despite ignoring this additional case, as the affix elements affected are not covered by either the EqRegion$^{\mathrm{U}}$ or EqRegion$^{\mathrm{B}}$ cases. Furthermore, as long as the minimum affix lengths are pruned in the maximally advantageous method, as described above, the additional case seems to occur infrequently. Nevertheless, it should be possible to extend the propagator to cover these cases by adding a third primitive pruning operation which takes into account the optional regions of the affixes, using similar reasoning to that for required regions presented here. This should allow not only for a PSL-consistent propagator, but also for extension to one that is fully PS-consistent.

2) Reverse: The propagator for Reverse follows directly from that of Equal. The only required modification is the swapping of the parameters corresponding to the prefixes and suffixes of $\mathbf{Y}$ on lines 13 through 16 of Figure 1.

3) Concatenation: The propagator for Concat$(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$ given in Figure 2 has many more cases than Equal; the prefix of $\mathbf{Z}$ could have regions aligned with each of the four affixes of $\mathbf{X}$ and $\mathbf{Y}$, and there are another four possible alignments for the suffix of $\mathbf{Z}$. Nevertheless, each of these eight possible aligned regions may be expressed as an instance of one of the two defined pruning operations, with the calculation of a suitable distance value.

Updating minimum length of the affix bounds is also similar to Equal, although in the case of Concat there is less information available. Specifically, $\underline{S^x}$ and $\underline{P^y}$ are difficult to increase, because of their position inside $\mathbf{Z}$. The adjustment of $\underline{P^x}$ must consider not only $\underline{S^x}$, but also the upper bounds of the lengths of the affixes of $\mathbf{Y}$, in order to avoid unjustified pruning on N^y; adjustment of $\underline{S^y}$ is symmetric.

```
CONCAT($\langle\vec{P}^x, P^x, \vec{S}^x, S^x\rangle,$
        $\langle\vec{P}^y, P^y, \vec{S}^y, S^y\rangle, \langle\vec{P}^z, P^z, \vec{S}^z, S^z\rangle$):
    if $\underline{P^z} + \underline{S^z} < \underline{P^x} + \underline{S^x} + \underline{P^y} + \underline{S^y}$ then
        $\underline{P^z} \leftarrow \max(\underline{P^z}, \min(\underline{P^x}, \underline{P^x}+\underline{S^x}+\underline{P^y}+\underline{S^y}-\underline{S^z}))$
        $\underline{S^z} \leftarrow \max(\underline{S^z}, \min(\underline{S^y}, \underline{P^x}+\underline{S^x}+\underline{P^y}+\underline{S^y}-\underline{P^z}))$
    else
        $\underline{P^x} \leftarrow \max(\underline{P^x}, \min(\underline{P^z}, \underline{P^z}+\underline{S^z}-(\overline{P^y}+\overline{S^y})-\underline{S^x}))$
        $\underline{S^y} \leftarrow \max(\underline{S^y}, \min(\underline{S^z}, \underline{P^z}+\underline{S^z}-(\overline{P^x}+\overline{S^x})-\underline{P^y}))$
    $n \leftarrow \min(\overline{P^z} + \overline{S^z}, \overline{P^x} + \overline{S^x} + \overline{P^y} + \overline{S^y})$
    $\overline{P^z} \leftarrow \min(\overline{P^z}, n - \underline{S^z})$
    $\overline{S^z} \leftarrow \min(\overline{S^z}, n - \underline{P^z})$
    $\overline{P^x} \leftarrow \min(\overline{P^x}, n - (\underline{P^y} + \underline{S^y} + \underline{S^x}))$
    $\overline{S^x} \leftarrow \min(\overline{S^x}, n - (\underline{P^y} + \underline{S^y} + \underline{P^x}))$
    $\overline{P^y} \leftarrow \min(\overline{P^y}, n - (\underline{P^x} + \underline{S^x} + \underline{S^y}))$
    $\overline{S^y} \leftarrow \min(\overline{S^y}, n - (\underline{P^x} + \underline{S^x} + \underline{P^y}))$
    EQREGION$^{\mathrm{U}}(\vec{P}^z, P^z, \vec{P}^x, P^x, 0)$
    EQREGION$^{\mathrm{B}}(\vec{P}^z, P^z, \vec{S}^x, S^x, P^x + S^x)$
    EQREGION$^{\mathrm{U}}(\vec{P}^z, P^z, \vec{P}^y, P^y, P^x + S^x)$
    EQREGION$^{\mathrm{B}}(\vec{P}^z, P^z, \vec{S}^y, S^y, P^z + S^z)$
    EQREGION$^{\mathrm{U}}(\vec{S}^z, S^z, \vec{S}^y, S^y, 0)$
    EQREGION$^{\mathrm{B}}(\vec{S}^z, S^z, \vec{P}^y, P^y, P^y + S^y)$
    EQREGION$^{\mathrm{U}}(\vec{S}^z, S^z, \vec{S}^x, S^x, P^y + S^y)$
    EQREGION$^{\mathrm{B}}(\vec{S}^z, S^z, \vec{P}^x, P^x, P^z + S^z)$
```

Fig. 2. Propagation of the constraint CONCAT.

The CONCAT propagator also does not maintain PSL-consistency. The cases missed by the EQUAL propagator described in the last section are missed here, as well. Furthermore, the possible alignments encountered while propagating CONCAT allow for cases in which an element from an affix array of one string decision variable might align with ranges of elements from more than one affix array of another string decision variable. These cases are somewhat more difficult to generalize than the additional case discussed in EQUAL, and further study is required to determine a method for achieving PSL-consistency, and ideally PS-consistency, for CONCAT.

4) Character At Position and Substring: SUBSTRING$(\mathbf{X}, \mathbf{Y}, I)$ may also be decomposed into a series of equalities of aligned regions, as shown in Figure 3. In this case, the distance between the regions must also take into account the decision variable I, representing the index of the starting position of $\mathbf{Y}$ in $\mathbf{X}$. Unfortunately, this alteration results in rather weak pruning in general. A symbol in either string which, in any satisfying solution, aligns with one of a set of symbols in the required region of one of the other string's affixes is pruned essentially as if it were subject to an ELEMENT constraint: each element of the domain of the individual symbol must have a witness in the union of the domains of the symbols with which it might align, and the index I is filtered to remove positions, resulting in an empty intersection of domains. The SUBSTRING propagator described here does not maintain PSL-consistency, for the same reasons described for CONCAT.

```
SUBSTRING($\langle\vec{P}^x, P^x, \vec{S}^x, S^x\rangle, \langle\vec{P}^y, P^y, \vec{S}^y, S^y\rangle, I$):
    $P^x + S^x \geq I + P^y + S^y$
    EQREGION$^{\mathrm{U}}(\vec{P}^x, P^x, \vec{P}^y, P^y, I)$
    EQREGION$^{\mathrm{B}}(\vec{P}^x, P^x, \vec{S}^y, S^y, I + P^y + S^y)$
    EQREGION$^{\mathrm{B}}(\vec{S}^x, S^x, \vec{P}^y, P^y, P^x + S^x - I)$
    EQREGION$^{\mathrm{U}}(\vec{S}^x, S^x, \vec{S}^y, S^y, P^x+S^x-(I+P^y+S^y))$
```

Fig. 3. Propagation of the constraint SUBSTRING propagator

CHARACTERAT propagation is identical, for $\mathbf{Y}$ of length 1.

C. Regular Language Membership

Given a finite automaton $\mathcal{M}$ that specifies $\mathcal{L}$, an automaton $\mathcal{M}^{\text{pre}}$ accepting the bounded-length prefixes of strings in $\mathcal{L}$ is constructed by unrolling $\mathcal{M}$ to length $\overline{P}$, and making all states with distance at least $\underline{P}$ from the start state be accepting states in $\mathcal{M}^{\text{pre}}$ (as long as they fall on a path leading to an accepting state in $\mathcal{M}$). Similarly, an automaton $\mathcal{M}^{\text{suf}}$ accepting bounded-length suffixes is constructed from $\mathcal{M}^{\text{rev}}$. We then make use of the regular constraint for sequences of bounded length described in [18], which we refer to as OPENREGULAR. From this we get the decomposition into OPENREGULAR$(\mathcal{M}^{\text{pre}}, \vec{P}, P)$ and OPENREGULAR$(\mathcal{M}^{\text{suf}}, \vec{S}, S)$.

VI. PRELIMINARY RESULTS

This theoretical paper aims at laying a self-contained foundation for bounded-length string decision variables in CP. A full experimental evaluation is orthogonal to this purpose, and would have to be omitted for space reasons. Nevertheless, we implemented a prototype of the propagators described here using the extended indexical language described in [22]. An *indexical* [23] is an expression of the form $x \in \sigma$ restricting the domain of the decision variable x to the intersection of its current domain and the interval σ. An indexical language is a high-level language for propagator description; [22] provides a compiler which generates propagator descriptions from checkers, and also has several solver-specific back-ends which allow for compilation of the propagator into source code.

We used this compiler to generate prototypes, in Gecode 3.7.3 [24], of propagators for the constraints given here (exclusive of REGULAR), using both the affix representation and the open sequence representation. We compared performance on several randomly generated concatenation problems. In each instance, a concrete string of length 100 to 200, over an alphabet of size ten, was generated. CONCAT constraints were used to constrain seven bounded-length string decision variables to be equal to the concrete string, with the variables occurring in a fixed sequence that included a single repeated variable. Branch and bound search was used to find the solution with the maximum length for the repeated variable (i.e., the longest repeated symbol sequence in the concrete string).

Preliminary results were encouraging. Despite the increased complexity inherent in propagating the affix representation, over several hundred instances we observed very little variation in runtime. It is possible that the constraints in the randomly generated problems were too loose to force the execution of the more computationally expensive portions of the propagators; it seems likely that hand-crafted test cases will be required to properly evaluate the impact of these portions on performance. More strongly constrained test cases should also better demonstrate the impact of the affix representation on search tree size; in the random instances we observed only a moderate improvement (approx. 2% on average) in the number of search tree nodes.

Full experimental analysis will, of course, require an affix representation implementation of Regular.

VII. Future Work

Immediate work is focused on extending the propagators to achieve PS-consistency, and on improving the efficiency of the initial prototype. We also plan to investigate branching heuristics, as our intuition is that the early stages of search must rely heavily on intelligent branching over string lengths. A CP angle is promising here, as for fixed-length strings (implemented as arrays of scalar decision variables), CP solvers have already been shown [25] to outperform systematically Hampi [9], Kaluza [15], and Sushi [6] by orders of magnitude, on their own benchmarks. A promising direction of research is alternative affix representations, such as using regular languages to state the affix languages, allowing for stronger domains than the array-based representation we describe here. Supplementing the domain with a data structure such as layered graphs [11], or MDD constraint stores [26], would allow propagation directly on the data structure, similar to the approach for unbounded regular domains in [5].

Acknowledgements: Work supported by grant 2009-4384 of the Swedish Research Council (VR). The authors wish to thank the anonymous reviewers, of this and a prior version, for their valuable comments and suggestions; J.-N. Monette for assistance with the indexical compiler; and F. Hassani Bijarbooneh and C. Schulte for Gecode assistance.

References

[1] M. Emmi, R. Majumdar, and K. Sen, "Dynamic test input generation for database applications," in *Software Testing and Analysis (ISSTA 2007)*, D. S. Rosenblum and S. G. Elbaum, Eds., ACM, pp. 151–162.

[2] N. Bjørner, N. Tillmann, and A. Voronkov, "Path feasibility analysis for string-manipulating programs," in *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2009)*, S. Kowalewski and A. Philippou, Eds., ser. LNCS, vol. 5505, Springer, pp. 307–321.

[3] G. Gange, J. A. Navas, P. J. Stuckey, H. Søndergaard, and P. Schachte, "Unbounded model-checking with interpolation for regular language constraints," in *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2013)*, N. Piterman and S. A. Smolka, Eds., ser. LNCS, vol. 7795, Springer, pp. 277–291.

[4] P. Bisht, T. Hinrichs, N. Skrupsky, and V. N. Venkatakrishnan, "WAPTEC: whitebox analysis of web applications for parameter tampering exploit construction," in *Computer and Communications Security (CCS 2011)*, Y. Chen, G. Danezis, and V. Shmatikov, Eds., ACM, pp. 575–586.

[5] K. Golden and W. Pang, "Constraint reasoning over strings," in *CP 2003*, F. Rossi, Ed., ser. LNCS, vol. 2833, Springer, pp. 377–391.

[6] X. Fu, M. C. Powell, M. Bantegui, and C.-C. Li, "Simple linear string constraints," *Formal Aspects of Computing*, 2012, Sushi is available from http://people.hofstra.edu/Xiang_Fu/XiangFu/projects/SAFELI/SUSHI.php.

[7] P. Hooimeijer and W. Weimer, "A decision procedure for subset constraints over regular languages," in *Programming Language Design and Implementation (PLDI 2009)*, M. Hind and A. Diwan, Eds., ACM, pp. 188–198.

[8] ——, "StrSolve: solving string constraints lazily," *Automated Software Engineering*, vol. 19, no. 4, pp. 531–559, 2012.

[9] A. Kieżun, V. Ganesh, P. J. Guo, P. Hooimeijer, and M. D. Ernst, "HAMPI: a solver for string constraints," in *Inter. Symp. on Software Testing and Analysis (ISSTA 2009)*, G. Rothermel and L. K. Dillon, Eds., Hampi is available from http://people.csail.mit.edu/akiezun/hampi/, ACM, pp. 105–116.

[10] L. D. Michel and P. Van Hentenryck, "Constraint satisfaction over bit-vectors," in *CP 2012*, M. Milano, Ed., ser. LNCS, vol. 7514, Springer, pp. 527–543.

[11] G. Pesant, "A regular language membership constraint for finite sequences of variables," in *CP 2004*, M. Wallace, Ed., ser. LNCS, vol. 3258, Springer, pp. 482–495.

[12] N. Beldiceanu, M. Carlsson, and T. Petit, "Deriving filtering algorithms from constraint checkers," in *CP 2004*, M. Wallace, Ed., ser. LNCS, vol. 3258, Springer, pp. 107–122.

[13] M. Sellmann, "The theory of grammar constraints," in *CP 2006*, F. Benhamou, Ed., ser. LNCS, vol. 4204, Springer, pp. 530–544.

[14] C.-G. Quimper and T. Walsh, "Global grammar constraints," in *CP 2006*, F. Benhamou, Ed., ser. LNCS, vol. 4204, Springer, pp. 751–755.

[15] P. Saxena, D. Akhawe, S. Hanna, F. Mao, S. McCamant, and D. Song, "A symbolic execution framework for javascript," in *Security and Privacy (S&P 2010)*, Kaluza is available from http://webblaze.cs.berkeley.edu/2010/kaluza/, IEEE Computer Society, pp. 513–528.

[16] F. Yu, T. Bultan, and O. H. Ibarra, "Symbolic string verification: combining string analysis and size analysis," in *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2009)*, S. Kowalewski and A. Philippou, Eds., ser. LNCS, vol. 5505, Springer, pp. 322–336.

[17] I. Ghosh, N. Shafiei, G. Li, and W.-F. Chiang, "JST: an automatic test generation tool for industrial java applications with strings," in *Inter. Conf. on Software Engineering (ICSE '13)*, D. Notkin, B. H. C. Cheng, and K. Pohl, Eds., IEEE / ACM, pp. 992–1001.

[18] M. J. Maher, "Open constraints in a boundable world," in *CPAIOR 2009*, W. J. van Hoeve and J. N. Hooker, Eds., ser. LNCS, vol. 5547, Springer, pp. 163–177.

[19] E. C. Freuder and A. K. Mackworth, "Constraint satisfaction: an emerging paradigm," in *Handbook of Constraint Programming*, F. Rossi, P. van Beek, and T. Walsh, Eds., Amsterdam: Elsevier, 2006, ch. 2, pp. 13–27.

[20] W.-J. van Hoeve and I. Katriel, "Global constraints," in *Handbook of Constraint Programming*, F. Rossi, P. van Beek, and T. Walsh, Eds., Amsterdam: Elsevier, 2006, ch. 6, pp. 169–208.

[21] R. Barták, "Dynamic global constraints in backtracking based environments," *Annals of Operations Research*, vol. 118, no. 1, pp. 101–119, 2003.

[22] J.-N. Monette, P. Flener, and J. Pearson, "Towards solver-independent propagators," in *CP 2012*, M. Milano, Ed., ser. LNCS, vol. 7514, Springer.

[23] P. Van Hentenryck, V. A. Saraswat, and Y. Deville, "Design, implementation, and evaluation of the constraint language cc(FD)," in *Constraint Programming, Basics and Trends*, A. Podelski, Ed., ser. LNCS, vol. 910, Springer, pp. 293–316.

[24] (2013). Gecode, [Online]. Available: www.gecode.org.

[25] J. He, P. Flener, and J. Pearson, "Solving string constraints: The case for constraint programming," in *CP 2013*, C. Schulte, Ed., ser. LNCS, Springer.

[26] S. Hoda, W.-J. van Hoeve, and J. N. Hooker, "A systematic approach to MDD-based constraint programming," in *CP 2010*, D. Cohen, Ed., ser. LNCS, vol. 6308, Springer, pp. 266–280.