Reasoning about Strings in Databases

Gösta Grahne Matti Nykänen
Esko Ukkonen
Department of Computer Science
P.O. Box 26, FIN–00014 University of Helsinki, Finland
{Gosta.Grahne,Matti.Nykanen,Esko.Ukkonen}@cs.helsinki.fi

Abstract

In order to enable the database programmer to reason about relations over strings of arbitrary length, we introduce Alignment Calculus, a modal extension of relational calculus. In addition to relations, a state in the model consists of a two-dimensional array where the strings are aligned on top of each other. The basic modality in the language (a transpose, or "slide") rearranges this alignment, and more complex formulae can be formed using a syntax reminiscent of regular expressions, in addition to the usual connectives and quantifiers. It turns out that the computational counterpart of the string-based portion of the logic is the class of multitape two-way finite state automata, which are devices particularly well suited for the implementation of string matching. A computational counterpart of the full logic is obtained from relational algebra by performing selection with these devices. Safety of formulae in Alignment Calculus implies that new strings generated from old ones have to be of bounded length. While an undecidable property in general, this boundedness is decidable for an important subclass of formulae. As far as expressive power is concerned, Alignment Calculus includes previous proposals for querying string databases, and gives full Turing computability. The language can be restricted to define exactly the regular sets and sets in each level of the polynomial-time hierarchy above **P**.

1 Introduction

In this paper we focus on the problem of strings in databases. Our primary source of motivation is the storage and qualitative processing of genetic information, represented as sequences of symbols. For instance, the theory of gene regulation explores the combinatorial or grammatical structure inherent in genetic sequences, as opposed to their statistical properties. This structure can be quite complex, since gene regulation involves non-context-free dependencies between different parts of a string [1]. These dependencies should be explicitly expressible, as they constitute knowledge about the family of strings that the current database represents. Representing this kind of knowledge is in turn becoming increasingly important when operational models of genetic phenomena [14] are being sought.

We extend the relational model to include finite strings over some finite alphabet Σ as primary objects of information. A relation of arity k is in our model then a finite subset of the k-fold Cartesian product of Σ^*, the set of all finite strings in alphabet Σ, with itself. In other words, each position in a tuple of a relation contains a finite string of arbitrary length instead of just a single atomic value.

It is quite clear that a database language operating on string relations should have a pattern-matching ability in order to be able to express queries of the form "list all tuples of relation r where the second component is of the form $(\mathtt{gc} + \mathtt{a})^*$". However, in applications such as the aforementioned gene regulation, the language needs to have expressive power beyond regular sets; such suggestions include [10, 21].

In addition to data extraction features, the string language needs also data restructuring constructs. For example, given two unary relations, one might want to concatenate each string from one of the relations with a string from the other relation, as opposed to merely taking the Cartesian product of the two relations. Or as a more complex example, one might want to shuffle two relations (we shall see in Section 2 how to express such a transformation).

How should one build a database language having such features? From the literature we find the following types of proposals. On the one hand, one can add, say to relational algebra, a selection predicate [10, 16, 21] for testing membership in a set specified by, for instance, a regular expression. This partially solves the data extraction problem but does not support restructuring operations very well. On the other hand, one can add a restructuring operator, based for example on transducers [7, 15, 26]. This gives good data restructuring abilities, but the resulting data extraction features are often rather implicit.

Then there is the *declarative* approach. The idea is to design a language for expressing properties of strings. One such proposal can be found in [18], which essentially suggests using the modalities of *temporal logic* for this purpose. Each successive position in a string is seen to be the timewise "next" instance of that string. The temporal modalities lend themselves naturally to reasoning about strings. But as shown by Wolper [29], using only the modalities *next* and *until*, the language cannot express that a property holds in every even position of the string. Using Wolper's *extended* temporal logic would be a step in the right direction. Still, extended temporal logic cannot express, for instance, the two-place predicate of equalness between strings, not to mention predicates such as saying that one string is a manifold of the other.

We therefore define a logic in which we can express both properties of individual strings

```
 a|b|c
ab|b|
 c|a|cd
```

Figure 1: An Alignment of three strings

and properties relating strings to each other. The purely relational part of the logic is handled by relational calculus. The string part of the logic is *state based* (as opposed to second or higher order) like for example temporal and dynamic logic. A state of the strings is a structure where the strings are aligned on top of each other in a certain way, and a state change is obtained by sliding some of the strings.

Our logic is defined in Section 2. In Section 3 we give an algebraic language and show that it has the same expressive power as the logic. The novel construct in the algebraic language is a selection operator based on certain finite state acceptors. In Section 4 we study the safety (evaluability within a finite domain) of formulae in the logic. The main tool for determining safety is the concept of limitation in the formulae describing the string properties: does the boundedness of some variables imply boundedness of some other variables? We show that while the limitation problem is undecidable in general, it remains decidable for an important subclass of formulae. In Section 5 we use the same subclass of formulae to obtain a characterization of the polynomial-time hierarchy. The components of our logic corresponding to the aforementioned acceptors are called *string formulae*. The string formulae as such have the same expressive power as the multitape two-way automata. This alone is enough for string matching [4], among other things. The string formulae together with one projection operator yield the power of full Turing computability.

An extended abstract of this work has been presented in [9].

2 Alignment Calculus

This section introduces formally *Alignment Calculus*, our logic for expressing string properties, and its interpretation as a string database query language.

In the world of strings we distinguish as state an *alignment* of the strings. Fig. 1 illustrates an alignment of the three strings `abc`, `abb` and `cacd`. Properties of an alignment are expressed with respect to the vertical window. For instance, in the alignment of Fig. 1 the following proposition is true: "window position of the topmost string equals `a` or the window position of the middle string is different from `c`". On the other hand, the following proposition is false in that alignment: "the window position of the middle and the bottom string of the alignment are equal".

Let Σ be henceforth a fixed finite alphabet with at least two characters. Formally an alignment is then a partial function $A : \mathbb{N} \times \mathbb{Z} \to \Sigma$ such that for all $i \in \mathbb{N}$ there exists a contiguous interval $K_i^A \in \mathbb{Z}$ for which

$$A(i,j) = \begin{cases} a & \text{for some } a \in \Sigma \text{ if } j \in K_i^A \\ \text{undefined} & \text{otherwise.} \end{cases}$$

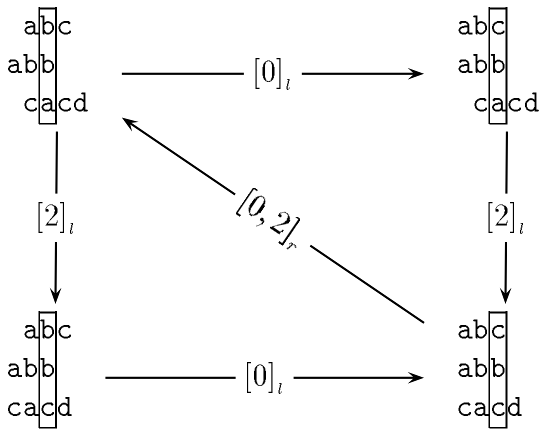


Figure 2: Transposing alignments

We further require either that $K_i^A = \emptyset$ (denoting the empty string ε) or that $K_i^A \cap \{-1, 0, 1\} \neq \emptyset$ so that the *window column* 0 is always at either end or in the middle of the defined area. Thus, if the alignment in Fig. 1 represents the three first rows of A, then we have for instance $A(2, -1) = \mathtt{c}$, $A(2, 0) = \mathtt{a}$, $A(2, 1) = \mathtt{c}$, $A(2, 2) = \mathtt{d}$, and $A(2, j)$ is undefined elsewhere.

Alignments are connected to each other through state transitions called transposes. A transpose says that certain rows in the alignment should be shifted one position to the left or to the right. A *left transpose* is a construct of the form $[i_1, \ldots, i_k]_l$, where k and each i_j are in $\mathbb{N}$, and it represents a function on the set of all alignments. This function is defined as

$$[i_1, \ldots, i_k]_l A(i, j) = \begin{cases} A(i, j+1) & \text{if } i \in \{i_1, \ldots, i_k\} \text{ and } K_i^A \cap \{0, 1\} \neq \emptyset \\ A(i, j) & \text{otherwise.} \end{cases}$$

That is, each row in $i_1, \ldots, i_k$ is shifted one character to the left with respect to the window column unless the window column is already at the right end of the row in which case the row does not move.

The *right transposes* are denoted $[i_1, \ldots, i_k]_r$ and are defined similarly to the left ones, except that the rows mentioned in the expression are transposed one position to the right, instead of to the left (e.g. $[3, 5]_r A(3, j) = A(3, j-1)$, and $[3, 5]_r A(5, j) = A(5, j-1)$, and all the other rows remain unchanged). In Fig. 2 we show some transposes of the alignment in Fig. 1.

The propositions on the window of a particular alignment are expressed through *window formulae* which are Boolean combinations of *atomic* window formulae of the form $x = \perp$, $x = a$, or $x = y$, where x and y are members of a countably infinite set V of variables, the symbol $\perp$ denotes "undefined", and a is a symbol in Σ.

The variables range over row numbers in alignments, and they are fixed through an *assignment* which is any injection $\theta : V \to \mathbb{N}$. Injectivity ensures that no two distinct variables can refer to the same row of the alignment at the same time. Given a particular alignment A and a window formula ϕ, the definition below tells us when *A satisfies ϕ with assignment θ*, in symbolic notation $A \models \phi\theta$.

1. $A \models (x = \bot)\theta$ if and only if $A(\theta x, 0)$ is undefined.

2. $A \models (x = a)\theta$ if and only if $A(\theta x, 0) = a$ where $a \in \Sigma$.

3. $A \models (x = y)\theta$ if and only if $A(\theta x, 0) = A(\theta y, 0)$.

4. $A \models (\phi \wedge \psi)\theta$ if and only if $A \models \phi\theta$ and $A \models \psi\theta$.

5. $A \models (\neg\phi)\theta$ if and only if $A \models \phi\theta$ does not hold, often denoted $A \not\models \phi\theta$.

For instance, if the three first rows of A are as in Fig. 1, and if θ maps x to 0, y to 1, and z to 2, then we have $A \models ((x = \mathtt{a}) \vee \neg(y = \mathtt{c}))\theta$, and $A \not\models (x = z)\theta$.

We also use the shorthands $x \neq y$ for $\neg(x = y)$, $\phi \vee \psi$ for $\neg(\neg\phi \wedge \neg\psi)$, and $x_1 = x_2 = x_3 = \ldots = x_m$ for $x_1 = x_2 \wedge x_2 = x_3 \wedge \ldots \wedge x_{m-1} = x_m$. Furthermore, we allow redundant parenthesization to clarify the structure of the formulae.

Transposes are expressed as indexed modalities in the language, and these modalities can be applied to window formulae. Formally, if ϕ is a window formula, and $\{x_1, \ldots, x_k\} \subset V$, then $[x_1, \ldots, x_k]_l\phi$ and $[x_1, \ldots, x_k]_r\phi$ are called *atomic string formulae*. For example, $[x]_l(x = \mathtt{c} \wedge y = \mathtt{b})$, $[x]_l(x = \mathtt{c})$, and $[x, z]_r(z = \mathtt{a} \vee y = \mathtt{b})$ are all atomic string formulae. The definition of satisfaction for atomic string formulae in alignment A with assignment θ is as follows.

6. $A \models [x_1, \ldots, x_k]_l\phi\theta$ if and only if $[\theta x_1, \ldots, \theta x_k]_l A \models \phi\theta$.

7. $A \models [x_1, \ldots, x_k]_r\phi\theta$ if and only if $[\theta x_1, \ldots, \theta x_k]_r A \models \phi\theta$.

As an illustration, let A be the top-left alignment, and A' the bottom right alignment in Fig. 2, and let θ map x to 0, y to 1, and z to 2. Then we have $A \models ([x]_l(x = \mathtt{c} \wedge y = \mathtt{b}))\theta$, $A \not\models ([x]_l(x = \mathtt{c}))\theta$, and $A' \models ([x, z]_r(z = \mathtt{a} \vee y = \mathtt{b}))\theta$.

We can compose atomic string formulae by concatenating them. We define a *formula word* to be an expression of the form $\phi_1\phi_2\ldots\phi_k$, where each ϕ_j is an atomic string formula. For instance, $[x, z]_r(z = \mathtt{a} \vee y = \mathtt{b})[x]_l(x = \mathtt{c} \wedge y = \mathtt{b})$ and $[x, z]_r(z = \mathtt{a} \vee y = \mathtt{b})[z]_l(x = \mathtt{c})$ are formula words. The unique formula word containing no atomic string formulae is denoted λ; it is vacuously true in every alignment under every assignment.

Before we proceed we shall introduce some notation for convenience: Atomic string formulae, like $[x, z]_r(z = \mathtt{a} \vee y = \mathtt{b})$ will sometimes be denoted $\tau\alpha$, where in formula above τ stands for $[x, z]_r$, and α stands for $(z = \mathtt{a} \vee y = \mathtt{b})$. If θ is an assignment and $\tau\alpha$ as above, then $\tau\theta$ denotes $[\theta x, \theta z]_r$, and $\alpha\theta$ denotes $(z = \mathtt{a} \vee y = \mathtt{b})\theta$.

Now we can denote formula words generally by $\tau_1\alpha_1\tau_2\alpha_2\ldots\tau_k\alpha_k$ and define

8. $A \models (\tau_1\alpha_1\tau_2\alpha_2\ldots\tau_k\alpha_k)\theta$ if and only if both $\tau_1\theta A \models \alpha_1\theta$ and $\tau_1\theta A \models (\tau_2\alpha_2\ldots\tau_k\alpha_k)\theta$ hold.

Once again, let A' be the bottom right alignment in Fig. 2, and let θ map x to 0, y to 1, and z to 2. Then we have the following: $A' \models ([x, z]_r(z = \mathtt{a} \vee y = \mathtt{b})[x]_l(x = \mathtt{c} \wedge y = \mathtt{b}))\theta$, and $A' \not\models ([x, z]_r(z = \mathtt{a} \vee y = \mathtt{b})[z]_l(x = \mathtt{c}))\theta$.

Sets of formula words can be represented by expressions called *string formulae*: all atomic string formulae are string formulae, and if ϕ and ψ are string formulae, then so are $\phi.\psi$, $\phi + \psi$, (ϕ), and $(\phi)^*$. We also use the shorthand ϕ^+ for $\phi.\phi^*$, and ϕ^{n+1} for $\phi.\phi^n$, where $\phi^0 = \lambda$.

A string formula can be seen as a regular expression over the alphabet of atomic string formulae. The set of formula words it thus defines is denoted $L(\phi)$. For example, if ϕ is the string formula

$$([x,z]_r(z = \mathtt{a} \vee y = \mathtt{b}).([x]_l(x = \mathtt{c} \wedge y = \mathtt{b}) + [z]_l(x = \mathtt{c}))$$

then

$$\begin{aligned} L(\phi) = \{&[x,z]_r(z = \mathtt{a} \vee y = \mathtt{b})[x]_l(x = \mathtt{c} \wedge y = \mathtt{b}), \\ &[x,z]_r(z = \mathtt{a} \vee y = \mathtt{b})[z]_l(x = \mathtt{c})\}. \end{aligned}$$

We now define

9. $A \models \phi\theta$ for a string formula ϕ if and only if $A \models (\tau_1\alpha_1\tau_2\alpha_2 \ldots \tau_k\alpha_k)\theta$ holds for some formula word $\tau_1\alpha_1\tau_2\alpha_2 \ldots \tau_k\alpha_k \in L(\phi)$.

If ϕ is the string formula given in the preceding example, A' is the bottom right alignment in Fig. 2, and θ maps x to 0, y to 1, and z to 2, then the reader can easily verify that indeed $A' \models \phi\theta$. If the fourth row in alignment A' were `abababa`, with the first `a` positioned in the window, and θ mapped variable u to 3, then we would have $A' \models ([u]_l(u = \mathtt{b}).[u]_l(u = \mathtt{a}))^*\theta$, and $A' \not\models ([u]_l(u = \mathtt{a}).[u]_l(u = \mathtt{b}))^*\theta$.

A *database db* is any function mapping each relation symbol R_i with arity $a(R_i)$ into a finite subset of $(\Sigma^*)^{a(R_i)}$. The collection of all databases is denoted DB. Thus we can say for instance that the tuple whose first component is the string represented by the third row in A, and whose second component is the string represented by the sixth row, is a member of binary relation $db(R_9)$. In symbolic notation this amounts to writing $\langle \sigma_A(2), \sigma_A(5)\rangle \in db(R_9)$, where $\sigma_A(i)$ denotes *the string represented by row i in the alignment A*. This is formally defined as

$$A(i, \min K_i^A)A(i, \min K_i^A + 1)A(i, \min K_i^A + 2) \ldots A(i, \max K_i^A) \in \Sigma^*.$$

For example, if A is the top left alignment in Fig. 2, then

$$\sigma_A(2) = A(2,-1)A(2,0)A(2,1)A(2,2) = \mathtt{cacd}.$$

We omit mentioning the alignment whenever it is clear from the context.

We then evaluate the truth of an *atomic relational formula* $R_i(x_1, \ldots, x_k)$ under assignment θ with respect to a pair consisting of an alignment and a database. Formally we have

10. $\langle A, db\rangle \models R_i(x_1, \ldots, x_k)\theta$ if and only if $\langle \sigma_A(\theta x_1), \ldots, \sigma_A(\theta x_k)\rangle \in db(R_i)$.

To complete the definition of our language we take as atomic formulae the set of all string formulae and atomic relational formulae, and close it under $\wedge$, $\neg$, and $\exists$ with the following semantics:

11. $\langle A, db\rangle \models (\phi \wedge \psi)\theta$ if and only if $\langle A, db\rangle \models \phi\theta$ and $\langle A, db\rangle \models \psi\theta$.

12. $\langle A, db\rangle \models (\neg\phi)\theta$ if and only if $\langle A, db\rangle \not\models \phi\theta$.

13. $\langle A, db\rangle \models (\exists x.\phi)\theta$ if and only if there exists an $i \in \mathbb{N}$ such that $\langle A, db\rangle \models \phi\theta[x = i]$. Here $\theta[x = i]$ denotes an assignment which agrees with θ everywhere except that x is now mapped to i instead.

We also use the shorthands $\forall x.\phi$ for $\neg\exists x.\neg\phi$, $\phi \vee \psi$ for $\neg(\neg\phi \wedge \neg\psi)$, and $\phi \to \psi$ for $(\neg\phi) \vee \psi$.

The language defined above has a *layered* structure: A transpose cannot be applied to a formula containing quantifiers, relation symbols or other transposes. Conversely, character comparisons can only appear within an enclosing transpose. The inner level of string formulae only deals with aligning given sequences to each other via character comparisons, the outer level of relational calculus only deals with sequences as indivisible entities. This is in contrast with conventional First-Order Linear Temporal Logic [2, Chapter 3.3] where modalities, connectives and quantifiers are allowed to mix freely.

Note that the truth definitions 1–9 do not depend on a particular database. Therefore they extend to tuples $\langle A, db\rangle$ as such; for example in definition 2, $\langle A, db\rangle \models (x = a)\theta$ if and only if $A(\theta x, 0) = a$. Note also that in the truth definitions 6 and 7 involving a transpose the relational part and the assignment remain intact; for example in definition 6 $\langle A, db\rangle \models [x_1, \ldots, x_k]_l \phi\theta$ if and only if $\langle [\tau x_1, \ldots, \tau x_k]_l A, db\rangle \models \phi\theta$.

When formulating a query on a database, the programmer is to assume that all the strings being reasoned about are in a starting position corresponding to an *initial alignment:* an alignment where the leftmost symbol in each string is placed one position to the right of the window, at column 1. In other words, $\min K_i^A = 1$ for each row i. Initial alignments are denoted A_0. We thus have for instance $A_0 \models (x = \perp)\theta$, for all assignments θ and initial alignments A_0. This also fixes left transposes as the "forward" and right transposes as the "reverse" direction of sequence processing. Accordingly we call a variable in a string formula *unidirectional* if it only appears in left transposes and *bidirectional* if it appears in right transposes as well.

We have now introduced all the apparatus necessary to define the meaning of queries in string databases. A query in our model is an expression $x_{i_1}, \ldots, x_{i_k} | \phi$, where ϕ is a formula whose free variables are $x_{i_1}, \ldots, x_{i_k}$. (We assume for simplicity the free variables of a query to be implicitly ordered by ascending index; that is, $i_1 < \ldots < i_k$.) The answer to such a query when posed to a database db is

$$[\![\phi]\!]_{db} = \bigcup_{A_0, \theta} \{\langle \sigma_{A_0}(\theta x_{i_1}), \ldots, \sigma_{A_0}(\theta x_{i_k})\rangle : \langle A_0, db\rangle \models \phi\theta\}. \tag{1}$$

Here the union is over all *full interpretations:* pairs consisting of an initial alignment A_0 and an assignment θ such that for every string $u \in \Sigma^*$ there exist infinitely many i outside the range of θ with $\sigma_{A_0}(i) = u$. This extra fullness condition ensures that the assignments in truth definition 13 can always be formed, and hence that the quantifiers range freely over Σ^*. The formulae ϕ containing no atomic relational formulae constitute *pure* Alignment Calculus. As noted above, their truth does not depend on the current database, and hence we can denote the corresponding answers with $[\![\phi]\!]$ instead.

A query ϕ then asks to list those tuples of strings, such that there is a way of initially aligning the strings, possibly together with other (quantified) strings freely selected from Σ^*, then substituting the strings for the free variables and having the formula become true with respect to the initial alignment and the database.

Given an initial alignment A_0 and a constant $\ell \in \mathbb{N}$, the *truncation* A_0^ℓ *of* A_0 *to length* ℓ is the initial alignment such that $A_0^\ell(i, j) = A_0(i, j)$ when $j \leq \ell$ and undefined otherwise. In other words, each $\sigma_{A_0^\ell}(i)$ is the string containing the ℓ first characters of string $\sigma_{A_0}(i)$.

Similarly the *truncation* $[\![\phi]\!]_{db}^{\ell}$ *of* $[\![\phi]\!]_{db}$ *to length* ℓ is defined following Eq. (1) as

$$[\![\phi]\!]_{db}^{\ell} = \bigcup_{A_0,\theta} \left\{ \left\langle \sigma_{A_0^\ell}(\theta x_{i_1}), \ldots, \sigma_{A_0^\ell}(\theta x_{i_k}) \right\rangle : \left\langle A_0^\ell, db \right\rangle \models \phi\theta \right\}$$

over the full interpretations as in Eq. (1) above. Note in particular that here the quantifiers range freely over $\Sigma^{\leq \ell}$ when determining $[\![\phi]\!]_{db}^{\ell}$.

In the sequel we shall leave out the union over the full interpretations when denoting answer sets. This existential quantification will be tacitly assumed.

Examples. Let R_1 be a binary and R_2 a unary string relation symbol. The following queries will serve as an illustration of our model. We freely omit parenthesis symbols and concatenation dots whenever we feel that the structure is clear without them. The symbol $\top$ denotes a tautological window formula, e.g. $x = x$.

1. List the second component of all tuples in R_1 where the first component is `abc`:

$$x|\exists y \colon R_1(y, x) \wedge ([y]_l y = \mathtt{a})([y]_l y = \mathtt{b})([y]_l y = \mathtt{c})([y]_l y = \bot)$$

2. List all tuples in R_1 where the first component equals the second component:

$$x, y|R_1(x, y) \wedge ([x, y]_l x = y)^*.([x, y]_l x = y = \bot)$$

   The second conjunct is a string formula verifying that x and y contain the same string. In what follows it will be denoted $x =_s y$.

3. List all tuples of R_2 that are a concatenation of the two components in a tuple in relation R_1:

$$\begin{aligned} x|\exists y, z \colon & R_1(y, z) \wedge R_2(x) \wedge \\ & ([x, y]_l x = y)^*([x, z]_l x = z)^*([x, y, z]_l x = y = z = \bot) \end{aligned}$$

4. List all tuples of R_1 where the first component is a manifold of the second (the manifolds of string v being the strings $vvv \ldots v$):

$$\begin{aligned} x, y|& R_1(x, y) \wedge \\ & (([x, y]_l x = y)^*([y]_l y = \bot)([y]_r y \neq \bot)^*([y]_r y = \bot))^*. \\ & ([x, y]_l x = y)^*([x, y]_l x = y = \bot) \end{aligned}$$

   The string formula repeatedly checks that y is indeed a prefix of the remaining part of x until x is exhausted. In what follows it will be denoted $x \in_s^* y$.

5. List all tuples in R_2 that are a shuffle of the two components of a tuple in R_1:

$$\begin{aligned} x|\exists y, z \colon & R_1(y, z) \wedge R_2(x) \wedge \\ & (([x, y]_l x = y) + ([x, z]_l x = z))^*([x, y, z]_l x = y = z = \bot) \end{aligned}$$

   (The shuffle, or interleaving, of strings u and v is the set of all strings of the form $u_1 v_1 u_2 v_2 \ldots u_k v_k$ where $u = u_1 u_2 \ldots u_k$, $v = v_1 v_2 \ldots v_k$, and each u_i and v_i can be of arbitrary length, including zero.)

6. List all tuples of R_1 where the second component is of the form $(\mathtt{gc}+\mathtt{a})^*$, as pointed out in Section 1:

$$x, y|R_1(x,y) \wedge (([y]_l y = \mathtt{g})([y]_l y = \mathtt{c})) + ([y]_l y = \mathtt{a}))^*.([y]_l y = \bot)$$

7. List all tuples of R_1 where the first component occurs in the second:

$$x, y|R_1(x,y) \wedge ([y]_l \top)^*([x,y]_l x = y)^*([x]_l x = \bot)$$

8. List all tuples of R_1 where the edit distance between the first and the second component is not larger than k:

$$\begin{aligned} x, y|R_1(x,y) \wedge & ([x,y]_l x = y)^* \\ & (([x,y]_l \top + [x]_l \top + [y]_l \top)([x,y]_l x = y)^*)^k \\ & ([x,y]_l x = y = \bot) \end{aligned}$$

(The edit distance between strings u and v is the minimum number of steps required to transform u to v. Each step can consist of replacing one symbol by another, or of inserting or deleting a symbol, see for example [20].)

The formula requires that x and y must match character by character, except that in at most k places the characters need not match. A replacement can be allowed by relaxing the window formula $x = y$ to $\top$. An insertion into x is taken into account by transposing only x, and a deletion from x by transposing only y.

Edit distance is an example of *a measure of similarity* useful in sequence database queries [13, page 36][21, Section 3]. Compared with the very general domain-independent framework studied in [13], Alignment Calculus lacks essentially a means to express numerical *degrees* of similarity, as witnessed by the fact that k is a constant and not a parameter in the query above. Much of this functionality can, however, be captured within the current semantics by using strings for counting; for example,

$$\begin{aligned} x, y, z|R_1(x,y) \wedge & ([x,y]_l x = y)^* \\ & (([x,y,z]_l(z = \mathtt{a}) + [x,z]_l(z = \mathtt{a}) + [y,z]_l(z = \mathtt{a}))([x,y]_l x = y)^*)^* \\ & ([x,y,z]_l x = y = z = \bot) \end{aligned}$$

lists all the tuples $\left\langle u, v, \mathtt{a}^k \right\rangle$, where $\langle u, v\rangle \in db(R_1)$, and the edit distance of u and v is at most $k \leq |u| + |v|$.

9. List all tuples of R_2 that are of the form $\mathtt{a}X\mathtt{b}X\mathtt{a}$, where $X \in \Sigma^*$. (The string formula $x =_s y$ for string equality was defined in Example 2 above.)

$$\begin{aligned} x|\exists y, z{:}(y =_s z) \wedge R_2(x) \wedge & \\ & ([x]_l x = \mathtt{a}).([x,y]_l x = y)^*.([x,y]_l x = \mathtt{b} \wedge y = \bot). \\ & ([x,z]_l x = z)^*.([x,z]_l x = \mathtt{a} \wedge z = \bot).([x]_l x = \bot) \end{aligned}$$

The formula states the existence of a string y such that x is of the form $\mathtt{a}y\mathtt{b}y\mathtt{a}$. Instead of explicitly transposing y to the right the formula states the existence of an identical copy z for verifying that the first and second occurrences of y are indeed equal. This shows how the logical "and" operation can be used to reset the strings into their initial alignment.

10. List all tuples of R_2 that are in the language consisting of strings containing an equal number of a's and b's and only those symbols:

$$
\begin{aligned}
x|\exists y, z\colon R_2(x) \wedge (([x,y]_l x = \mathtt{a} \wedge y \neq \bot) + ([x,z]_l x = \mathtt{b} \wedge z \neq \bot))^*.\\
([x,y,z]_l x = y = z = \bot)\\
\wedge([y,z]_l y \neq \bot \wedge z \neq \bot)^*([y,z]_l y = z = \bot)
\end{aligned}
$$

The formula says that each occurrence of the character a in x must match some position in a string y. Likewise, each occurrence of the character b must match a position in a string z. Furthermore, it says that strings y and z must be exhausted simultaneously, thus requiring them to be of equal length.

11. List all tuples of R_2 that are in the language $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \colon n \in \mathbb{N}\}$:

$$
\begin{aligned}
x|\exists y{:}R_2(x)\wedge\\
([x,y]_l x = \mathtt{a} \wedge y \neq \bot)^*.([y]_l y = \bot).\\
([x]_l \top.[y]_r x = \mathtt{b} \wedge y \neq \bot)^*.([y]_r y = \bot).\\
([x,y]_l x = \mathtt{c} \wedge y \neq \bot)^*.([x,y]_l x = y = \bot)
\end{aligned}
$$

Here we require that x is of the form $\mathtt{a}^*\mathtt{b}^*\mathtt{c}^*$ and that there is a "counter string" y, such that each of the three portions of x can be put into a one-to-one correspondence with y. The third line of the formula also shows how simultaneous left and right transpositions are expressible in string formulae.

12. List all tuples of R_2 that are of the form $(\mathtt{a} + \mathtt{b})^*$, and whose second half is a translation of the first half obtained by replacing each a by b, and each b by a. This type of problem occurs for example in [1].

$$
\begin{aligned}
x|\exists y, z\colon R_2(x)\wedge([x,y]_l x = y)^*.([y]_l y = \bot).([x,z]_l x = z)^*.([z]_l z = \bot)\\
\wedge([y,z]_l (y = \mathtt{a} \wedge z = \mathtt{b}) \vee (y = \mathtt{b} \wedge z = \mathtt{a}))^*.([y,z]_l y = z = \bot)
\end{aligned}
$$

The formula states that x is of the form yz where z is the translation of y.

Note that in all the examples above there is at most one bidirectional variable in any string formula. We shall later see that such *right-restricted* formulae form an important subclass of Alignment Calculus.

3 Alignment Calculus and Computation

Since the queries are expressed in Alignment Calculus in a declarative fashion, we need a procedural interpretation of the formulae in order to be able to compute the answers. Our procedural language is an extension of relational algebra, the main addition being a selection operator based on certain finite state devices. These finite state devices correspond to the string formulae in the query.

Informally, a *k-tape finite state acceptor (k-FSA)* is a "nondeterministic k-tape two-way finite state automaton with endmarkers". Formally, a k-FSA $\mathcal{A} = \langle Q, s, F, T\rangle$ is a system where Q is a finite set of *states*, $s \in Q$ is a distinguished *start* state, $F \subseteq Q$ is a set of *final* states and $T \subseteq (Q \times (\Sigma \cup \{¢, \$\})^k) \times (Q \times -1, 0, +1)^k)$ is *a transition relation.*

The elements $\langle\langle p, c_1, \ldots, c_k\rangle, \langle q, d_1, \ldots, d_k\rangle\rangle \in T$, which are called *transitions*, have the additional restriction that if $c_i = ¢$ then $d_i \neq -1$ and if $c_i = \$$ then $d_i \neq +1$ for every tape $i = 1, \ldots, k$. This restriction means that the head cannot leave the tape area marked by the left and right *endmarkers* ¢ and \$.

The *configurations of $\mathcal{A}$ on input* $W = \langle w_1, \ldots, w_k\rangle$, where each w_i is a string from Σ^*, are $\langle p, n_1, \ldots, n_k\rangle$ such that $p \in Q$ and $0 \leq n_i \leq |w_i| + 1$ for every tape $i = 1, \ldots, k$. (Here $|w_i|$ denotes the *length* of the string w_i, measured as usual by the number of character occurrences in it.) A configuration $\langle q, n_1 + d_1, \ldots, n_k + d_k\rangle$ is a *next* configuration of $\langle p, n_1, \ldots, n_k\rangle$ on W if and only if $\langle\langle p, w_1[n_1], \ldots, w_k[n_k]\rangle, \langle q, d_1, \ldots, d_k\rangle\rangle \in T$, where

$$a_1 \ldots a_m[j] = \begin{cases} ¢ & j = 0 \\ a_j & 1 \leq j \leq m \\ \$ & j = m+1 \end{cases}$$

denotes the j'th character on the tape containing the string $a_1 \ldots a_m \in \Sigma^*$. In addition, $w[j \ldots k]$ is a shorthand for $w[j]w[j+1]w[j+2]\ldots w[k]$.

A *computation of $\mathcal{A}$ on* W is any sequence $C_1C_2C_3\ldots$ of its configurations on W where C_{i+1} is a next configuration of C_i. A computation $\mathcal{C} = C_1C_2C_3\ldots$ *accepts* W if and only if its first configuration C_1 is the *initial* configuration $\langle s, 0, \ldots, 0\rangle$, it is finite, the state of its last configuration C_m belongs to F, and C_m has no next configuration on W. The *language* $L(\mathcal{A})$ *accepted by* $\mathcal{A}$ consists of the tuples W accepted by $\mathcal{A}$ – that is, those W for which a finite accepting computation exists.

We often view $\mathcal{A}$ as a graph with vertices Q and an edge labelled '$c_1\ d_1 \ldots c_k\ d_k$' from p to q if and only if T contains a transition $\langle\langle p, c_1, \ldots, c_k\rangle, \langle q, d_1, \ldots, d_k\rangle\rangle$. A computation can then be viewed as tracing a path in this graph. In this view it is also natural to define the *size* of $\mathcal{A}$, denoted $|\mathcal{A}|$, as the number $|Q|$ of transitions in it.

Analogously to variables in string formulae, tape i of $\mathcal{A}$ is said to be *bidirectional* if and only if T contains a transition where $d_i = -1$; otherwise it is *unidirectional*. The operational intuition is that bidirectional tapes can be scanned back and forth, while unidirectional tape squares can only be read once. A transition is *stationary* if and only if $d_1 = \ldots = d_k = 0$; that is, no tape movement occurs. These form the FSA counterpart of the ε-transitions in classical, one-tape automata theory.

A k-FSA can be modified to *disregard tape* l by replacing its every transition

$$\langle\langle p, \ldots, c_{l-1}, c_l, c_{l+1}, \ldots\rangle, \langle q, \ldots, d_{l-1}, d_l, d_{l+1}, \ldots\rangle\rangle$$

with

$$\langle\langle p, \ldots, c_{l-1}, ¢, c_{l+1}, \ldots\rangle, \langle q, \ldots, d_{l-1}, 0, d_{l+1}, \ldots\rangle\rangle.$$

That is, tape l is retained but never moved from its leftmost endmarker ¢.

We now have a means of computing the "bindings" making string formulae true in initial alignments.

Theorem 3.1 *For each string formula ϕ on variables $x_1, \ldots, x_k$ there is an effectively constructible k-FSA $\mathcal{A}_\phi$ such that $L(\mathcal{A}_\phi) = [\![\phi]\!]$.*

Moreover, $\mathcal{A}_\phi$ can be chosen to fulfill the following properties:

1. *Tape i is bidirectional only if variable x_i is.*

```
 ¢abc$
¢abb$
 ¢cacd$
```

Figure 3: The tapes corresponding to an Alignment.

2. *The start state s has no incoming transitions.*

3. *Either $\mathcal{A}_\phi$ consists of a single nonfinal start state or every transition is on some path from the start state into a unique final state f.*

4. *This final state f (if it exists) is different from the start state, has no outgoing transitions and its incoming transitions are exactly the stationary transitions of $\mathcal{A}_\phi$.*

5. *Disregarding the bidirectional tapes of $\mathcal{A}_\phi$, for each path P from s to f there exists a computation tracing exactly P.*

Proof: The intuition behind the theorem is the familiar equivalence of regular expressions and finite automata, with string formulae as expressions and FSAs as automata. The fundamental building blocks of string formulae and FSAs, atomic string formulae and transitions, are closely related: both specify a one-step movement and a Boolean test. The extra Properties (1) to (5) will become useful later; intuitively they state that we can even perform a kind of "data flow analysis" on the FSA yielding a lot of explicit information on its operation. For example, Property (5) states how to determine the unidirectional inputs accepted by the FSA.

First note that $A_0 \models \phi\theta$ if and only if $A'_0 \models \phi\theta'$, where $\sigma_{A_0}(\theta x_i) = \sigma_{A'_0}(\theta' x_i)$ for all $i = 1, \ldots, k$. Hence, without loss of generality we can restrict attention to the assignment θ where $x_i \mapsto i$ (that is, row i contains the string bound to x_i) and omit mentioning it explicitly.

Say that a configuration $\langle p, n_1, \ldots, n_k \rangle$ of $\mathcal{A}_\phi$ on input $\langle w_1, \ldots, w_k \rangle$ *corresponds to* the alignment A if and only if for every tape $i = 1, \ldots, k$ both $w\left[\left|K_i^A\right| + 1\right] = \$$ and $w\left[\min K_i^A + n_i \ldots \max K_i^A + n_i\right] = \sigma_A(i)$ hold. In other words, row i contains exactly w_i aligned so that the window column 0 contains its n_i'th character. (Technically, this leaves the structure of rows $k+1, \ldots$ open. In string formulae, however, the truth definition does not modify the assignment so these extra rows can safely be ignored. Note also that FSA's differentiate between the left and right ends of the empty string ε while alignments do not.) For example, Fig. 3 shows the tape configuration corresponding to the alignment in Fig. 1. Now the truth of a window formula in a window column (definitions 1–5) carries naturally over to the corresponding tape configurations, where $A \models (x_i = \bot)$ is interpreted as "tape i contains either a ¢ or a \$".

Then we prove the following stronger claim by induction on the structure of ϕ:

> There is a computation of $\mathcal{A}_\phi$ starting at a configuration C and ending at a configuration C' *if and only if* $L(\phi)$ contains a formula word $\psi = \tau_1\alpha_1 \ldots \tau_m\alpha_m$ such that C corresponds to the alignments A such that $A \models \psi$ and C' corresponds to alignments $A' = \tau_m(\ldots(\tau_1 A)\ldots)$.

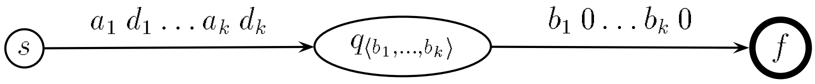


Figure 4: One path in the implementation of an atomic string formula.

This claim is needed when $\phi_1.\phi_2$ is translated by joining the inductively obtained $\mathcal{A}_{\phi_1}$ and $\mathcal{A}_{\phi_2}$ in a serial manner, so that the latter can safely start in whatever tape configuration the former halted. Moreover, Properties (1) to (4) are enforced during the construction. Property (5) is also enforced, but in an amended form, where the first configuration of the computation need not be initial. This amendment is again required for the orderly concatenation of FSAs.

Let us now proceed with the induction. For an atomic string formula transposing left $[x_{i_1}, \dots, x_{i_p}]_l \psi$ the state space will contain the nonfinal start state s; the final state f satisfying the requirements of Property (3); and an intermediate nonfinal state $q_{\langle b_1,\dots,b_k\rangle}$ for every $\langle b_1, \dots, b_k\rangle \in (\Sigma \cup \{¢, \$\})^k$. These are used to enforce the invariant for Property (5). The transition graph will consist of the two-edge paths in Fig. 4 where the index $\langle b_1, \dots, b_k\rangle$ corresponds to a window column satisfying ψ (restricted to the relevant rows $1, \dots, k$). Furthermore, for every tape $j = 1, \dots, k$ the direction d_j and the index character b_j are chosen to fulfill the restriction that $d_j = +1$ and $b_j \in \Sigma \cup \{\$\}$ if j is one of the transposed variable indices $i_1, \dots, i_p$ and $a_i \neq \$$. For the other j we require that $d_j = 0$ and $b_j = a_j$.

Consider now all the transitions entering f generated in Fig. 4. By the construction, these are exactly on the character combinations $b_1, \dots, b_k$ such that ψ is true in the corresponding alignments, as they must have $A(i, 0) = g(b_i)$ for each tape $i = 1, \dots, k$, where $g(c) = c$ when $c \in \Sigma$ and undefined for ¢ and \$. These transitions are stationary, so these alignments A' are also the ones corresponding to the accepting configurations C'. Consider then all the transitions entering an intermediate state $q_{\langle b_1,\dots,b_k\rangle}$ from s generated in Fig. 4. By the construction, these are exactly on the character combinations $a_1, \dots, a_k$ such that applying $d_1, \dots, d_k$ to the tapes can possibly result in $b_1, \dots, b_k$ appearing under the heads. Moreover, $d_i = +1$ if and only if variable x_i is among the transposed ones and $a_i \neq \$$; it is 0 otherwise. Hence, we have that the alignments A corresponding to the starting configurations C (in which the state is s) are exactly those that can yield some suitable A' when $[i_1, \dots, i_p]_l$ is applied to it. All in all, the existence of a computation from C to C' traversing a path generated in Fig. 4 – which by the construction are exactly the accepting computations – is equivalent to the existence of the corresponding A and $A' = [x_{i_1}, \dots, x_{i_p}]_l A$ such that $A \models \psi$, and the inductive claim holds. In addition, Property (5) holds (even for all the tapes and not just the unidirectional ones), since for every $a_1, \dots, a_k$ the following $b_1, \dots, b_k$ is possible for some suitable C and C'.

Properties (1) and (2) are easily seen from the construction. Property (4) holds except when in Fig. 4 we have $d_1 = \ldots = d_k = 0$ (and hence also $a_1 = b_1, \dots, a_k = b_k$); that is, the paths that begin with a stationary transition. Fortunately we can always *bypass* such transitions as shown in Fig. 5, where the bypassed transition pair is drawn with solid arrows while the new bypassing transition is dashed. This is the FSA version of the well-known technique for not generating ε-transitions when converting a regular expression into a traditional Nondeterministic Finite Automaton (NFA) [8]. Note also that bypassing preserves Property (5). Property (3) can be enforced by removing every state without

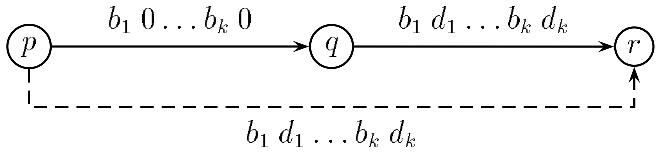


Figure 5: Bypassing a stationary transition.

incoming transitions except s.

Atomic string formulae transposing right are handled similarly; details are omitted.

For the concatenation $\phi_1.\phi_2$, $\mathcal{A}_\phi$ is constructed as follows: Obtain inductively $\mathcal{A}_{\phi_1}$ and $\mathcal{A}_{\phi_2}$. If $\mathcal{A}_{\phi_1}$ or $\mathcal{A}_{\phi_2}$ has no final state, then $\mathcal{A}_\phi$ will be a singleton rejecting start state. This is correct, since by induction $L(\phi_1.\phi_2)$ contains no satisfiable formula word ψ.

Assume then that $\mathcal{A}_{\phi_1}$ and $\mathcal{A}_{\phi_2}$ each do have a final state f_1 and f_2. Merge f_1 into the start state s_2 of $\mathcal{A}_{\phi_2}$. Apply then the bypassing technique of Fig. 5 to the new incoming stationary transitions of s_2 to reinstate Property (4). Then Property (3) can also be reinstated by first deleting s_2 (together with the transitions entering and leaving it, which have now been bypassed), then checking reachability by standard techniques. If the start state s_1 of $\mathcal{A}_{\phi_1}$ (now also of $\mathcal{A}_\phi$) and the final state f_2 of $\mathcal{A}_{\phi_2}$ (now also of $\mathcal{A}_\phi$) are unconnected (because bypassing produced no transitions at all), $\mathcal{A}_\phi$ will contain just the start state s_1. Properties (1) and (2) follow directly from induction.

To see that the inductive claim holds, examine any accepting computation $\mathcal{C}$ of $\mathcal{A}_\phi$ from C to C'. By the construction, it can be uniquely divided into $\mathcal{C}_1 bb'\mathcal{C}_2$ where the segment $\mathcal{C}_t$ uses only transitions found already in $\mathcal{A}_{\phi_t}$ and configuration $b' = \langle q', n_1 + d_1, \ldots, n_k + d_k\rangle$ follows from configuration $b = \langle q, n_1, \ldots, n_k\rangle$ using a bypassing transition generated by the construction. That is, $\mathcal{A}_{\phi_1}$ had originally a transition $\langle\langle q, u_1[n_1], \ldots, u_k[n_k]\rangle, \langle f_1, 0, \ldots, 0\rangle\rangle$ and $\mathcal{A}_{\phi_2}$ a transition $\langle\langle s_2, u_1[n_1], \ldots, u_k[n_k]\rangle, \langle q', d_1, \ldots, d_k\rangle\rangle$. Hence, $\mathcal{C}_1 b\,\langle f_1, n_1, \ldots, n_k\rangle$ is an accepting computation of $\mathcal{A}_{\phi_1}$ and $\langle s_2, n_1, \ldots, n_k\rangle\, b'\mathcal{C}_2$ of $\mathcal{A}_{\phi_2}$ on $\langle u_1, \ldots, u_k\rangle$. By our stronger inductive claim this is equivalent to having formula words $\psi = \tau_1\alpha_1 \ldots \tau_m\alpha_m \in L(\phi_1)$ and $\psi' = \tau'_1\alpha'_1 \ldots \tau'_{m'}\alpha'_{m'} \in L(\phi_2)$ such that for the alignments A corresponding to C, A'' corresponding to both b and $\langle s_2, n_1, \ldots, n_k\rangle$ (as the state is ignored) and A' corresponding to C' the following three facts hold: $A \models \psi$, $A'' = \tau_m(\ldots(\tau_1 A)\ldots) \models \psi'$ and $A' = \tau_{m'}(\ldots(\tau'_1 A'')\ldots)$. This in turn is equivalent to the claim.

Property (5) is seen in a similar way: Let U_1 and U_2 be the set of unidirectional tapes in $\mathcal{A}_{\phi_1}$ and $\mathcal{A}_{\phi_2}$, respectively. Any path P through $\mathcal{A}_\phi$ can be partitioned with respect to its bypassing transition into $P_1\,\langle\langle p, c_1, \ldots, c_k\rangle, \langle p', d_1, \ldots, d_k\rangle\rangle\, P_2$; that is, into subpath P_1 before the bypassing transition, the transition itself and subpath P_2 after it. Property (5) applies inductively to the paths $P_1\,\langle\langle p, c_1, \ldots, c_k\rangle, \langle f_1, 0, \ldots, 0\rangle\rangle$ and $\langle\langle s_2, c_1, \ldots, c_k\rangle, \langle p', d_1, \ldots, d_k\rangle\rangle\, P_2$ where the bypassing transition has been revoked. Let then the accepting configuration of $\mathcal{A}'_{\phi_1}$ be $C''_1 = \langle f_1, \ldots, n_i, \ldots\rangle$ in the corresponding computation $\mathcal{C}_1 C''_1$ on some input strings v_i where $i \in U_1$ and the initial configuration of $\mathcal{A}'_{\phi_2}$ be $C''_2 = \langle s_2, \ldots, n_j, \ldots\rangle$ in the corresponding computation $C''_2\mathcal{C}_2$ on some other input strings w_j where $j \in U_2$. For any unidirectional tape $i \in U_1$ of $\mathcal{A}_{\phi_1}$ $\mathcal{C}_1 C''_1$ has never visited the characters $v_i[n_i + 1], v_i[n_i + 2], v_i[n_i + 3], \ldots$ to the right of the head; similarly, for any unidirectional tape $j \in U_2$ of $\mathcal{A}_{\phi_2}$ $C''_2\mathcal{C}_2$ has not visited the characters

$w_j[n'_j - 1], w_j[n'_j - 2], w_j[n'_j - 3], \ldots$ to the left of the head. (This obviously fails for the bidirectional tapes.) Moreover, by the bypassing construction it must be the case that $v_i[n_i] = w_i[n'_i] = c_i$ for all the tapes $i = 1, \ldots, k$ in $\mathcal{A}_\phi$ and hence especially its unidirectional tapes $U = U_1 \cap U_2$. We can construct the desired computation on the input strings $u_h = w_h[1 \ldots n'_h - 1]v_h[n_h \ldots |v_h|]$ for $h \in U$ as $\mathcal{C}'_1 C'' \mathcal{C}'_2$ where $\mathcal{C}'_1$ is $\mathcal{C}_1$ disregarding the tapes not in U; $\mathcal{C}'_2$ is $\mathcal{C}_2$ disregarding the tapes not in U, taking into account that character $w_h[n_h + r]$ is now found in tape position $n'_h + r$ instead – that is, $n'_h - n_h$ is added to all indices $h \in U$; and C'' is the configuration replacing C''_1 and C''_2 as a result of applying the bypassing transition instead. This completes the concatenation case.

The construction of $\mathcal{A}_\phi$ for the Kleene closure ψ^* proceeds as follows: obtain inductively $\mathcal{A}_\psi$. If it contains just a start state s, it suffices unmodified. Otherwise add a new final state f' and a stationary transition from s into it on every character combination from $(\Sigma \cup \{¢, \$\})^k$. Intuitively, these express not entering the loop $\mathcal{A}_\psi$ at all. Remove all stationary transitions from s to the original final state f of $\mathcal{A}_\psi$, because they are already represented with the transitions added in the preceding step. Implement the loop $\mathcal{A}_\psi$ by merging f into s and apply the bypassing technique of Fig. 5, following the concatenation case above. That is, we interpret ψ^* as the infinite repeating formula $\lambda + (\psi.(\lambda + \psi.(\ldots)))$. Finally apply reachability analysis to reclaim Property (3). The claim and properties follow from a straightforward inductive generalization of the argumentation presented in the concatenation case above; details are omitted.

In selection $\phi_1 + \phi_2$ we simply merge together the start state s_1 of $\mathcal{A}_{\phi_1}$ and s_2 of $\mathcal{A}_{\phi_2}$ to form the new start state, and their respective final states f_1 and f_2 (if any) to form the new final state. The claim, properties and invariant follow directly from induction. Hence, the whole claim is proved.

Now, given a string formula ϕ on variables $x_1, \ldots, x_k$ to translate into a k-FSA, we use the claim proved above to first construct a $\mathcal{A}_\phi$ for which the stronger claim holds. Then we concatenate (in the sense described above) it at the end of the single-transition k-FSA $\langle\langle s, ¢, \ldots, ¢\rangle, \langle f, 0, \ldots, 0\rangle\rangle$ with start state s and final state f. Intuitively, this k-FSA tests that the alignment in which the truth is determined is indeed initial. This removes all paths from the start state which start in a non-initial tape configuration; thus Property (5) is now made to hold even without the amendment. $\square$

The converse of Theorem 3.1 is also true, so the computational power of k-FSAs correspond exactly to string formulae over k variables.

Theorem 3.2 *For each k-FSA $\mathcal{A}$ there is a string formula $\phi_\mathcal{A}$ on variables $x_1, \ldots, x_k$ such that $[\![\phi_\mathcal{A}]\!] = L(\mathcal{A})$.*

Moreover, variable x_i is bidirectional if and only if tape i is.

Proof: The idea is to describe each transition $t = \langle\langle p, c_1, \ldots, c_k\rangle, \langle q, d_1, \ldots, d_k\rangle\rangle$ of $\mathcal{A}$ with a string formula

$$\psi_t = [\,]_l \left(\bigwedge_{i=1}^{k} x_i = c'_i \right) .\tau_l^t \top . \tau_r^t \top$$

where $c'_i = c_i$ when $c_i \in \Sigma$ and '$\perp$' otherwise. Here $[\,]_l$ denotes the left transpose of no variables, which reduces to the identity function on transposes by definition. In addition x_i appears in τ_l^t (τ_r^t, respectively) if and only if $d_i = +1$ (-1, respectively) in t. The intention is that t can be taken in a configuration $C = \langle p, n_1, \ldots, n_k\rangle$ on input $\langle u_1, \ldots, u_k\rangle$

if and only if ψ_t is true in the corresponding alignments A. (Recall the correspondence between configurations and alignments from the proof of Theorem 3.1. We also omit mentioning the assignment θ explicitly here as well.) Moreover, $\tau_r^t \tau_l^t A$ corresponds to the resulting tape configuration $\langle q, n_1 + d_1, \ldots, n_k + d_k \rangle$. That is, ψ_t expresses exactly the operational semantics of t. Then the final string formula $\phi_{\mathcal{A}}$ is written to accept the formula word $\psi_{t_1} \ldots \psi_{t_m}$ if and only if the transitions $t_1 \ldots t_m$ form a path through $\mathcal{A}$; any standard conversion from nondeterministic finite automata to regular expressions suffices. This also implies the extra bidirectionality condition on $\phi_{\mathcal{A}}$, since τ_t performs exactly the transpositions corresponding to the tape movements in t.

We can safely assume that $\mathcal{A}$ has an unique final state f which is also distinct from the start state and has no outgoing transitions. A slight complication arises since string formulae do not differentiate between the left and right ends of a string, while FSAs do. Namely, if any state p did have transitions $t_1 = \langle \langle p, \ldots, c_{i-1}, ¢, c_{i+1}, \ldots \rangle, \langle q_1, \ldots \rangle \rangle$ and $t_2 = \langle \langle p, \ldots, c_{i-1}, \$, c_{i+1}, \ldots \rangle, \langle q_2, \ldots \rangle \rangle$ it would be wrong to indiscriminately test for $x_i = \perp$ in both ψ_{t_1} and ψ_{t_2}. Fortunately $\mathcal{A}$ can be normalized to get rid of these conflicts by indexing the state space as in Theorem 3.1: the state space will consist of $p_{\langle b_1, \ldots, b_k \rangle}$ where $b_i \in \{¢, \star, \$\}$ denotes whether the state can be exited only on the condition that the head of tape i must be on ¢, between the endmarkers, or on \$. There will be a transition

$$\left\langle \left\langle p_{\langle b_1, \ldots, b_k \rangle}, c_1, \ldots, c_k \right\rangle, \left\langle q_{\langle e_1, \ldots, e_k \rangle}, d_1, \ldots, d_k \right\rangle \right\rangle$$

if and only if there originally was a transition

$$\langle \langle p, c_1, \ldots, c_k \rangle, \langle q, d_1, \ldots, d_k \rangle \rangle$$

and for each tape $i = 1, \ldots, k$ one of the following cases holds:

- $d_i = 0$ and $b_i = e_i$; that is, the tape does not move.
- $d_i = +1$ and $e_i \in \{\star, \$\}$; that is, the tape moves left, and may end up between the endmarkers or on the right one (but certainly not on the left one).

  Note that the transitions leaving $q_{\langle e_1, \ldots, e_k \rangle}$ verify that the index is indeed respected.
- Symmetrically $d_i = -1$ and $e_i \in \{¢, \star\}$.

The new start state will be $s_{\langle ¢, \ldots, ¢ \rangle}$, where s is the original start state, because in the beginning all heads are known to be on ¢. The states $f_{\langle \ldots \rangle}$ will be the new final states. By the assumption, no transitions exit from them, and so they can even be merged into one new unique final state. Every state and transition not on any path from the new start state to the new final state can be ignored.

Let now the states of the normalized $\mathcal{A}$ be numbered $1, \ldots, n$ so that the start state is assigned number 1 and the final one n (in addition, $n \geq 2$). Following [22, Theorem 3.17] we define inductively the formulae E_{ijk} to express the paths from state i to state j not going through any of the states $k+1, \ldots, n$: initially $E_{ij0} = \psi_{t_1} + \ldots + \psi_{t_a}$ where $t_1, \ldots, t_a$ are the transitions from i to j. (If there are none, $E_{ij0} = [\,]_l \neg \top$ instead to signal that no valid computation can proceed in one step from state i to state j.) Inductively

$$E_{ijk} = \begin{cases} E^{+}_{ij(k-1)} & i = j = k \\ E_{ij(k-1)} . E^{*}_{jk(k-1)} & i \neq j = k \\ E^{*}_{ik(k-1)} . E_{kj(k-1)} & i = k \neq j \\ E_{ij(k-1)} + E_{ik(k-1)} . E^{*}_{kk(k-1)} . E_{kj(k-1)} & i \neq j \neq k \end{cases}$$

where the results are simplified using the rules

$$\begin{aligned} E.[\,]_l\neg\top &= [\,]_l\neg\top.E \\ &= [\,]_l\neg\top \\ E.([\,]_l\neg\top)^* &= ([\,]_l\neg\top)^*.E \\ &= E \\ E+[\,]_l\neg\top &= [\,]_l\neg\top+E \\ &= E \end{aligned}$$

to eliminate the unsatisfiable atomic string formulae '$[\,]_l\neg\top$' from the result (or to reduce the formula into one if the start and final states of $\mathcal{A}$ are not connected with each other by transitions). Now $\phi_{\mathcal{A}} = E_{1nn}$ and $\phi_{t_1}\dots\phi_{t_m}$ (viewed as a formula word) is a member of $L(\phi_{\mathcal{A}})$ if and only if $t_1\dots t_m$ is a path of transitions through $\mathcal{A}$. For example, the first case says that the paths of at least one transition from state i back to itself visiting only states $1,\dots,i$ are obtained by repeating the paths from i back to itself visiting $1,\dots,i-1$.

Assume now that $C_0\dots C_m$ is an accepting computation of $\mathcal{A}$ on $\langle u_1,\dots,u_k\rangle$ traversing the path $t_1\dots t_m$. For all $i=1,\dots,m$ the following two facts hold: by the construction we have $A_{i-1}\models\psi_{t_i}$, where A_{i-1} is the alignment corresponding to C_{i-1}, exactly when t_i is possible in C_{i-1}. We also have $A_i=\tau_r^i\tau_l^i A_{i-1}$ exactly when C_i can follow C_{i-1} as a result of applying t_i. Hence, the assumption is equivalent to having $A_0\models\psi_{t_1}\dots\psi_{t_m}$. Therefore equivalently $A_0\models\phi_{\mathcal{A}}$, and A_0 is the initial alignment corresponding to the initial configuration C_0, as required. □

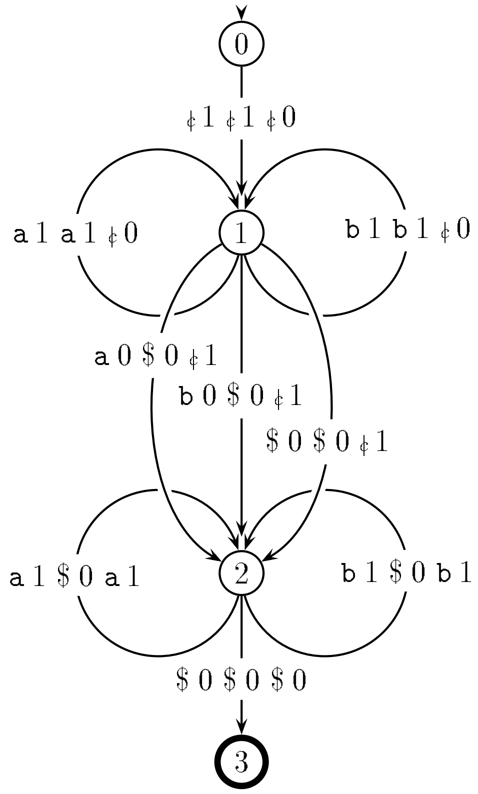
The most interesting property of FSA's (and hence of string formulae as well) for our application is the following.

Definition 3.1 *Let $\mathcal{A}$ be a $(k+l)$-FSA. Its tapes* $1,\dots,k$ limit *the other tapes* $k+1,\dots,k+l$ *if and only if there is a function $\mathcal{W}_{\mathcal{A}}:\mathbb{N}^k\to\mathbb{N}$ such that $\langle u_1,\dots,u_k,v_1,\dots,v_l\rangle\in L(\mathcal{A})$ implies $|v_i|\le\mathcal{W}_{\mathcal{A}}(|u_1|,\dots,|u_k|)$ for every $i=1,\dots,l$.*

We denote this condition $\{1,\dots,k\}\rightsquigarrow\{k+1,\dots,k+l\}$. The *limitation problem* is to determine the existence (and preferably even a method of computing the values) of $f_{\mathcal{A}}$ given $\mathcal{A}$, k and l. Intuitively the limitation problem is, "Consider $\mathcal{A}$ no longer an acceptor but a kind of a generalized *Mealy machine* [11, Chapter 2.7] with input tapes $1,\dots,k$, output tapes $k+1,\dots,k+l$, nodeterministic control and no requirement to consume input while producing output. Does $\mathcal{A}$ have any input $\langle u_1,\dots,u_k\rangle$ for which infinitely many different outputs $\langle v_1,\dots,v_l\rangle$ are possible?" The problem extends naturally to any partition of tapes, or equivalently variables, to inputs and outputs. Hence, we try to determine the ways in which a given string acceptance device can be used safely as a string production device in this particular query, rather than forcing the database programmer to work with explicit string production devices as in [7, 15, 26]. (Related transducer problems have been studied by Weber [27, 28].) We defer, however, further discussion of the limitation problem to Section 4.

In Fig. 6 we show an example of a string formula and a corresponding finite state acceptor drawn as a graph. The alphabet Σ is for simplicity assumed to be $\{\mathtt{a},\mathtt{b}\}$.

Next we examine briefly the problem of actually *simulating* an FSA to see if it accepts the given inputs. First an auxiliary result showing how to specialize an FSA given some of its inputs as constants.



$$([x, y]_\imath x = y)^*.([x, z]_\imath x = z)^*.([x, y, z]_\imath x = y = z = \perp)$$

Figure 6: A string formula and a corresponding 3-FSA

Lemma 3.1 *Let $\mathcal{A}$ be a $(k+l)$-FSA and $u_1, \ldots, u_k \in \Sigma^*$ strings. Then we can construct an l-FSA $\mathcal{B}$ where $L(\mathcal{B}) = \{\langle v_1, \ldots, v_l \rangle : \langle u_1, \ldots, u_k, v_1, \ldots, v_l \rangle \in L(\mathcal{A})\}$ in polynomial time with respect to $|\mathcal{A}| \prod_{i=1}^{k}(|u_i| + 2)$.*

Proof: Each state of $\mathcal{B}$ is of the form $p_{\langle n_1, \ldots, n_k \rangle}$ where p is a state of $\mathcal{A}$ and $0 \leq n_i \leq |u_i| + 1$ for each $i = 1, \ldots, k$. Such a state is final if and only if p is. The start state is $s_{\langle 0, \ldots, 0 \rangle}$ where s is the start state of $\mathcal{A}$.

Each transition $\langle \langle p, c_1, \ldots, c_{k+l} \rangle, \langle q, d_1, \ldots, d_{k+l} \rangle \rangle$ of $\mathcal{A}$ induces into $\mathcal{B}$ the transitions

$$\left\langle \left\langle p_{\langle n_1, \ldots, n_k \rangle}, c_{k+1}, \ldots, c_{k+l} \right\rangle, \left\langle q_{\langle n_1 + d_1, \ldots, n_k + d_k \rangle}, d_{k+1}, \ldots, d_{k+l} \right\rangle \right\rangle$$

where $u_i[n_i] = c_i$ for all $i = 1, \ldots, k$. This yields the stated time bound from the maximum number of transitions induced into $\mathcal{B}$ by a transition of $\mathcal{A}$.

In other words, $\mathcal{B}$ remembers in its finite control where the heads would be on the tapes $1, \ldots, k$ of $\mathcal{A}$ given their contents $u_1, \ldots, u_k$. This also induces a one-to-one correspondence between a configuration $\langle p, n_1, \ldots, n_{k+l} \rangle$ of $\mathcal{A}$ on input $\langle u_1, \ldots, u_k, v_1, \ldots, v_l \rangle$ and a configuration $\left\langle p_{\langle n_1, \ldots, n_k \rangle}, n_{k+1}, \ldots, n_{k+l} \right\rangle$ of $\mathcal{B}$ on input $\langle v_1, \ldots, v_l \rangle$ from which correctness follows. □

The net result is that for any *fixed* FSA the acceptance problem is polynomial.

Theorem 3.3 *Let $\mathcal{A}$ be a fixed k-FSA. Then $L(\mathcal{A}) \in$ **P**.*

Proof: Given the input $U = \langle u_1, \ldots, u_k \rangle$ we can check if $\mathcal{A}$ accepts it in time polynomial with respect to $|u_1|, \ldots, |u_k|$ with the following algorithm.

Apply Lemma 3.1 to $\mathcal{A}$ and U (that is, $l = 0$). Because $\mathcal{A}$ is predefined, $|\mathcal{A}|$ and k are now constants, and the time required is polynomial with respect to $|u_1|, \ldots, |u_k|$. The result is a 0-FSA $\mathcal{B}$ of polynomial size: a graph whose nodes correspond to the configurations of $\mathcal{A}$ on U, and whose edges point to the possible next configurations. Now $\mathcal{A}$ has an accepting computation on U if and only if $\mathcal{B}$ contains a path from the initial state to a final state without outgoing edges. Checking this is polynomial with respect to $|\mathcal{B}|$. □

If we now turn our attention to the problem of evaluating queries formulated in the full language, the first issue we have to tackle is the possible infiniteness of the answer set. For instance, the answer to the query $x|([x]_l x = \mathtt{a})^*([x]_l x = \bot)$ is the infinite set $\{\varepsilon, \mathtt{a}, \mathtt{aa}, \ldots\}$. Since our domain (i.e. Σ) is finite, a necessary and sufficient condition for finite answers would be that every component in a tuple in the answer is of bounded length. This concept of *(semantic) safety* corresponds to the full semantics of Section 2 where quantification ranges over the entire Σ^*, making it the Alignment Calculus counterpart for the *(output-restricted) unlimited interpretation* of the traditional relational calculus [12]. In our framework this definition is unrealistic, however. For example, the formula $\phi^\omega : \forall x \exists y.(x = y)^*(x = \bot \wedge y \neq \bot)$ states that there exist arbitrarily long strings. Viewed as a query, this indeed does have a finite answer, namely the full relation of arity 0; this conclusion cannot be drawn, however, from an inspection of any finite active domain. In other words, the query ϕ^ω itself adds an infinite set of constants into the active domain. This means that, although in the relational calculus one can use essentially any semantics for finite queries (with perhaps an exponential blowup in formula length) [12, Theorem 3.1], this is no longer the case in Alignment Calculus. Hence, we restrict ourselves to formulae whose validation needs only a finite amount of information, although we might not *a priori* know how much. In other words, we adopt theoretically the *countable invention interpretation* and seek in practice to limit it further into the *finite* case.

Definition 3.2 *The Alignment Calculus formula ϕ is* domain independent *if and only if there exists a limit function $\mathcal{W}_\phi\colon DB \to \mathbb{N}$ such that $[\![\phi]\!]_{db} = [\![\phi]\!]^{\ell}_{db}$ for every database db and constant $\ell \geq \mathcal{W}_\phi(db)$.*

The intuition is that the finite domain $\Sigma^{\leq \mathcal{W}_\phi(db)}$ is now a sufficient active domain for the correct evaluation of $[\![\phi]\!]_{db}$. It is essential to let $\mathcal{W}_\phi$ depend not only on ϕ but also on db; otherwise there would be no domain independent way to express for example the safe query asking for the strings which are a concatenation of a string from R_1 and another from R_3. The safety concept of Ginsburg and Wang [7], for example, falls short in this respect.

As an example of a safe query, consider $x|R(x) \wedge ([x]_l x = \mathtt{a})^*([x]_l x = \bot)$ The answer set is finite because the candidate strings have to appear in database relation R; any corresponding $\mathcal{W}$ majorizes the length of the longest string in R. An answer set can of course also be finite as a consequence of the finiteness of the language specified by a string formula. (More involved reasons for finiteness are studied in Section 4 by considering the constraints induced by the limitation properties of string formulae as in Definition 3.1.) In either case there is an explicit bound on the active domain depending on the formula and the database.

Let us now define *Alignment Algebra*, our string-oriented counterpart to classical relational algebra:

- If R is a relation symbol of arity k, it is also an expression in Alignment Algebra of the same arity. Its *value* when applied to particular database db is $db(R)$, the relation assigned to symbol R by db.

- Σ^* is an expression of arity 1 whose value is Σ^*, the set of all finite strings from the alphabet Σ.

  In addition $\Sigma^{\leq l}$, where $l \in \mathbb{N}$ is a constant, is an expression of arity 1 with $db(\Sigma^{\leq l}) = \{u \in \Sigma^*\colon |u| \leq l\}$ as its value.

  In other words, we include explicit *domain symbols* for both the underlying infinite domain Σ^* and its finite truncations $\Sigma^{\leq l}$.

- If E and F are expressions with equal arities, then $E \cup F$ is an expression of this common arity with $db(E \cup F) = db(E) \cup db(F)$.

- If E and F are expressions with equal arities, then $E \setminus F$ is an expression of this common arity with $db(E \setminus F) = db(E) \setminus db(F)$.

- $E \cap F$ is a shorthand for $E \setminus (E \setminus F)$.

- If E and F are expressions, then $E \times F$ is an expression of arity $a(E) + a(F)$ with $db(E \times F) = db(E) \times db(F)$.

- If E is an expression and $[i_1, \ldots, i_u]$ a list of distinct numbers between 1 and $a(E)$, then $\pi_{[i_1,\ldots,i_u]}E$ is an expression of arity u with

$$db(\pi_{[i_1,\ldots,i_u]}E) = \left\{ \langle v_{i_1}, \ldots, v_{i_u} \rangle : \left\langle v_1, \ldots, v_{a(E)} \right\rangle \in db(E) \right\}.$$

  It is also possible to have $u = 0$ in which case $db(\pi_{[\,]}E)$ is a relation of arity 0; it is $\emptyset$ if $db(E) = \emptyset$ and the full relation $\{\langle\,\rangle\}$ otherwise. Note that $\{\langle\,\rangle\}$ can be represented algebraically with for example $\pi_{[\,]}\Sigma^{\leq 0}$.

- If E is an expression and $\mathcal{A}$ an $a(E)$-FSA, then $\sigma_{\mathcal{A}}E$ is an expression of arity $a(E)$ with $db(\sigma_{\mathcal{A}}E) = db(E) \cap L(\mathcal{A})$. In other words, selection is performed by retaining only the tuples of E that $\mathcal{A}$ accepts.

- There are no other expressions in Alignment Algebra.

An expression E is said to be *finitely evaluable* if Σ^* appears only in subexpressions of the form $\sigma_{\mathcal{A}}(F \times (\Sigma^*)^n)$, where F is finitely evaluable and $\mathcal{A}$ fulfills $\{1, \ldots, a(F)\} \rightsquigarrow \{a(F)+1, \ldots, a(F)+n\}$. This subclass is interesting because of the intuition behind Definition 3.1: while the infinitary operator Σ^* is unimplementable as such, in finitely evaluable subexpressions it is used merely as the finite output of the generalized Mealy machine $\mathcal{A}$.

First we prove that Alignment Algebra is no more expressive than Alignment Calculus, and further that finitely evaluable expressions are no more expressive than domain independent queries.

Theorem 3.4 *For each Alignment Algebra expression E there exists an Alignment Calculus formula ϕ_E with free variables $x_1, x_2, \ldots, x_{a(E)}$ such that $db(E) = [\![\phi_E]\!]_{db}$ for every database db.*

Moreover, if E is finitely evaluable then ϕ_E is domain independent.

Proof: The claims can be proved by induction on the structure of E.

For a relation symbol R the corresponding formula ϕ_R is $R(x_1, \ldots, x_{a(R)})$: by definition $\left\langle u_1, \ldots, u_{a(R)} \right\rangle \in [\![\phi_R]\!]_{db}$ if and only if there exists a full interpretation A_0, θ where $\sigma_{A_0}(\theta x_i) = u_i$ for all $i = 1, \ldots, a(R)$ and $\langle A_0, db \rangle \models \phi_R\theta$. By the truth definition this is equivalent in turn to $\left\langle u_1, \ldots, u_{a(R)} \right\rangle \in db(R)$, proving that $db(R) = [\![\phi_R]\!]_{db}$ as required. Choose for the second claim $\mathcal{W}_{\phi_R}(db) = \max(R, db)$, where

$$\max(R, db) = \max\{|u| : \langle \ldots, u, \ldots \rangle \in db(R)\} \tag{2}$$

extracts the length of the longest string in $db(R)$. Hence, A_0^ℓ never truncates a string appearing in R when $\ell \geq \mathcal{W}_{\phi_R}(db)$, and so $\langle A_0, db \rangle \models \phi_R\theta$ if and only if $\left\langle A_0^\ell, db \right\rangle \models \phi_R\theta$ proving that $[\![\phi_R]\!]_{db} = [\![\phi_R]\!]_{db}^\ell$ as required.

It suffices to select any identically true formula with free variable x_1 for ϕ_{Σ^*}; take for example $[\,]_l x_1 = \bot$. It is not domain independent but neither is Σ^* finitely evaluable by itself.

Choosing $\phi_{\Sigma^{\leq k}} = ([x_1]_l \top)^k . [x_1]_l x_1 = \bot$ yields (recalling that a left transpose has no effect when $A_0(\theta x_1, 0) = A_0(\theta x_1, 1) = \bot$) a formula that is true if and only if $|\sigma_{A_0}(\theta x_1)| \leq$

k as required. Such a formula is also domain independent with $\mathcal{W}_{\phi_{\Sigma \leq k}}(db) = k$ since $[\![\phi_{\Sigma \leq k}]\!]^{\ell}_{db} = \Sigma^{\leq \min(k,\ell)}$ by construction.

For union choose $\phi_{E \cup F} = \phi_E \vee \phi_F$. Now $\mathbf{u} = \langle u_1, \ldots, u_a \rangle \in [\![\phi_{E \cup F}]\!]_{db}$ (where a is the common arity of E and F) if and only if there exists a full interpretation A_0, θ such that $\sigma_{A_0}(\theta x_i) = u_i$ for all $i = 1, \ldots, a$ where $\langle A_0, db \rangle \models \phi_E \theta$ or $\langle A_0, db \rangle \models \phi_F \theta$ holds. By definition this is equivalent to having $\mathbf{u} \in [\![\phi_E]\!]_{db}$ or $\mathbf{u} \in [\![\phi_F]\!]_{db}$. That is,

$$[\![\phi_{E \cup F}]\!]_{db} = [\![\phi_E]\!]_{db} \cup [\![\phi_F]\!]_{db} \tag{3}$$

from which $[\![\phi_{E \cup F}]\!]_{db} = db(E) \cup db(F) = db(E \cup F)$ by induction as required. $E \cup F$ is finitely evaluable if and only if both E and F are; then by induction both ϕ_E and ϕ_F are domain independent, and the function $\mathcal{W}_{\phi_{E \cup F}}(db) = \max(\mathcal{W}_{\phi_E}(db), \mathcal{W}_{\phi_F}(db))$ exists. When $\ell \geq \mathcal{W}_{\phi_{E \cup F}}(db)$ we can calculate as follows:

$$\begin{aligned}
[\![\phi_{E \cup F}]\!]_{db} &= [\![\phi_E]\!]_{db} \cup [\![\phi_F]\!]_{db} \text{ by Eq. (3)} \\
&= [\![\phi_E]\!]^{\ell}_{db} \cup [\![\phi_F]\!]^{\ell}_{db} \text{ by the choice of } \ell \\
&= \left\{ \left\langle \sigma_{A_0^{\ell}}(\theta x_{i_1}), \ldots, \sigma_{A_0^{\ell}}(\theta x_{i_k}) \right\rangle : \left\langle A_0^{\ell}, db \right\rangle \models \phi_E \theta \right\} \bigcup \\
&\quad \left\{ \left\langle \sigma_{A_0^{\ell}}(\theta x_{i_1}), \ldots, \sigma_{A_0^{\ell}}(\theta x_{i_k}) \right\rangle : \left\langle A_0^{\ell}, db \right\rangle \models \phi_F \theta \right\} \\
&= \left\{ \left\langle \sigma_{A_0^{\ell}}(\theta x_{i_1}), \ldots, \sigma_{A_0^{\ell}}(\theta x_{i_k}) \right\rangle : \left\langle A_0^{\ell}, db \right\rangle \models \phi_E \theta \text{ or } \left\langle A_0^{\ell}, db \right\rangle \models \phi_F \theta \right\} \\
&= [\![\phi_{E \cup F}]\!]^{\ell}_{db}
\end{aligned}$$

Hence, the second claim holds as well.

Choosing $\phi_{E \setminus F} = \phi_E \wedge \neg \phi_F$ can be justified by a similar calculation. So can $\phi_{E \times F} = \phi_E \wedge \phi'_F$, where ϕ'_F is obtained from ϕ_F by renaming each free variable x_i into $x_{i+a(E)}$. In both cases the expression is finitely evaluable if and only if both E and F are, so $\mathcal{W}(db) = \max(\mathcal{W}_{\phi_E}(db), \mathcal{W}_{\phi_F}(db))$ is again the corresponding limit function by a similar calculation as above.

For a selection choose $\phi_{\sigma_{\mathcal{A}} E} = \phi_E \wedge \phi_{\mathcal{A}}$, where $\phi_{\mathcal{A}}$ is supplied by Theorem 3.2: $[\![\phi_{\sigma_{\mathcal{A}} E}]\!]_{db} = [\![\phi_E]\!]_{db} \cap [\![\phi_{\mathcal{A}}]\!] = db(E) \cap L(\mathcal{A}) = db(\sigma_{\mathcal{A}} E)$. If $\sigma_{\mathcal{A}} E$ is finitely evaluable, then E is of the form $F \times (\Sigma^*)^l$ where F is inductively finitely evaluable and $\mathcal{A}$ fulfills the condition $\{1, \ldots, a(F)\} \rightsquigarrow \{a(F)+1, \ldots, a(F)+l\}$ via some function $f_{\mathcal{A}}\colon \mathbb{N}^{a(F)} \to \mathbb{N}$. Hence, the function

$$\mathcal{W}_{\phi_{\sigma_{\mathcal{A}} E}}(db) = \max(\mathcal{W}_{\phi_F}(db), f_{\mathcal{A}}(\mathcal{W}_{\phi_F}(db), \ldots, \mathcal{W}_{\phi_F}(db)))$$

does exist. When $\ell \geq \mathcal{W}_{\phi_{\sigma_{\mathcal{A}} E}}(db)$ we have $[\![\phi_F]\!]_{db} = [\![\phi_F]\!]^{\ell}_{db}$ by definition; in particular, $\left\langle u_1, \ldots, u_{a(F)} \right\rangle \in [\![\phi_F]\!]_{db}$ implies $|u_i| \leq \mathcal{W}_{\phi_F}(db)$ for all $i = 1, \ldots, a(F)$. This in turn means that $\left\langle u_1, \ldots, u_{a(F)}, v_1, \ldots, v_l \right\rangle \in L(\mathcal{A})$ implies $|v_j| \leq f_{\mathcal{A}}(\mathcal{W}_{\phi_F}(db), \ldots, \mathcal{W}_{\phi_F}(db)))$, since $f_{\mathcal{A}}$ is nondecreasing by its definition. Hence, $[\![\phi_{\sigma_{\mathcal{A}} E}]\!]_{db} = [\![\phi_F \wedge \phi_{\Sigma^*}(x_{a(F)+1}) \wedge \ldots \wedge \phi_{\Sigma^*}(x_{a(F)+l}) \wedge \phi_{\mathcal{A}}]\!]_{db} = [\![\phi_F \wedge \phi_{\mathcal{A}}]\!]_{db} = [\![\phi_F \wedge \phi_{\mathcal{A}}]\!]^{\ell}_{db} = [\![\phi_{\sigma_{\mathcal{A}} E}]\!]^{\ell}_{db}$ as required.

Finally for a projection $\pi_{[i_1, \ldots, i_u]} E$ choose $\exists y_{j_1} \ldots \exists y_{j_{a(E)-u}} \phi'_E$, where induction provides a formula ϕ_E for E whose free variables are (without loss of generality) denoted with alternative symbols $y_1, \ldots, y_{a(E)}$; the indices $j_1, \ldots, j_{a(E)-u}$ are the ones from $1, \ldots, a(E)$ not appearing in $i_1, \ldots, i_u$; and ϕ'_E is obtained from ϕ_E by renaming the free occurrences of y_{i_k} to x_k for $k = 1, \ldots, u$. In other words, projecting a column away translates into a quantification and the reordering of the remaining columns into a renaming of the remaining free variables. Now

$$\begin{aligned}
db(\pi_{[\imath_1,\ldots,\imath_u]}E) &= \left\{\langle v_{i_1}, \ldots, v_{i_u}\rangle : \left\langle v_1, \ldots, v_{a(E)}\right\rangle \in db(E)\right\} \\
&= \left\{\langle v_{i_1}, \ldots, v_{i_u}\rangle : \left\langle v_1, \ldots, v_{a(E)}\right\rangle \in [\![\phi_E]\!]_{db}\right\} \\
&= \left\{\langle w_1, \ldots, w_u\rangle : \left\langle w_{p(1)}, \ldots, w_{p(u)}\right\rangle \in [\![\exists y_{j_1} \ldots \exists y_{j_{a(E)-u}} \phi_E]\!]_{db}\right\}
\end{aligned} \tag{4}$$

holds. Here p is the permutation of $\{1, \ldots, u\}$ such that $p(k)$ is the position occupied by y_{i_k} when the free variables $y_{i_1}, \ldots, y_{i_u}$ are ordered by ascending index. Hence, the renaming of y_{i_k} to x_k in ϕ'_E is correct, and the first claim follows. If E is finitely evaluable, $\mathcal{W}_{\phi_E}$ exists. The second claim then follows directly by induction from Eq. (4). □

Now we proceed to prove the converse direction that all of Alignment Calculus can also be translated to Alignment Algebra. At the same time we demonstrate the role Σ^* plays in the result: it is the universe from which the strings $\sigma_{A_0}(\theta x)$ are drawn when verifying the truth of ϕ.

Theorem 3.5 *Let ϕ be a formula in Alignment Calculus. There exists a corresponding expression in Alignment Algebra E_ϕ such that $[\![\phi]\!]_{db} = db(E_\phi)$ for every database db.*

Moreover, for every constant $\ell \in \mathbb{N}$ it is true that $[\![\phi]\!]^\ell_{db} = db(E_\phi \restriction \ell)$, where $E_\phi \restriction \ell$ is obtained from E_ϕ by replacing every occurrence of Σ^* with an occurrence of $\Sigma^{\leq \ell}$.

Proof: Note that, given an Alignment Algebra expression F and an ordered partition $B = \langle B_1, \ldots, B_n\rangle$ of $\{1, \ldots, a(F)\}$ into nonempty parts B_j, one can construct an expression denoted $F \barwedge B$ such that

$$db(F \barwedge B) = \left\{\langle u_{b_1}, \ldots, u_{b_n}\rangle \,\middle|\, \langle u_1, \ldots, u_n\rangle \in db(F) \text{ and } p \in B_j \text{ implies } u_p =_s u_{b_j}\right\}.$$

In other words, B specifies which fields of a tuple from $db(F)$ must be equivalent, and $db(F \barwedge B)$ contains the compliant tuples with redundant columns eliminated and the remaining ones ordered as in B. This is accomplished with $F \barwedge B = \pi_{[\min B_1,\ldots,\min B_n]}\sigma_{\mathcal{A}_\phi} F$, where the $a(F)$-FSA $\mathcal{A}_\phi$ is obtained by applying Theorem 3.1 to the string formula

$$\phi\colon \left(\tau \bigwedge_{p \in B_j} x_p = x_{b_j}\right)^* .\tau x_1 = \ldots = x_{a(F)} = \bot$$

in which the left transpose τ consists of such x_j that $|B_j| > 1$. This ϕ encodes the desired selection formula

$$\bigwedge_{p \in B_j} x_p =_s x_{b_j}.$$

as a single string formula.

Then we construct E_ϕ by induction on the structure of ϕ: first consider as ϕ an atomic relational formula $R(\nu_1, \ldots, \nu_{a(R)})$ where each ν_j is a variable occurrence, and let $x_{i_1}, \ldots, x_{i_m}$ be the occurring variables ordered by ascending index. Build the partition B

with $B_k = \{j \colon \nu_j = x_{i_k}\}$ for $k = 1, \ldots, m$ and form $R \barwedge B$. Now for the first claim we can calculate

$$\begin{aligned}
[\![\phi]\!]_{db} &= \left\{\langle \sigma_{A_0}(\theta x_{i_1}), \ldots, \sigma_{A_0}(\theta x_{i_m})\rangle : \left\langle \sigma_{A_0}(\theta \nu_1), \ldots, \sigma_{A_0}(\theta \nu_{a(R)})\right\rangle \in db(R)\right\} \\
&= \left\{\langle \sigma_{A_0}(\theta x_{i_1}), \ldots, \sigma_{A_0}(\theta x_{i_m})\rangle : \begin{array}{l}\left\langle u_1, \ldots, u_{a(R)}\right\rangle \in db(R) \text{ such that} \\ \sigma_{A_0}(\theta \nu_j) =_s u_j \text{ for all } j = 1, \ldots, a(R)\end{array}\right\} \\
&= \left\{\langle \sigma_{A_0}(\theta x_{i_1}), \ldots, \sigma_{A_0}(\theta x_{i_m})\rangle : \begin{array}{l}\left\langle u_1, \ldots, u_{a(R)}\right\rangle \in db(R) \text{ such that} \\ \sigma_{A_0}(\theta x_{i_k}) =_s u_j \text{ whenever } j \in B_k\end{array}\right\} \\
&= \left\{\langle u_{b_1}, \ldots, u_{b_m}\rangle : \left\langle u_1, \ldots, u_{a(R)}\right\rangle \in db(R) \text{ and } p \in B_k \text{ implies } u_p =_s u_{b_k}\right\} \\
&= db(R \barwedge B)
\end{aligned}$$

and for the second claim

$$\begin{aligned}
[\![\phi]\!]^{\ell}_{db} &= \left\{\left\langle \sigma^{\ell}_{A_0}(\theta x_{i_1}), \ldots, \sigma^{\ell}_{A_0}(\theta x_{i_m})\right\rangle : \left\langle \sigma^{\ell}_{A_0}(\theta \nu_1), \ldots, \sigma^{\ell}_{A_0}(\theta \nu_{a(R)})\right\rangle \in db(R)\right\} \\
&= \left\{\langle u_{b_1}, \ldots, u_{b_m}\rangle : \begin{array}{l}\left\langle u_1, \ldots, u_{a(R)}\right\rangle \in db(R), |u_j| \leq \ell \text{ for all } j = 1, \ldots, a(R) \\ \text{and } p \in B_k \text{ implies } u_p =_s u_{b_k}\end{array}\right\} \\
&= db(R \barwedge B) \cap (\Sigma^{\leq \ell})^m.
\end{aligned}$$

Hence, choosing $E_\phi = R \barwedge B \cap (\Sigma^*)^m$ suffices for both claims.

For a string formula ϕ with variables $x_{i_1}, \ldots, x_{i_m}$ in ascending order, Theorem 3.1 provides (after temporarily renaming x_{i_j} to x_j) a corresponding m-FSA $\mathcal{A}_\phi$. Now

$$\begin{aligned}
[\![\phi]\!] &= L(\mathcal{A}_\phi) \\
&= (\Sigma^*)^m \cap L(\mathcal{A}_\phi) \\
&= db(\underbrace{\sigma_{\mathcal{A}}(\Sigma^*)^m}_{E_\phi})
\end{aligned}$$

yields the correct choice. The second claim holds as well since both $[\![\phi]\!]$ and $E_\phi \restriction \ell$ denote the restriction of $L(\mathcal{A}_\phi)$ to strings of length at most ℓ.

The construction of $E_{\phi\wedge\psi}$ proceeds as follows: let $x_{i_1}, \ldots, x_{i_m}$ be the variables appearing free in $\phi \wedge \psi$, $x_{j_1}, \ldots, x_{j_a}$ the ones in ϕ and $x_{k_1}, \ldots, x_{k_b}$ the ones in ψ, all ordered by ascending index. Build the partition B by putting into B_h the (one or two) positions where x_{i_h} appears in the ordered sequence $x_{j_1}, \ldots, x_{j_a}, x_{k_1}, \ldots, x_{k_b}$ and form $E_{\phi\wedge\psi} = (E_\phi \times E_\psi) \barwedge B$ where E_ϕ and E_ψ are obtained inductively. This is correct since

$$\begin{aligned}
[\![\phi \wedge \psi]\!]_{db} &= \{\langle \sigma_{A_0}(\theta x_{i_1}), \ldots, \sigma_{A_0}(\theta x_{i_m})\rangle : \langle A_0, db\rangle \models \phi\theta \text{ and } \langle A_0, db\rangle \models \psi\theta\} \\
&= \left\{\langle \sigma_{A_0}(\theta x_{i_1}), \ldots, \sigma_{A_0}(\theta x_{i_m})\rangle : \begin{array}{c}\langle \sigma_{A_0}(\theta x_{j_1}), \ldots, \sigma_{A_0}(\theta x_{j_a})\rangle \in [\![\phi]\!]_{db} \\ \text{and} \\ \langle \sigma_{A_0}(\theta x_{k_1}), \ldots, \sigma_{A_0}(\theta x_{k_b})\rangle \in [\![\psi]\!]_{db}\end{array}\right\} \\
&= \{\langle \sigma_{A_0}(\theta x_{i_1}), \ldots, \sigma_{A_0}(\theta x_{i_m})\rangle : \langle \sigma_{A_0}(\theta x_{j_1}), \ldots, \sigma_{A_0}(\theta x_{k_b})\rangle \in db(E_\phi \times E_\psi)\} \\
&= \left\{\langle u_{b_1}, \ldots, u_{b_m}\rangle : \begin{array}{l}\langle u_1, \ldots, u_{a+b}\rangle \in db(E_\phi \times E_\psi) \\ \text{and } j_p = k_q \text{ implies } u_p =_s u_q\end{array}\right\} \\
&= db((E_\phi \times E_\psi) \barwedge B).
\end{aligned}$$

Essentially the same calculation proves $[\![\phi \wedge \psi]\!]^\ell_{db} = db((E_\phi \upharpoonright \ell \times E_\psi \upharpoonright \ell) \barwedge B) = db(((E_\phi \times E_\psi) \barwedge B) \upharpoonright \ell)$ as well. It is also worth noting that we constructed here the *natural join* $E_\phi \bowtie E_\psi$ [24, pp. 59–60] by considering the ordered free variables as column attributes (with the addition that we also ordered the columns of the result as well).

For a negation $\neg\phi$ with free variables $x_{i_1}, \ldots, x_{i_m}$ in ascending order choose $E_{\neg\phi} = (\Sigma^*)^m \setminus E_\phi$ where E_ϕ is obtained inductively. By definitions $\langle u_1, \ldots, u_m\rangle \in [\![\neg\phi]\!]^\ell_{db}$ if and only if there exists a full interpretation A_0, θ such that if

$$\sigma_{A_0^\ell}(\theta x_{i_j}) = u_j \text{ for all } j = 1, \ldots, m \tag{5}$$

holds, $\left\langle A_0^\ell, db\right\rangle \models \phi\theta$ does not. This is in fact the case for *every* full interpretation A_0, θ satisfying Eq. (5) because the truth of ϕ is independent of $\sigma_{A_0^\ell}(\theta x_k)$ when x_k does not appear free in ϕ. Hence, equivalently $\langle u_1, \ldots, u_m\rangle \in (\Sigma^{\leq \ell})^m$ but $\langle u_1, \ldots, u_m\rangle \notin [\![\phi]\!]^\ell_{db} = db(E_\phi \upharpoonright \ell)$ by Eq. (5). That is $[\![\neg\phi]\!]^\ell_{db} = (\Sigma^{\leq\ell})^m \setminus [\![\phi]\!]^\ell_{db} = db(E_{\neg\phi} \upharpoonright \ell)$ by induction, proving the second claim. The first claim follows from the same reasoning by dropping the constant ℓ.

Consider finally the formula $\exists x_{i_1}.\phi$ where $x_{i_1}, \ldots, x_{i_m}$ are the variables appearing free in ϕ ordered by ascending index. (Restricting attention to the case where the quantification is over the first free variable results in no loss of generality.) If x_{i_1} does not appear in ϕ at all, then $\exists x_{i_1}.\phi$ is logically equivalent to ϕ and the inductively obtained E_ϕ suffices unmodified; otherwise calculate

$$\begin{aligned}
[\![\exists x_{i_1}.\phi]\!]_{db} &= \{\langle \sigma_{A_0}(\theta x_{i_2}), \ldots, \sigma_{A_0}(\theta x_{i_m})\rangle : \langle A_0, db\rangle \models \exists x_{i_1}.\phi\theta\} \\
&= \left\{\langle \sigma_{A_0}(\theta x_{i_2}), \ldots, \sigma_{A_0}(\theta x_{i_m})\rangle : \begin{array}{l} A_0 \text{ has a row } h \text{ such that} \\ \langle A_0, db\rangle \models \phi\theta[x_{i_1} = h] \end{array}\right\} \\
&= \left\{\langle \sigma_{A_0}(\theta x_{i_2}), \ldots, \sigma_{A_0}(\theta x_{i_m})\rangle : \begin{array}{l} A_0 \text{ has a row } h \text{ such that} \\ \langle \sigma_{A_0}(h), \sigma_{A_0}(\theta x_{i_2}), \ldots, \sigma_{A_0}(\theta x_{i_m})\rangle \in [\![\phi]\!]_{db} \end{array}\right\} \\
&= \{\langle u_2, \ldots, u_m\rangle : \langle u_1, \ldots, u_m\rangle \in db(E_\phi)\} \\
&= db(\underbrace{\pi_{[2,\ldots,m]} E_\phi}_{E_{\exists x_{i_1}.\phi}})
\end{aligned}$$

to see the correct choice. Essentially the same calculation yields

$$\begin{aligned}
[\![\exists x_{i_1}.\phi]\!]^\ell_{db} &= \left\{\langle u_2, \ldots, u_m\rangle : \left\langle \sigma_{A_0^\ell}(h), u_2, \ldots, u_m\right\rangle \in db(E_\phi \upharpoonright \ell)\right\} \\
&= db(E_{\exists x_{i_1}.\phi} \upharpoonright \ell)
\end{aligned}$$

as required for the second claim. □

The second claim of Theorem 3.5 shows that given an ℓ we can evaluate $[\![\phi]\!]^\ell_{db}$ finitely by considering each Σ^* in E_ϕ a generic symbol standing for $\Sigma^{\leq\ell}$. On the other hand, for any domain independent formula ϕ and database db there exist ℓ such that $[\![\phi]\!]_{db} = [\![\phi]\!]^\ell_{db}$, namely $\ell \geq \mathcal{W}_\phi(db)$ where $\mathcal{W}_phi\colon DB \to \mathbb{N}$ is a limit function for ϕ. Hence, we conclude that domain independent formulae of Alignment Calculus are finitely evaluable using Alignment Algebra with

$$[\![\phi]\!]_{db} = db(E_\phi \upharpoonright \mathcal{W}_\phi(db)). \tag{6}$$

That is, given any database db we first determine $\mathcal{W}_\phi(db)$ and then restrict attention to strings of at most that length when evaluating $db(E_\phi)$, making it a finitary operation.

Section 4 examines in more detail the problem of inferring whether $\mathcal{W}_\phi$ exists for a given query ϕ.

The above wrinkle that limit functions must be used stems from the fact noted before that Alignment Calculus queries must have the ability to *generate* new (and longer) strings not appearing in the database. An example is the domain independent (and hence also safe) query $x|\exists y, z\colon R_1(y) \wedge R_3(z) \wedge \psi(x, y, z)$, where ψ is the formula in Fig. 6. (see also Example 3 in Section 2). The query asks for the strings that are a concatenation of a string from $db(R_1)$ and another string from $db(R_3)$, as was mentioned before. Its translation to Alignment Algebra is literally

$$E = \pi_{[1]}\pi_{[1,2]}((R_1 \bar{\wedge} \langle\{1\}\rangle \times R_3 \bar{\wedge} \langle\{1\}\rangle) \bar{\wedge} \langle\{1\}, \{2\}\rangle \times \\ \sigma_{\mathcal{A}_\psi}(\Sigma^* \times \Sigma^* \times \Sigma^*)) \bar{\wedge} \langle\{3\}, \{1,4\}, \{2,5\}\rangle$$

which can be simplified into $\pi_{[1]}\sigma_{\mathcal{A}_\psi}(\Sigma^* \times R_1 \times R_3)$. Note how Σ^* is needed in the algebraic expression to implement the generation of these new strings. The function $\mathcal{W}(db) = \max(R_1, db) + \max(R_3, db)$, where 'max' was defined in Eq. (2), is evidently a limit function for this query. Therefore given any database db it is sufficient to evaluate

$$db(e \restriction \mathcal{W}(db)) = db(\pi_{[1]}\sigma_{\mathcal{A}_\psi}(\Sigma^{\leq \max(R_1,db)+\max(R_3,db)} \times R_1 \times R_3))$$

to yield the correct result, and this is indeed a finitary operation.

4 Safety Analysis

In the previous section we assumed that the alignment formulae were semantically safe. An important question is whether semantic safety can be syntactically determined. It is well known that safety is undecidable for purely relational formulae [24]. Thus there is no hope that safety would be decidable for alignment formulae. We shall, however, first look into the possibility of determining safety for string formulae alone, continuing the thread begun in Definition 3.1 above. It turns out that there is a source of undecidability in the string formulae, too.

We are interested in more than just determining whether the language defined by a string formula is finite. Consider the following two queries, where $x \in_s^* y$ stands for the string formula that says that x is a manifold of y (see Example 4 in Section 2).

$$y|\exists x.R(x) \wedge y \in_s^* x$$
$$y|\exists x.R(x) \wedge x \in_s^* y$$

The first query is unsafe, whereas the second one is safe. In neither case does a boundedness of y come from y having to appear in a database relation. However, in the second query x has to be a manifold of y, and thus y is at most as long as x. Furthermore, x has to appear in a database relation. The interesting observation is that "x limits y". Note that in the first query it is not true that "x limits y". As an additional example, in the query of the previous section asking for compositions of strings in relations r_1 and r_2 it is true that "y and z together limit x". In other words, we are interested in inferring the finiteness constraints $\phi\colon \{x_1, \ldots, x_k\} \rightsquigarrow \{x_{k+1}, \ldots, x_{k+l}\}$ proposed by Ramakrishnan et al [17]. Indeed, the limitation problem of Definition 3.1 is to determine these constraints for string formulae, so let us now examine it in more detail.

Theorem 4.1 *Given a string formula ϕ with an unidirectional variable x_1 and two bi-directional variables x_2, x_3, it is undecidable to determine if $\phi\colon \{x_1\} \rightsquigarrow \{x_2, x_3\}$ holds.*

Proof: Let $G = \left\{a_1^r \dots a_{s_r}^r \to b_1^r \dots b_{t_r}^r \colon r = 1, \dots, m\right\}$ be an *unrestricted grammar* [11, Chapter 9.2] with start symbol S and $\sharp$ a new separator not in G (or use a doubling technique to encode this extended alphabet into the original one). We construct a string formula $\phi_G(x_1, x_2, x_3)$ on alphabet Σ_G consisting of all the (terminal and nonterminal) symbols in G and $\sharp$ such that $[\![\phi_G]\!]$ consists of the tuples $\langle u, v_1\sharp \cdots \sharp v_n, w_1\sharp \cdots \sharp w_n\rangle$ where the condition

$$\underbrace{(n > 1) \wedge (u =_s v_1) \wedge (v_n =_s S) \wedge \left(\bigwedge_{i=1}^{n} v_i =_s w_i\right)}_{(1)} \wedge \underbrace{\bigwedge_{i=1}^{n-1} v_{i+1} \Rightarrow_G w_i}_{(2)}$$

holds. Note that Conditions (1) and (2) together are equivalent to $S \Rightarrow_G v_{n-1} \Rightarrow_G v_{n-2} \Rightarrow_G \dots \Rightarrow_G u$; that is, $v_1, \dots, v_n$ is a derivation of u in G.

Condition (1) actually denotes the tuples $\langle u, u\sharp u'S, u\sharp u'S\rangle$ where u does not contain '$\sharp$.' The formula

$$\begin{aligned}\phi_{(1)}(x_1, x_2, x_3){:}&[x_1, x_2, x_3]_l(x_1 = x_2 = x_3 \neq \sharp)^*.\\ &[x_1, x_2, x_3]_l(x_1 = \bot \wedge x_2 = x_3 = \sharp)^*.\\ &[x_2, x_3]_l(x_2 = x_3 \neg\bot)^*.[x_2, x_3]_l(x_2 = x_3 = \bot)\end{aligned}$$

has these as $[\![\phi_{(1)}]\!]$.

For the loop of Condition (2) we build a formula $\phi_{(2)}(x_2, x_3)$ as follows:

1. For every rule r, build the subformula $\psi_r\colon ([x_2]_l x_2 = a_1^r) \dots ([x_2]_l x_2 = a_{s_r}^r).([x_3]_l x_3 = b_1^r) \dots ([x_3]_l x_3 = b_{t_r}^r)$ to verify that columns $1, \dots, s_r$ of row θx_2 contain $a_1^r \dots a_{s_r}^r$ by moving the window to column s_r and similarly that $b_1^r \dots b_{t_r}^r$ appears to the right of the window in row θx_3. Note that this is the only place dependent on G.

2. Build from these the subformula

$$\psi_G\colon ([x_2, x_3]_l x_2 = x_3 \neq \sharp)^*.(\psi_1 + \ldots + \psi_m).([x_2, x_3]_l x_2 = x_3 \neq \sharp)^*$$

   to verify the condition $v_{i+1} \Rightarrow_G w_i$ on the assumption that rows θx_2 and θx_3 contain sentential forms v_{i+1} and w_i to the right of their initial window columns.

3. Finally $\phi_{(2)}$ is $[x_2]_l x_2 \neq \sharp)^*.[x_2]_l x_2 = \sharp.(\psi_G.[x_2, x_3]_l x_2 = x_3 = \sharp)^*.\psi_G.[x_2, x_3]_l x_2 = \bot \wedge x_3 = \sharp$, which repeats ψ_G for each '$\sharp$'-separated segment keeping the window column of row θx_2 in segment to the right of the segment where the window column of row θx_3 currently resides. Note that x_1 does not appear in this subformula.

The whole ϕ_G is then

$$\phi_{(1)}.\underbrace{([x_2, x_3]_r x_2 = x_3 \neq \bot)^*.([x_2, x_3]_r x_2 = x_3 = \bot)}_{(\star)}.\phi_{(2)} \tag{7}$$

where the subformula ($\star$), which also contains the only right transposes of the whole formula, serves only to reset rows θx_2 and θx_3 into initial alignment (row θx_1 need not be reset since $\phi_{(2)}$ does not refer to it).

Now asking whether x_1 limits x_2 and x_3 in ϕ_G is equivalent to asking if there exists a function f_G such that any sentential form u cannot be produced in more than $n = f_G(|u|)$ rule applications. We now prove this to be undecidable.

Let M be a Turing machine. We compute a grammar M_G which simulates it "backwards" in the sense that the language consists of the inputs to M and $v_{i+1} \Rightarrow_{M_G} v_i$ if and only if v_{i+1} is the next configuration of v_i (ignoring certain bookkeeping productions at the beginning and end of the parse):

- The initial rule is $S \to \rhd TqT \lhd$, where q is a state of M, '$\rhd$' is a new marker for the left end of the tape, and '$\lhd$' is a new marker for the left end of the area M has not visited yet. The nonterminal T generates an abitrary snippet of tape with $T \to aT$ and $T \to \varepsilon$ for every a in the tape alphabet Γ_M of M.
- The final rules are $\rhd q_0 \to F$, where q_0 is the start state of M, $Fa \to aF$ for each a in the input alphabet of M and $F \lhd \to \varepsilon$. That is, derivation is successful when q_0 is at the left end of the tape containing the input string.
- For every transition from state q to state p writing character Y on character X and moving the tape left, the corresponding grammar rule is $Yp \to qX$.
- In addition, for every transition from q to p symbol writing Y on the blank symbol and moving left, there is also the rule $Yp \lhd \to q \lhd$ for extending the tape snippet M has already used.
- For every transition from q to p writing Y on X and moving right, the rules are $pZY \to ZqX$ for every $Z \in \Gamma_M$.
- For every transition from q to p writing Y on blank and moving right, there are also the rules $pZY \lhd \to Zq \lhd$ for every $Z \in \Gamma_M$.

Now $[\![\phi_{M_G}]\!]$ contains exactly the tuples $\langle u, C, C\rangle$ where C is a partial computation of M on input u (together with the necessary initial and final rule applications). Hence, x_1 limits x_2 and x_3 exactly when for each u there exist only a finite number of such C; that is, when M is *total*. Totality of Turing machines is undecidable. $\square$

Far from everything is lost however. Theorem 4.1 makes use of the fact that rows of an alignment can be transposed both left and right. We believe that the right-restricted formulae introduced at the end of Section 2 suffice for practical queries; for them the limitation problem remains decidable. We shall also see in the next section that certain right-restricted formulae characterize the polynomial-time hierarchy.

Theorem 4.2 *The limitation problem is decidable for the right-restricted string formulae ϕ.*

Moreover, if the corresponding limitation function $\mathcal{W}_\phi$ exists, then

$$\mathcal{W}_\phi(n_1, \ldots, n_k) \le \ell \sum_{i=1}^{k} (n_i + 1) \tag{8}$$

if none of the tapes $1, \ldots, k$ *is bidirectional and*

$$\mathcal{W}_\phi(n_1, \ldots, n_k) \leq \ell(n_k + 1) \sum_{i=1}^{k-1} (n_i + 1) \tag{9}$$

if the bidirectional tape is k*, where* ℓ *is a constant depending on* ϕ*.*

Proof: By Theorem 3.1 we can equivalently solve the limitation problem for the corresponding FSA $\mathcal{A}_\phi$. Without loss of generality we may assume that $\mathcal{A}_\phi$ satisfies Properties (1) to (5) of Theorem 3.1. In addition, if $\mathcal{A}_\phi$ consists of just a rejecting start state, selecting $\mathcal{W}_\phi(n_1, \ldots, n_k) = 0$ suffices.

Consider $\mathcal{A}_\phi$ a generalized Mealy machine as discussed after Definition 3.1. There are exactly two ways in which $\mathcal{A}_\phi$ can break the limitation condition: the *easy* way is to accept without printing the trailing \$ onto some output tape, while the *hard* way is to generate output in a loop that consumes no input. To detect the latter case label each transition of $\mathcal{A}_\phi$ as *reading* if some unidirectional input tape is advanced by it, and *writing* if some unidirectional output tape is advanced by it. (Note that these two labels are independent; a transition may well carry both, neither or only one.) Then the limitation problem is solved by inspection of these labels. This inspection is quite simple for unidirectional FSAs; for bidirectional ones the solution is also strightforward but somewhat more tedious.

Consider first the simpler case where $\mathcal{A}_\phi$ is unidirectional. Properties (3) to (5) together imply that the accepting transitions are exactly the character combinations appearing under the tape heads in the accepting computations of $\mathcal{A}_\phi$. Hence, the easy way occurs exactly when they contain a combination where some output tape has a different character than \$. They also imply that the hard way occurs exactly when $\mathcal{A}_\phi$ contains a loop consisting only of nonreading transitions of which at least one is a writing transition. Both cases are clearly decidable. To see Eq. (8), note that the sum counts exactly the total number of reading operations possible, and by the absence of hard loops no writing but nonreading transition can be used twice without an intervening reading transition. Hence, ℓ can be chosen proportional to the maximum length of any writing but nonreading path in $\mathcal{A}_\phi$. To see that the limit can indeed be reached, consider the $(k+1)$-UFSA $\mathcal{B}_s$ with s states recognizing

$$\left\langle w_1, \ldots, w_k, \mathtt{a}^{s(|w_1|+\cdots+|w_k|+k)} \right\rangle \tag{10}$$

where the states $1, \ldots, s$ are organized in a ring in which the only transitions reading inputs $1, \ldots, k$ are the ones between s and 1 closing the circle, and those only read a single tape at a time. That is, $\mathcal{B}_s$ checks that the total number of possible reading moves is divisible by s. On every transition of this check a new character '$\mathtt{a}$' gets printed on the output tape $k+1$.

Let $\mathcal{A}_\phi$ then contain a bidirectional tape b as well. The problem in this case in that given an arbitrary path P of transitions through $\mathcal{A}_\phi$ we can no longer be certain that there actually is some valid computation traversing exactly it: P might make contradictory assumptions about different parts of the contents w_b of tape b. For example, the partial path of a 1-FSA $\langle \langle p_1, \mathtt{a} \rangle, \langle p_2, +1 \rangle \rangle, \langle \langle p_2, \mathtt{a} \rangle, \langle p_3, -1 \rangle \rangle, \langle \langle p_3, \mathtt{b} \rangle, \ldots \rangle$ looks innocent enough, but in fact has no suitable w_1 traversing it: the head is on the same square on the first and last transition, but their characters differ. Property (5) states, however, that if P indeed makes no such contradictory assumptions, then we can always find suitable contents for the unidirectional tapes so that $\mathcal{A}_\phi$ can indeed traverse P during a computation on them. In other words, we can safely disregard the unidirectional tapes of $\mathcal{A}_\phi$ to get a bidirectional 1-FSA which can naturally be reinterpreted as a *two-way nondeterministic finite automaton with endmarkers* $\mathcal{A}'_\phi$ [11, Chapter 2.6]. Then we can use an extension of the classical crossing sequence construction [11, Theorem 2.5 and Exercises 2.18–2.20] to get the corresponding nondeterministic finite automaton $\mathcal{A}''_\phi$, and in this way eliminate the paths that make contradictory assumptions from $\mathcal{A}_\phi$. Moreover, the construction below preserves enough information to reconstruct the interesting paths P from the structure of $\mathcal{A}''_\phi$.

We normalize $\mathcal{A}_\phi$ first in two ways without violating Property (5) and mark the changes so that they can be ignored later. (Or alternatively we could account for endmarkers and stationarity within the extended crossing sequence construction.)

First, we make $\mathcal{A}_\psi$ accept by attempting to wind tape b *past* its right endmarker \$. For this, create a new rejecting state $p_{\langle c_1,\ldots,c_{k+l}\rangle}$ where $c_i \in \Sigma \cup \{¢, \$\}$, and transitions from $p_{\langle c_1,\ldots,c_{b-1},c_b,c_{b+1},\ldots,c_{k+l}\rangle}$ to $p_{\langle c_1,\ldots,c_{b-1},c,c_{b+1},\ldots,c_{k+l}\rangle}$ on the character combination $c_1,\ldots,c_{k+l}$, where $c_b \neq \$$ and $c \in \Sigma \cup \{\$\}$. Moreover, the only tape to move is b and its direction is $+1$. These form the loops to wind tape b onto \$. Then replace every original accepting transition

$$\langle\langle p, c_1, \ldots, c_{b-1}, c_b, c_{b+1}, \ldots, c_{k+l}\rangle, \langle q, 0, \ldots, 0, 0, 0, \ldots, 0\rangle\rangle \tag{11}$$

with the corresponding entries into the loop, following the prescription given above, and mark them *cleanup*. Add also an exit where $c_b = \$$ which finally passes over the endmarker. If, however, $c_b = \$$ already in Eq. (11), it suffices to mark it and convert its winding 0 into $+1$ for tape b.

Second, get rid of each remaining transition t from p_1 to p_2 on $c_1,\ldots,c_{k+l}$ which does not move tape b as follows: Create a new state p_t and draw a transition from p_1 to p_t on $c_1,\ldots,c_{k+l}$ moving just tape b. The direction d_b is chosen to be -1 unless $c_b = ¢$. Then draw a path from p_t to p_2 on every combination $c_1,\ldots,c_{b-1},c,c_{b+1},\ldots,c_{k+l}$ where $c \in \Sigma \cup \{¢\}$, unless d_b was chosen to be $+1$ in which case $c \in \Sigma \cup \{\$\}$. The winding behaviour is taken from the original t, except that tape b is to be rewound in direction $-d_b$. Mark these transitions *dancing.*

Then define the concepts of our construction: a *crossing sequence* is a sequence of pairs $\langle p_i, d_i\rangle$ where p_i is a state of $\mathcal{A}_\phi$ and $d_i = \pm 1$ a direction (after the modifications above). Intuitively, a crossing sequence encodes the behaviour of $\mathcal{A}_\phi$ on the border between two adjacent squares s and $s+d$ of tape b: $\langle p, d\rangle$ denotes winding the tape from s to $s+d$ and entering state p for this visit on $s+d$ as a result of a transition of $\mathcal{A}_\phi$. A *match* is a sequence of transitions of $\mathcal{A}_\phi$. Intuitively, a match of two adjacent crossing sequences, the left one on the border between squares $s-1$ and s, and the right one between s and $s+1$, encodes any local behaviour which would cause these crossings to occur. In other words, it encodes the behaviour on s. Note that this *local* consistency does not yet ensure the existence of any computation actually performing it: such *global* consistency depends on the contents of the squares to the left of $s-1$ and to the right of $s+1$ as well.

Now we can define the relation *"the crossing sequence Q matches the crossing sequence P on character c by match T"*, denoted $m(Q;P;c;T)$ for brevity, inductively:

- $m(\varepsilon;\varepsilon;c;\varepsilon)$ holds, where 'ε' denotes the empty sequence and $c \in \Sigma \cup \{¢, \$\}$ is arbitrary.

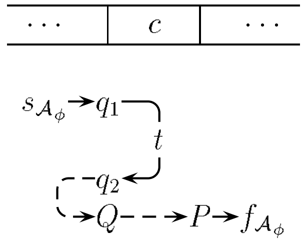


Figure 7: Reverting the direction of the head in the crossing sequence.

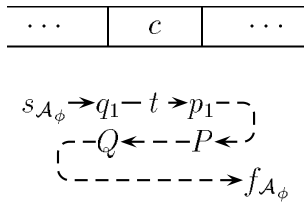


Figure 8: Continuing in the same direction in the crossing sequence.

- Assume that $m(Q; P; c; T)$ holds, where neither P nor Q begins with a pair with direction -1, and t is a transition of $\mathcal{A}_\phi$ from state q_1 to state q_2 on character c which winds tape b to direction '-1.' Then $m(\langle q_1, +1\rangle \langle q_2, -1\rangle Q; P; c; tT)$ also holds. Fig. 7 illustrates this case.

- By left-right symmetry we also have $m(Q; \langle p_1, -1\rangle \langle p_2, +1\rangle P; c; tT)$ when the common topmost direction (if any) of Q and P is -1, and t goes from p_1 to p_2 on c with direction $+1$.

- Assume that $m(Q; P; c; T)$ holds, where neither P nor Q begins with a pair with direction $+1$, and t is a transition of $\mathcal{A}_\phi$ from state q_1 to state p_1 on character c which winds tape b to direction '$+1$.' Then $m(\langle q_1, +1\rangle Q; \langle p_1, +1\rangle P; c; tT)$ also holds. Fig. 8 illustrates this case.

- By left-right symmetry we also have $m(\langle q_1, -1\rangle Q; \langle p_1, -1\rangle P; c; tT)$ when the common topmost direction (if any) of Q and P is $+1$, and t goes from p_1 to q_1 on c with direction -1.

Conversely, from any computation C of $\mathcal{A}_\phi$ one can (uniquely) reconstruct the corresponding crossing sequences and matches; call the result the *crossing picture* of C. It is easy to see that any crossing sequence occurring in the crossing picture of an accepting computation must be of the form $\langle p_1, +1\rangle \langle p_2, -1\rangle \langle p_1, +1\rangle \dots \langle p_{2m+1}, +1\rangle$: the head of tape b starts at the left end, stops at the right end and cannot jump over squares without visiting them. Call such sequences (of odd length, alternating directions and beginning with $+1$) *valid.* A crossing sequence is *direct* if no pair appears twice in it, and a computation with direct sequences only is also called direct. Similarly, a crossing sequence and computation is *almost* direct if no pair appears three times. Other sequences and computations are called *indirect.*

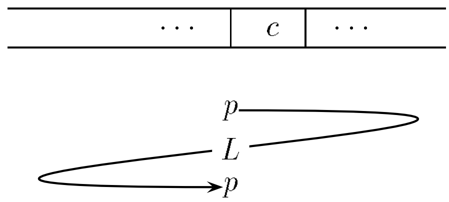


Figure 9: Looping in the crossing sequence.

Now $\mathcal{A}''_\phi$ can be finitely constructed as follows: The state space consists of the valid, almost direct crossing sequences. The start state $s_{\mathcal{A}''_\phi}$ is the sequence $\left\langle s_{\mathcal{A}_\phi}, +1 \right\rangle$ where $s_{\mathcal{A}_\phi}$ is the start state of $\mathcal{A}_\phi$. The only final state $f_{\mathcal{A}''_\phi}$ is $\left\langle f_{\mathcal{A}_\phi}, +1 \right\rangle$ where $f_{\mathcal{A}_\phi}$ is the final state of $\mathcal{A}_\phi$; There is a transition from Q to P on c *with matching label* T if and only if it holds that $m(Q; P; c; T)$. (Note that, due to nondeterminism and the disregarded unidirectional tapes, there can in general be several transitions between two states on the same character, each with a different matching label; we retain them all.) Then we can purge the states and transitions not on any path from $s_{\mathcal{A}''_\phi}$ to $f_{\mathcal{A}''_\phi}$.

The central observation is that $\mathcal{A}''_\phi$ *accepts the input* $u = u_1 u_2 u_3 \ldots u_n$ *by traversing the path* $R = r_0 m_0 r_1 m_1 r_2 m_2 \ldots r_{n+1} m_{n+1} r_{n+2}$*, where each* r_i *is a crossing sequence on the border of* $i-1$ *and* i *and* m_i *the match between* r_i *and* r_{i+1}*, if and only if* R *is a crossing picture for some almost direct accepting computation of* $\mathcal{A}_\phi$ *where tape* b *contains the input* u*:* Let $\mathcal{A}''_\phi$ accept u via R. By the construction of $\mathcal{A}''_\phi$, R is a crossing picture of an almost direct accepting computation of $\mathcal{A}'_\phi$ on u. By construction it also draws a path through the transition network of $\mathcal{A}_\phi$, and so by Property (5) there exist contents $w_1, \ldots, w_{b-1}, w_{b+1}, \ldots, w_{k+l}$ for which $\mathcal{A}_\phi$, after disregarding tape b, accepts via R. Now these two computations share a common path and their tapes are disjoint, so they can be naturally combined into an almost direct accepting computation on $\mathcal{A}_\phi$. Conversely, let R be the crossing picture of any almost direct accepting computation of $\mathcal{A}_\phi$ on inputs $\langle w_1, \ldots, w_{b-1}, u, w_{b+1}, \ldots, w_{k+l} \rangle$. Since the computation is accepting, its leftmost crossing sequence r_0 must be $s_{\mathcal{A}''_\phi}$ and rightmost r_{n+2} must be $f_{\mathcal{A}''_\phi}$. Each intermediate crossing sequence is almost direct by the assumption, and since the picture is about a computation, each pair of consecutive sequences r_i and r_{i+1} match by m_i. Hence, the path traversed is in $\mathcal{A}''_\phi$.

To paraphrase this observation, $\mathcal{A}''_\phi$ is a finitely constructible structure (labelled graph) encoding the path of every almost direct computation of $\mathcal{A}_\phi$. Hence, it can be used to answer finitely any limitation question about $\mathcal{A}_\phi$ by "graphical" considerations analogous to the unidirectional case above:

- "Does $\mathcal{A}_\phi$ contain an easy unidirectional output tape o?" Or, does $\mathcal{A}_\phi$ ever accept without printing the rightmost endmarker \$ on o?

  This happens if and only if there is a computation whose last transition has some other symbol for tape o. By the observation, the existence of such *almost direct* computation can be determined by checking the last transitions in the matching labels of any arc entering $f_{\mathcal{A}''_\phi}$.

  This suffices, since if there is such a computation, there is also an almost direct (indeed, even fully direct) one: we can *cut short* any indirect loop like L in Fig. 9 (and its symmetric variant) into simply a single occurrence of the repeating state-direction pair $\langle p, +1\rangle$ without removing any transitions of the cleanup phase of the computation. Repeating this transformation, we finally arrive at a crossing picture whose components can all be found in $\mathcal{A}''_\phi$ and by the observation a direct computation exists (but not necessarily with exactly the same unidirectional in- and output strings).

- "Is the bidirectional output tape b easy?" Or, does $\mathcal{A}_\phi$ ever accept without visiting \$ before the cleanup phase for any other reason than dancing?

  By the same cutting argument as above it suffices to examine the direct computations of $\mathcal{A}_\phi$; indeed, cutting can even remove such visits from a computation. Hence, it suffices to see if a matching label of an accepting arc contains dancing and cleanup transitions only.

- "Is the bidirectional output tape b hard?" Or, does there exist some input strings $u, v, w, u_1, \ldots, u_k \in \Sigma^*$ for which every output uv^jw is possible for the same unidirectional inputs $u_1, \ldots, u_k$?

  This v can be chosen so that the input tapes $1, \ldots, k$ are not read while the head is inside v, since otherwise $\mathcal{A}_\phi$ would eventually exhaust the inputs for large enough j. It is also sufficient to examine the direct computations of $\mathcal{A}_\phi$, since any indirect behaviour does not serve to advance the output production. Therefore, by the observation, the question reduces to determining if $\mathcal{A}''_\phi$ contains a loop whose arcs have no reading operations in their matching labels.

  This case also yields a partial result towards Eq. (8): if ℓ_1 is proportional to the maximum number of consecutive arcs not reading unidirectional input, then

$$\ell_1 \sum_{i=1}^{k}(n_i + 1) \tag{12}$$

  is an upper bound on the length of the bidirectional output on b.

- "Assuming tape b is the input tape k, does $\mathcal{A}_\phi$ contain a hard unidirectional output tape o?" Or, does there exist $u_1, \ldots, u_{k-1}, u, v, w \in \Sigma^*$ such that on inputs $\langle u_1, \ldots, u_{k-1}, uvw\rangle$ $\mathcal{A}_\phi$ can move tape b back and forth on v arbitrarily long generating output but not reading any unidirectional input from the tapes $1, \ldots, k-1$. This assumption holds if b is an input tape to begin with; otherwise it can be checked by performing the preceding case first.

  The corresponding crossing pictures look like Fig. 9, where L contains no reading transitions but at least one writing transition. Hence, it no longer suffices to examine direct computations only, but it still suffices to examine almost direct ones, because if any indirect loop as in Fig. 10 (and its symmetric variant) has this property, then at least one of the subloops L_1 and L_2 does as well.

  It still remains to show that it suffices to examine v of bounded length. Fig. 11 shows a crossing picture of an almost direct loop, where the match m between crossing sequences r and r' is repeated thrice on the same side of the almost direct

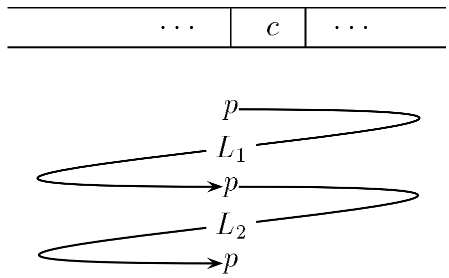


Figure 10: Double looping in the crossing sequence.

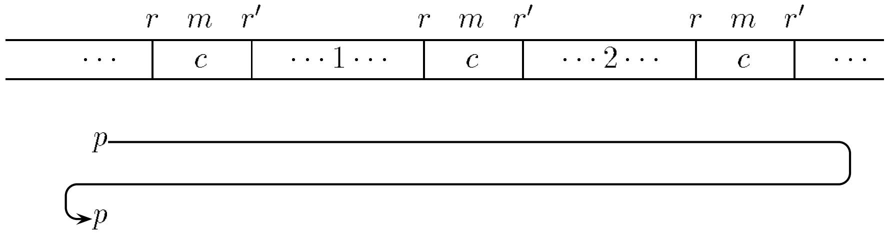


Figure 11: Repeating the match m.

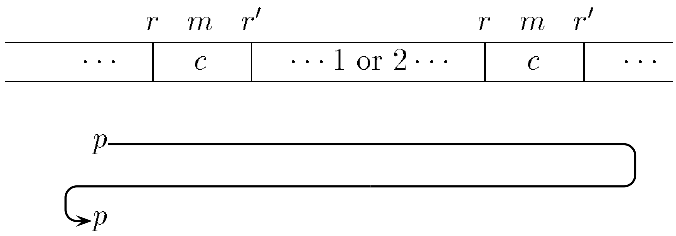


Figure 12: Eliminating one repeat of match m.

crossing sequence yielding the loop. If this loop has the property sought, then either of the subloops in Fig. 12 also does. (The symmetric case is identical.) Hence, it is sufficient to restrict attention to v with length no more than twice the total number of arcs in $\mathcal{A}''_\phi$.

To see Eq. (9), select ℓ to be the number of states in $\mathcal{A}_\phi$. Then the sum is the total number of read operations possible on the unidirectional input tapes $1, \ldots, k-1$ and for each such operation after at most $\ell(n_k + 1)$ steps $\mathcal{A}_\phi$ reenters the same state and square on the bidirectional tape k. To see that this bound can indeed be reached, consider the $(k+1)$-FSA $\mathcal{B}'_s$ with s states, where s is even, recognizing

$$\left\langle w_1, \ldots, w_k, \mathtt{a}^{s(|w_k|+1)(|w_1|+\cdots+|w_{k-1}|+k-1)} \right\rangle$$

which can be obtained from the $\mathcal{B}_s$ of Eq. (10) by adding a loop to each odd-numbered state that winds k from ¢ all the way to \$, and rewinding it back in each even-numbered state.

To see the sharper bound of Eq. (8), recall the bound of Eq. (12) for the length of the bidirectional output tape and choose ℓ_2 to be the maximum length of any matching label in $\mathcal{A}''_\phi$. (Note, however, that this ℓ_2 can be exponential with respect to the number of states in $\mathcal{A}_\phi$, while in the unidirectional case ℓ could be selected linear.) By the crossing sequence construction, for every character output on the bidirectional tape there are at most ℓ_2 output operations for the unidirectional ones, so choosing $\ell = \ell_1 \ell_2$ yields the result. The bound is tight (up to the size of ℓ) by considering the output of the $\mathcal{B}_s$ in Eq. (10) bidirectional. $\square$

The solution to the limitation problem represents a first step towards a syntactic characterization of safety and an implementation of Alignment Calculus. However, we also need a constructive method for expanding the safety, and limit functions $\mathcal{W}_\psi$, of the embedded string formulae ψ into the safety, and limit function $\mathcal{W}_\phi$, of the whole Alignment Calculus query ϕ in which they appear, if we are to evaluate $[\![\phi]\!]_{db}$ in the manner put forth in Theorem 3.5; one such method was developed in [3]. Note also that not all string formulae of a domain independent Alignment Calculus formula need to be right-restricted, provided that only right-restricted string formulae are used for generating new strings while the others are used just for examining them.

5 Expressive power

In this section we study the definitional power of pure Alignment Calculus. Furthermore, we assume for simplicity that x_1 is the only free variable. We then say that ϕ *defines* the language $[\![\phi]\!]$. By the data complexity [25] of a set defined by a formula ϕ we mean the complexity of the decision problem "$w \in [\![\phi]\!]$?" measured as a function of $|w|$; expression complexity is in turn measured as a function of the length $|\phi|$ of (that is, the number of symbol occurrences in) the formula ϕ.

Theorem 5.1 *The unidirectional unquantified string formulae represent exactly the regular languages.*

Proof: Any regular expression A can be translated into a formula $\phi_A.[\,]_l x_1 = \bot$ where ϕ_A is obtained from A by replacing every occurrence of $c \in \Sigma$ with $[x_1]_l x_1 = c$. Now

$$L(\phi_A) = \{[x_1]_l x_1 = c_1 \dots [x_1]_l x_1 = c_n : c_1 \dots c_n \in L(A)\}$$

and hence a formula word in $L(\phi_A.[\,]_l x_1 = \bot)$ is true exactly in the initial alignments where $\sigma_{A_0}(\theta x_1) \in L(A)$.

Conversely, given any such formula ϕ Theorem 3.1 yields the corresponding 1-FSA $\mathcal{A}_\phi$ which is a nondeterministic finite automaton with endmarkers. Hence, the language it accepts is regular. □

Since moving the only tape back and forth does not increase expressivity (as proved implicitly in Theorem 4.2) and since the regular languages are closed under conjunction and intersection, Theorem 5.1 readily generalizes to all unquantified formulae of one variable.

Theorem 5.2 *The formulae $\exists x_2, x_3.\phi$, where ϕ is a string formula with an unidirectional variable x_1 and two bidirectional variables x_2, x_3, represent exactly the recursively enumerable languages.*

Proof: Any recursively enumerable language can be represented with an unrestriced grammar G, and in Theorem 4.1 we constructed the corresponding string formula ϕ_G of the required type. Now $\exists x_2, x_3.\phi_G$ is true if and only if x_1 can be derived by G.

Conversely, Theorem 3.1 provides a 3-FSA $\mathcal{A}_\phi$ corresponding to ϕ. As checking whether $\mathcal{A}_\phi$ accepts a given input $\langle u, v, w\rangle$ or not is decidable by Theorem 3.3, the search for such v and w for a given u is recursively enumerable. □

Theorem 5.2 implies that the abstract definitional power of pure Alignment Calculus equals the *Arithmetical Hierarchy* [19, Chapter 14] (on strings instead of $\mathbb{N}$). In fact, even a stronger version of Theorem 5.2 can be given, which also justifies the layered structure of our language.

Corollary 5.1 *The formulae $\exists x_2, x_3.\phi \wedge \psi$, where ϕ and ψ are unidirectional string formulae on variables x_1, x_2, x_3 represent exactly the recursively enumerable languages. In fact, it even suffices to choose a ψ without x_1.*

Proof: Examine Eq. (7) in Theorem 4.1. Its subformula ($\star$) serves only to rewind x_2 and x_3 after the truth of $\phi_{(1)}$ is verified in order to verify the truth of $\phi_{(2)}$ which in turn does not contain x_1. Hence, the same effect can also be obtained with a conjunction like in Example 9 of Section 2 as well. □

A different approach to computational aspects of sequence queries was taken in [15], which extended classical Datalog [24, Chapter 3.2] with new interpreted function symbols for manipulating sequences. In that approach, computability is attained via explicit recursion, while in ours it surfaces as a by-product of an expressive language for describing string relations.

The following result follows from the fact that extended temporal logic has the expressive power of regular expressions [2, Theorem 6.5 and Chapter 6.6.1].

Theorem 5.3 *Every set definable in extended temporal logic is definable in pure Alignment Calculus. There are sets definable in pure Alignment Calculus that are not definable in extended temporal logic.*

Proof: Theorem 5.1 established the former, Theorem 5.2 the latter claim. □

For those programmers who are comfortable reasoning with temporal modalities, our logic offers the possibility of regarding a transpose as going into the future (left) or the past (right) in those linear time structures (rows) that are mentioned in the transpose. The following conventions can then be used.

$$\begin{array}{lll}
\textit{next along } x_1,\ldots,x_k\ \phi & \equiv_{df} & [x_1,\ldots,x_k]_l\phi \\
\phi \textit{ along } x_1,\ldots,x_k \textit{ until } \psi & \equiv_{df} & ([x_1,\ldots,x_k]_l\phi)^*.([x_1,\ldots,x_k]_l\psi) \\
\textit{eventually along } x_1,\ldots,x_k\ \phi & \equiv_{df} & ([x_1,\ldots,x_k]_l\top)^*.([x_1,\ldots,x_k]_l\phi) \\
\textit{henceforth along } x_1,\ldots,x_k\ \phi & \equiv_{df} & ([x_1,\ldots,x_k]_l\phi)^*. \\
 & & ([x_1,\ldots,x_k]_l x_1 = \ldots = x_k = \bot)
\end{array}$$

For instance, the two-place predicate "x *occurs in* y" (see the seventh sample query in Section 2) could be formulated as

$$\textit{eventually along } y\ (x = y \textit{ along } x, y \textit{ until } x_\bot).$$

The modalities for past tense are obtained by using right transposes instead of the left ones in the list above, e.g.

$$\phi \textit{ along } x_1,\ldots,x_k \textit{ since } \psi \equiv_{df} ([x_1,\ldots,x_k]_r\phi)^*.([x_1,\ldots,x_k]_r\psi).$$

Ginsburg and Wang [7, 26] have proposed an extension of Relational Calculus for dealing with *sequences* or lists of items drawn from an infinite alphabet $\mathcal{U}$ of atomic constants (or atoms for brevity). For example, [Peter,Paul,Mary] is a sequence of atoms 'Peter', 'Paul', and 'Mary'. Relations are in turn defined over these sequences. Sequences are manipulated by *sequence predicates* $x_{n+1} \in [\![A]\!](x_1,\ldots,x_n)$, where A is a regular expression over the alphabet $\{\alpha_1,\ldots,\alpha_n\}$ and the variables $x_1,\ldots,x_{n+1}$ range over sequences. Such a predicate occurrence is true when there exists a $\beta \in L(A)$ for which the following two properties hold: first, β contains exactly as many occurrences of α_i as the length of (the number of atom occurrences in) the sequence (currently assigned to) x_i; and second, the j'th occurrence of α_i in β is in position m if and only if position m of x_{n+1} contains the same atom as the j'th position of x_i. Operationally, A represents a pattern in which items are copied from *channels* $x_1,\ldots,x_n$ into x_{n+1} until all the inputs have been exhausted. Intuitively x_{n+1} might be described as a "regular shuffle" of $x_1,\ldots,x_n$. For example $x_3 \in [\![\alpha_1^*\alpha_2^*]\!](x_1,x_2)$ declares x_3 to be the concatenation of x_1 and x_2. Details can be found in [26, Chapter 2.1].

To compare the power of sequence predicates in the infinite $\mathcal{U}$ and string formulae in the finite Σ we choose a translation injection $e{:}\,\mathcal{U} \to \Sigma^*$ mapping atoms to strings. Given also a new terminator symbol $\sharp \notin \Sigma$ we can extend e to sequences with $e([a_1,\ldots,a_m]) = e(a_1)\sharp\ldots\sharp e(a_m)\sharp$. With these tools we obtain the following result:

Theorem 5.4 *For every sequence predicate $P{:}\ x_{n+1} \in [\![A]\!](x_1,\ldots,x_n)$, where $x_1,\ldots,x_{n+1}$ are distinct variables, there exists an unidirectional string formula ϕ_P on $x_1,\ldots,x_{n+1}$ such that P is true in the assignment mapping x_i to s_i for $i = 1,\ldots,n+1$ if and only if $\langle e(s_1),\ldots,e(s_{n+1})\rangle \in [\![\phi_P]\!]$.*

Proof: Thinking operationally, we build the string formula ϕ_A by replacing each α_i in A with the subformula $([x_j,x_{n+1}]_l x_{n+1} = x_j \neq \sharp)^*.[x_j,x_{n+1}]_l x_{n+1} = x_j = \sharp$ which implements the copying of the next translated atom from channel i to channel $n+1$. The final ϕ_P is then $\phi_A.[x_1, \ldots, x_{n+1}]_l x_1 = \ldots = x_{n+1} = \bot$. □

The full calculus of Ginsburg and Wang allows sequence constants and repeating variables among the arguments $x_1, \ldots, x_{n+1}$. Constants can be embedded into ϕ_P by Lemma 3.1 (via Theorems 3.1 and 3.2). Repetitions can be eliminated by rewriting each $\phi(\ldots, x_i, \ldots, x_i, \ldots)$ into $\exists y_i.(x_i =_s y_i \wedge \phi(\ldots, x_i, \ldots, y_i, \ldots))$ before applying Theorem 5.4 although the result is no longer just a string formula.

We note that the programming paradigms underlying our Alignment Calculus and the logic Ginsburg and Wang propose are different. The basic string processing construct in sequence logic is the transducer mapping. Since regular sets are closed under transducer mappings [6], computational power is obtained by resorting to features outside the string processing machinery, such as first order logic. In Alignment Calculus, on the other hand, full Turing computability is available already when using unquantified string formulae on three variables and a projection operator, as in Theorem 5.2. As pointed out by Ginsburg and Wang [7], it is of course always possible to increase the expressive power of sequence logic by strengthening the language used to specify the transducers; this approach was taken by Mecca and Bonner [15] whose sequence manipulation symbols represent generalized transducers capable of invoking other transducers as subroutines. However, in Alignment Calculus it is not necessary to go beyond regular expressions in the syntax of the string formulae, or to employ complex computational devices.

We have also been able to obtain a correspondence between a subclass of pure Alignment Calculus and the polynomial-time hierarchy [23]. For this correspondence we need the class of *quantifier limited* formulae. A formula ϕ is quantifier limited by its free variables $x_1, \ldots, x_m$ if

- ϕ is a string formula, possibly negated; or
- ϕ is of the form $\psi \wedge \chi$ or $\psi \vee \chi$ where ψ and χ are in turn quantifier limited by $x_1, \ldots, x_m$; or
- ϕ is of the form $\exists y_1, \ldots, y_n(\psi \wedge \chi)$ or $\forall y_1, \ldots, y_n(\psi \rightarrow \chi)$ where χ is quantifier limited by its free variables $x_1, \ldots, x_m, y_1, \ldots, y_n$ and ψ is a right-restricted string formula for which the limitation condition $\psi\colon \{x_1, \ldots, x_m\} \rightsquigarrow \{y_1, \ldots, y_n\}$ holds.

  The reader is asked to recall from the end of Section 2 that right-restriction in string formulae means that at most one variable is bidirectional, that is, transposed both left and right.

The intuition behind this formula class is that each quantifier is limited by a "type qualifier" ψ, and that this limitation remains decidable.

Theorem 5.5 *Let $\exists y_1, \ldots, y_n \phi$ ($\forall y_1, \ldots, y_n \phi$, respectively) be quantifier limited by its free variable x_1 and contain $k-1$ quantifier alternations. Then it defines a set in Σ_k^p (in Π_k^p, respectively). Furthermore, for each $k > 0$ there is such a formula that defines a Σ_k^p-hard (a Π_k^p-hard, respectively) set.*

Proof: The first claim is proved by a straightforward double induction as follows. Examine the formula $\exists y_1, \ldots, y_n \phi$ which is quantifier limited by its free variables $x_1, \ldots, x_m$ and alternates $k-1$ times. By assumption ϕ is of the form $\psi \wedge \chi$ where ψ is a right-restricted string formula satisfying $\psi\colon \{x_1, \ldots, x_m\} \rightsquigarrow \{y_1, \ldots, y_n\}$ and χ is obtained by combining

- possibly negated string formulae;

- subformulae $\exists z_1, \ldots, z_p \phi'$ quantifier limited by $x_1, \ldots, x_m, y_1, \ldots, y_n$ alternating at most $k-1$ times; and

- subformulae $\forall z_1, \ldots, z_p \phi'$ quantifier limited by $x_1, \ldots, x_m, y_1, \ldots, y_n$ alternating at most $k-2$ times

with conjunctions and disjunctions. By Theorems 3.1 and 4.2 there exists a polynomial q such that if $A_0 \models \psi\theta$, then $|\sigma_{A_0}(\theta y_i)| \leq q(|\sigma_{A_0}(\theta x_1)|, \ldots, |\sigma_{A_0}(\theta x_m)|)$ for all $i = 1, \ldots, n$. Hence, the leading quantifier can be considered polynomial without affecting the relation defined by the whole formula. It remains to be shown that the relation defined by the subformula ϕ is in Σ_k^p. By Theorems 3.1 and 3.3 the relations defined by string formulae belong to $\mathbf{P} = \Sigma_0^p = \Pi_0^p$, and since this class is closed under complement, so do the relations defined by negations of string formulae as well. The subformulae $\exists z_1, \ldots, z_p \phi'$ define relations in Σ_k^p by induction on formula complexity. The subformulae $\forall z_1, \ldots, z_p \phi'$ define relations in $\Pi_{k-1}^p \subseteq \Sigma_k^p$ by induction on k. Finally Σ_k^p is closed under intersection and union and the claim follows. The case $\forall y_1, \ldots, y_n \phi$ is similar. Note, however, that by Corollary 5.1 the definition of quantifier limited formulae cannot be weakened to allow more complex formulae ψ satisfying $\psi \colon \{x_1, \ldots, x_m\} \rightsquigarrow \{y_1, \ldots, y_n\}$ as type qualifiers, because in that case the function q need no longer be polynomial.

The existence of Σ_k^p and Π_k^p complete sets is proved by exhibiting for each k a formula of the given type checking the truth of any Quantified Boolean Formula (QBF)

$$Q^k X_1^k \cdots Q^k X_{m_k}^k \cdots Q^1 X_1^1 \cdots Q^1 X_{m_1}^1 \Phi$$

given as the input x_1, where the propositional formula Φ on $X_1^1, \ldots, X_{m_k}^k$ is in the proper normal form: conjunctive (CNF) if $Q^1 = \exists$ and disjunctive (DNF) otherwise [23, Theorem 4.1 (2)]. Without loss of generality we can also assume that the QBF formula has no free variables, that each propositional variable X_j^i is textually represented by a nonempty binary index string denoted $I(i,j)$, and that these indices occur in strictly ascending canonical order in the quantifier prefix.

Define for every $k \in \mathbb{N}$ the following FSAs:

- M_k is an unidirectional 2-FSA verifying that if the formula encoded on tape 1 alternates $k-1$ times, then tape 2 contains a string in the language $\{\mathcal{T}, \mathcal{F}\}^*$ whose length equals the number of variables in the k'th alternation block of the formula. (If it alternates less, tape 2 is to be empty.) The limitation property $\{1\} \rightsquigarrow \{2\}$ holds for M_k.

- M^k is an unidirectional $(2+k)$-FSA verifying that for the quantifier prefix

$$Q^k I(k,1) \cdots Q^k I(k,m_k) \cdots Q^1 I(1,1) \cdots Q^1 I(1,m_1)$$

of the formula encoded on tape 1, tape 2 contains

$$I(k,1) T_1^k \cdots I(k,m_k) T_{m_k}^k \cdots I(1,1) T_1^1 \cdots I(1,m_1) T_{m_1}^1$$

where tape $2+i$ contains exactly $T_1^i \ldots T_{m_i}^i \in \{\mathcal{T}, \mathcal{F}\}^*$. (If the prefix alternates less than $k-1$ times, the unused tapes are verified to be empty; if more, the missing values are assumed arbitrarily to be $\mathcal{T}$.)

Intuitively, tape $2+i$ contains the truth values assigned to the variables in the $(i-1)$'th alternation block

$$Q^{k+1-i}I(k+1-i,1)\cdots Q^{k+1-i}I(k+1-i,m_{k+1-i}),$$

and tape 2 contains a truth value distribution for the propositional part of the formula encoded on tape 1. Note, however, unless the encoded formula respects the restriction on the order of variable indices $I(i,j)$, the distribution may have multiple conflicting values for a given index.

The limitation property $\{1\} \rightsquigarrow \{2,\ldots,2+k\}$ holds for M^k.

- $M^k_\exists$ is a right-restricted 2-FSA, whose bidirectional tape is 2, verifying the following four conditions:

  That the quantifier prefix of the formula encoded on tape 1 indeed respects the restriction on the order of variable indices $I(i,j)$. Assuming tape 2 contains a copy of the indices, separated by $\mathcal{T}$ or $\mathcal{F}$ as in those tapes accepted by M^k on the same formula, this can be verified in one pass over the prefix by keeping the head of tape 1 on the index following the index under the head of tape 2. (The same trick was utilized in constructing $\phi_{(2)}$ for Theorem 4.1.) That is, if $M^k(u,v,w_1,\ldots,w_k)$ accepts first, then $M^k_\exists(u,v)$ accepts only if u respects the ordering, and in that case v contains a valid truth table row without multiple entries per variable.

  That the prefix begins with '$\exists$' and alternates exactly $k-1$ times, checkable within the finite control concurrently with the preceding condition.

  That, once the prefix is checked, each variable index appearing in the propositional part on tape 1 appears also on tape 2. This can be performed by nondeterministically guessing an index on tape 2, then verifying that it matches the one next on tape 1. In other words, tape 1 is consumed in order while tape 2 is used as "random-access read-only memory". This guess involves rewinding tape 2, but eliminates the need for rewinding tape 1. After a successful check the value given to this variable by this truth value distribution can also be extracted.

  And finally, that this propositional part is of the correct normal form (that is, conjunctive if and only if k is odd) and true under the truth value distribution on tape 2. By the simple structure of the normal form(s), this can be checked within the finite control concurrently with the preceding condition.

- $M^k_\forall$ which is as above, except that the prefix must start with '$\forall$' instead. In addition, here the propositional part is required to be in CNF if and only if k is even.

By Theorem 3.2 there exist the corresponding string formulae $\phi_i(x_1,z_i)$ (in which $1 \leq i \leq k$), $\phi^k(x_1,y,z_1,\ldots,z_k)$, $\phi^k_\exists(x_1,y)$ and $\phi^k_\forall(x_1,y)$. With them we can give the desired formula for every level of the polynomial-time hierarchy, except $\Sigma^p_0 = \Pi^p_0 = \mathbf{P}$, as follows.

- For Σ^p_{2k-1} the formula is

$$\begin{array}{l}\exists z_{2k-1}.\phi_{2k-1}(x_1, z_{2k-1})\wedge\\ \forall z_{2k-2}.\phi_{2k-2}(x_1, z_{2k-2}) \to\\ \quad\vdots\\ \quad\exists z_1.\phi_1(x_1, z_1)\wedge\\ \quad\exists y.\phi^{2k-1}(x_1, y, z_1, \ldots, z_{2k-1})\wedge\\ \quad\quad\phi^{2k-1}_{\exists}(x_1, y).\end{array}$$

  The conjunction is true if and only if there exists a y concatenating the current z_i into a truth value distribution mirroring the structure of the quantifier prefix of x_1 and making its propositional part true. Note that $\phi^{2k-1}_{\exists}(x_1, y)$ also verifies that x_1 indeed encodes a legal QBF formula.

  The limitation condition $\{x_1\} \leadsto \{y, z_1, \ldots, z_{2k-1}\}$ holds for ϕ^{2k-1} as it is equivalent to M^{2k-1}, and $\{x_1\} \leadsto \{z_i\}$ holds for ϕ_i as they are equivalent to M_i where $i = 1, \ldots, 2k-1$. Hence the formula is easily seen to be quantifier limited.

- For Π^p_{2k} the corresponding formula is

$$\begin{array}{l}\forall z_{2k}.\phi_{2k}(x_1, z_{2k}) \to\\ \exists z_{2k-1}.\phi_{2k-1}(x_1, z_{2k-1})\wedge\\ \quad\vdots\\ \quad\exists z_1.\phi_1(x_1, z_1)\wedge\\ \quad\exists y.\phi^{2k}(x_1, y, z_1, \ldots, z_{2k})\wedge\\ \quad\quad\phi^{2k}_{\forall}(x_1, y).\end{array}$$

  which can be justified as above.

- For Σ^p_{2k} the formula is

$$\begin{array}{l}\exists z_{2k}.\phi_{2k}(x_1, z_{2k})\wedge\\ \forall z_{2k-1}.\phi_{2k-1}(x_1, z_{2k-1}) \to\\ \quad\vdots\\ \quad\forall z_1.\phi_1(x_1, z_1) \to\\ \quad\forall y.\phi^{2k-1}(x_1, y, z_1, \ldots, z_{2k-1}) \to\\ \quad\quad\phi^{2k-1}_{\exists}(x_1, y).\end{array}$$

  That is, for every y concatenating the current z_i into a truth table row, the propositional part of x_1 is also true on it.

- For Π^p_{2k-1} the corresponding formula is

$$\begin{array}{l}\forall z_{2k-1}.\phi_{2k-1}(x_1, z_{2k-1}) \to\\ \exists z_{2k-2}.\phi_{2k}(x_1, z_{2k})\wedge\\ \quad\vdots\\ \quad\forall z_1.\phi_1(x_1, z_1) \to\\ \quad\forall y.\phi^{2k-1}(x_1, y, z_1, \ldots, z_{2k-1}) \to\\ \quad\quad\phi^{2k-1}_{\forall}(x_1, y).\end{array}$$

  which can be justified as above.

Hence, the latter claim also holds. $\square$

Let us now turn from data to expression complexity. Here the queries are closed formulae and as such do not define languages; instead, the problem is to determine whether they define the full or empty relation of arity 0. In other words, we must examine the *decidability* of Alignment Calculus.

We know from Theorem 5.2 that two bidirectional variables in a string formula suffice to make expression complexity undecidable. Furthermore, Corollary 5.1 tells us the same thing happens in conjunctions of two unidirectional string formulae with at least two variables. The case of just one variable is decidable by Theorem 5.1. It remains then to examine the case of right-restricted string formulae.

Theorem 5.6 *Let the alphabet Σ be fixed beforehand. Then for each $k > 0$ the expression complexity of the queries ψ of the form $\exists x_1, \ldots, x_k \phi$, where ϕ is a right-restricted string formula on variables $x_1, \ldots, x_k$, is* **PSPACE**-*complete.*

Proof: First, note that *unidirectional quantifier elimination* is possible in quantified string formulae such as the ones above: as an example consider $\exists x.\psi$, where ψ is a string formula in which x is unidirectional. Form by Theorem 3.1 a corresponding FSA $\mathcal{A}_\psi$, and let $\mathcal{A}'_\psi$ be the FSA obtained by omitting the tape corresponding to x. Now the string formula $\phi_{\mathcal{A}'_\psi}$ obtained from Theorem 3.2 is equivalent to the original one by Property (5). We omit here the last step of this transformation, however, to obey the stated space bound.

Because for each fixed k the k-FSA $\mathcal{A}_\phi$ corresponding to ϕ by Theorem 3.1 can be produced in space $O(|\Sigma|^k |\phi|)$, the problem reduces to the *nonemptiness problem for right-restricted k-FSAs*.

For $k = 1$ this nonemptiness problem equals the acceptance problem for the "nondeterministic checking stack automata with an empty input" which is known to be **PSPACE**-complete [5, Problem AL5]. For $k > 1$ the construction of $\mathcal{A}_\phi$ to satisfy Property (5) performs simultaneously the reduction of this case into the case $k = 1$ of the nonemptiness problem. (Indeed, as this construction can be carried out in space $O(|\Sigma|^k |\mathcal{A}|)$ for any k-FSA $\mathcal{A}$, we also see that for all $k > 0$ the nonemptiness problem is **PSPACE**-complete.) Hence, the expression complexity of the queries ψ is indeed in **PSPACE**.

This does not, however, suffice for **PSPACE**-completeness: the k-FSAs $\mathcal{A}_\phi$ arising from string formulae ϕ have a far more uniform structure than arbitrary k-FSAs $\mathcal{A}$. Therefore we reduce the *acceptance problem for Linear Bounded Automata (LBAs)* [11, Chapter 9.3] to the truth of formulae ψ: $\exists x_1 \phi$. (We essentially follow the construction of Theorem 4.1 for the *Context-Sensitive Grammars (CSGs)* [11, Chapter 9.3]. As the length of the strings in CSG derivations in nondecreasing we can do without the second quantifier; this, however, is more straightforward to express using LBAs.) Let M be an LBA with

- alphabet Γ, including both the input and tape alphabets;
- left and right endmarkers $\triangleright$ and $\triangleleft$, respectively;
- states $p_0, \ldots, p_m$ where p_0 is the start state and p_m the final state which has no outgoing transitions; and
- input string $I = c_1 \ldots c_n$.

We write a string formula ϕ on alphabet $\Gamma' = \Gamma \cup \{\rhd, \lhd, p_0, \ldots, p_m\}$ such that $|\phi| = O(nt\,|\Gamma|)$, where t denotes the number of transitions in M. This formula ϕ will be true if and only if the value of x_1 is (an encoding of) an accepting computation of M on I.

1. The subformula $\psi(n, a, b)$: $[\,]_l x_1 = a.([x_1]_l x_1 \neq \bot)^{n+1}.[x_1]_l x_1 = b.([x_1]_r \top)^{n+1}$ verifies that the current position contains the symbol a, that the $(n+2)$nd position to its right contains the symbol b, and returns to the right neighbour of a. In other words, $\psi(n, a, b)$ verifies that the next configuration has a b in the same place as this one has the current a.

2. Examine any transition of M from state p_i on character $a \in \Gamma$ which writes character $b \in \Gamma$ over a and moves the head right and enters state p_j. This behaviour can be encoded as context-sensitive rules $dp_ia \rightarrow dbp_j$ where $d \in \Gamma \cup \{\rhd, \lhd\}$. Each such rule r can in turn be expressed with a subformula χ_r: $\psi(n, d, d).\psi(n, p_i, b).\psi(n, a, p_j)$. Similar rules can be written for the transitions moving the head left and keeping it in place as well; denote as χ the formula obtained by combining all such χ_r with $+$. In other words, χ encodes the transition table of M.

3. One rule application can then be expressed with the subformula
$$\chi'\colon \psi(n, \rhd, \rhd).(\psi(n, \rhd, \rhd) + \chi'')^*.\chi.(\psi(n, \lhd, \lhd) + \chi'')^*.\psi(n, \lhd, \lhd)$$
where the subformula χ'' is obtained by combining $\psi(n, a, a)$ for all $a \in \Gamma$ with $+$.

4. Finally the whole formula states that the first configuration is the initial one, loops applying the rules and finally checks that the last configuration accepts:
$$\begin{aligned}\phi\colon \exists x_1([x_1]_l x_1 = \lhd\ .[x_1]_l x_1 = p_0.[x_1]_l x_1 = \rhd\ .[x_1]_l x_1 = c_1 \ldots [x_1]_l x_1 = c_n.\\ [x_1]_l x_1 = \lhd\ .([x_1]_r x_1 \neq \bot)^*.(\chi')^*.([x_1]_l \top)^*.[x_1]_l x_1 = p_m).\end{aligned}$$

This proves **PSPACE**-hardness, once Γ' is encoded into Σ in ϕ. □

6 Conclusions

We have introduced Alignment Calculus, a modal logic extension of relational calculus intended for databases where the entries in the columns are strings instead of atomic values. The logic is state-based, and a database state consists of not only relations but also an alignment of the pertinent strings. Properties of these alignments are specified using atomic string formulae, and modal operators express state changes leading to new alignments. The relational properties of the strings are described as in relational calculus. As a computational counterpart to Alignment Calculus we defined Alignment Algebra, a variant of relational algebra in which selection conditions are expressed by certain multitape finite state acceptors corresponding to the string formulae.

Alignment Calculus was shown to be very expressive, and to admit syntactic subclasses describing some well-known language classes. We also studied safety (evaluability within a finite domain) of queries expressed in Alignment Calculus. The modal extensions add a new source of undecidability to the problem. We identified, however, the class of right-restricted string formulae for which the safety problem remains decidable. This class is conjectured to contain most of the string queries encountered in practice; this is supported by the fact that queries in each level of the polynomial-time hierarchy can be expressed using these string formulae.

References

[1] J. Collado-Vides. The search for a grammatical theory of gene regulation is formally justified by showing the inadequacy of context-free grammars. *Computer Applications in the Biosciences*, 7(3):321–326, 1991.

[2] E.A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 16, pages 994–1072. Elsevier, 1990.

[3] M. Escobar-Molano, R. Hull, and D. Jacobs. Safety and translation of calculus queries with scalar functions (extended abstract). In *ACM SIGACT–SIGMOD–SIGART Symposium on Principles of Database Systems*, pages 253–264, 1993.

[4] Z. Galil and J. Seiferas. Time-space optimal string matching. *Journal of Computer and System Sciences*, 26:280–294, 1983.

[5] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, 1979.

[6] S. Ginsburg and H. Spanier. Mappings of languages by two-tape devices. *Journal of the ACM*, 12:423–434, 1965.

[7] S. Ginsburg and X. Wang. Pattern matching by rs-operations: Towards a unified approach to querying sequenced data (extended abstract). In *ACM SIGACT–SIGMOD–SIGART Symposium on Principles of Database Systems*, pages 293–300, 1992.

[8] R.R. Goldberg. Finite state automata from regular expression trees. *The Computer Journal*, 36(7):623–630, 1993.

[9] G. Grahne, M. Nykänen, and E. Ukkonen. Reasoning about strings in databases. In *ACM SIGACT–SIGMOD–SIGART Symposium on Principles of Database Systems*, pages 303–312, 1994.

[10] C. Helgesen and P.R. Sibbald. PALM - a pattern language for molecular biology. In *International Conference on Intelligent Systems in Molecular Biology*, pages 172–180, 1993.

[11] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[12] R. Hull and J. Su. Domain independence and the relational calculus (2nd revision). Technical Report 88–64, University of Southern California, Computer Science Department, November 1993.

[13] H.V. Jagadish, A.O. Mendelzon, and T. Milo. Similarity-based queries. In *ACM SIGACT–SIGMOD–SIGART Symposium on Principles of Database Systems*, pages 36–45, 1995.

[14] H.H. McAdams and L. Shapiro. Circuit simulation of genetic networks. *Science*, 269:650–656, 1995.

[15] G. Mecca and A.J. Bonner. Sequences, Datalog and transducers. In *ACM SIGACT–SIGMOD–SIGART Symposium on Principles of Database Systems*, pages 23–35, 1995.

[16] P. Pistor and R. Traunmüller. A database language for sets, lists and tables. *Information Systems*, 11:323–336, 1986.

[17] R. Ramakhrisnan, F. Bancilhon, and A. Silberschatz. Safety of recursive Horn clauses with infinite relations (extended abstract). In *ACM SIGACT–SIGMOD–SIGART Symposium on Principles of Database Systems*, pages 328–339, 1987.

[18] J. Richardson. Supporting lists in a data model (a timely approach). In *Very Large Data Bases Conference*, pages 127–138, 1992.

[19] H. Rogers. *Theory of Recursive Functions and Effective Computability.* McGraw-Hill, 1967.

[20] D. Sankoff and J.B. Kruskal, editors. *Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison.* Addison-Wesley, 1983.

[21] D.B. Searls. String Variable Grammar: A logic grammar formalism for the biological language of DNA. *Journal of Logic Programming*, pages 73–102, 1995.

[22] S. Sippu and E. Soisalon-Soininen. *Parsing Theory*, volume I: Languages and Parsing. Springer-Verlag, 1988.

[23] L.J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1977.

[24] J.D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume I. Computer Science Press, 1988.

[25] M. Vardi. The complexity of relational query languages. In *ACM Symposium on Theory of Computing*, pages 137–145, 1982.

[26] X. Wang. *Pattern Matching by Rs-Operations: Towards a Unified Approach to Querying Sequenced Data.* PhD thesis, University of Southern Carolina, 1992.

[27] A. Weber. On the valuedness of finite transducers. *Acta Informatica*, 27:749–780, 1990.

[28] A. Weber. On the lengths of values in a finite transducer. *Acta Informatica*, 29:663–687, 1992.

[29] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56:72–99, 1983.