Shuffle expressions and words with nested data

Henrik Björklund[1]* Mikołaj Bojańczyk[2]**

[1] University of Dortmund
[2] Warsaw University

Abstract. In this paper, we develop a theory that studies words with nested data values with the help of shuffle expressions. We study two cases, which we call "ordered" and "unordered". In the unordered case, we show that emptiness (of the two related problems) is decidable. In the ordered case, we prove undecidability. As a proof vehicle for the latter, we introduce the notion of higher-order multicounter automata.

1 Introduction

A data word is a word where each position, in addition to its finite alphabet label, carries a *data value* from an infinite domain. Recent times have seen a flurry of research on data languages, motivated chiefly by applications in XML databases and parameterized verification; see, e.g., [8, 10, 3, 5, 12, 4, 1]. One of the main results is that satisfiability for first-order logic with two variables is decidable over data words, as long as the only operation allowed on the data values is equality testing [3]. The same paper also demonstrates a close connection between data languages, *shuffle expressions*, and *multicounter automata*:

1. Multicounter automata. These are nondeterministic automata with many counters, which can be incremented and decremented, but zero tests are only allowed at the end of the word.
2. Shuffle expressions. These are regular expressions extended with intersection and the shuffle operation.
3. Two-variable data languages. These are properties of data words that can be expressed in certain two-variable logics.

The connection between multicounter automata (or Petri nets) and shuffle expressions was discovered in [6], while the connection between the first two and data languages was discovered in [3].

In this paper, we develop and investigate extensions of the above. We focus on *nested* data values and shuffle expressions. There are two principal motivations.

– When data values are only used to induce an equivalence relation on the positions, such as in the logics mentioned above, one of the chief applications is parameterized verification. A number of processes run in parallel, and in the resulting sequence of actions, the individual actions are annotated by the unique id of the process that performed them. This data word can be used to check global properties (by looking at the whole string) and local properties (by considering the sequence of actions of a single process).

* Supported by the Deutsche Forschungsgemeinschaft Grant SCHW678/3-1.
** Supported by Polish goverment grant no. N206 008 32/0810.

To model a system where processes can have subprocesses, and so on, we would want a data word with *nested* data values: each action carries the id of the subprocess which performed it as well as the id of the process that spawned the subprocess, and so on.

– In [6], nesting of the shuffle operation was not considered. This runs contrary to the unrestricted nesting of other operations in regular expressions, and begs the question what happens when unrestricted nesting of shuffles is allowed. We discover that the resulting languages are actually intimately related to languages with nested data. We also discover that this leads to undecidability; and decidability can be recovered only after adding a form of commutativity to the shuffle operation.

We note that our notion of shuffle expressions is different than the one used in some of the literature–see, e.g.,[7]–since we consider four different shuffle operators and allow intersection with regular languages.

Our main topic is logics over words with nested data. We study two two-variable fragments of first order, one with order and one without. The first one is shown to be undecidable, while the second is decidable. Nested shuffle expressions are the main proof vehicle, and an object of independent study. For instance, the expressions we consider capture the language:

$$a^{n_1}\#\cdots\#a^{n_k}\#b^{m_1}\#\cdots\#b^{m_k}\# \quad : \quad n_1,\ldots,n_k \text{ is a permutation of } m_1,\ldots,m_k$$

It is our opinion that the two concepts—logic with nested data and shuffle expressions—shed light on each other in a useful cross-fertilization.

2 A Logic for Words with Nested Data

Let A be a finite alphabet and Δ an infinite set, whose elements will be called *data values*. For $k \in \mathbb{N}$, a *word with k layers of data* is a word where every position, apart from a *label* in A, has k labels $d_1,\ldots,d_k \in \Delta$. The label d_i is called the *i-th data value* of the position. Therefore, such a word is an element $w \in (A \times \Delta^k)^*$. In w, the data values can be seen as inducing k equivalence relations $\sim_1,\ldots,\sim_k$ on the positions of w. That is, two positions are related by $\sim_i$ if they agree on the i-th data value.

We are interested in the data values only insofar as equality is concerned. Therefore, the relations $\sim_1,\ldots,\sim_k$ supply all the information required about a word; and could be used as an alternative definition. Adding more structure—such as a linear order—to the data values leads very quickly to undecidable decision problems, already with one layer, and even for very weak formalisms.

A word with k layers of data is said to have *nested data* if for each $i = 2,\ldots,k$ the relation $\sim_i$ is a *refinement* of $\sim_{i-1}$. In other words, if two positions agree on value i, then they also agree on value $i-1$. For the rest of this section, we fix k and only talk about words and languages with k layers of nested data. Instead of "word with k nested layers of data", we will just write "word with nested data".

In the spirit of Büchi's sequential calculus, we will use logic to express properties of words with nested data. In this setting, a word with nested data is treated as a model for logic, with the logical quantifiers ranging over word positions. The data values are accessed via the relations $\sim_i$. For instance, the formula

$$\forall x \forall y \quad (x \sim_2 y \Rightarrow x \sim_1 y) \wedge \cdots \wedge (x \sim_k y \Rightarrow x \sim_{k-1} y)$$

states that the data values are nested. (This formula is a tautology in our setting, where only models representing words with nested data values are considered.) Each label a from the finite label set A can be accessed by a unary relation; for instance $\forall x\ a(x)$ is true in words where all positions have label a. The linear order on word positions can be accessed via the relation $<$. For instance,

$$\forall x \quad (\forall y\ y \leq x) \Rightarrow (\forall y < x\ y \not\sim_1 x)$$

says that the last position has a different first (and consequently, all others as well) data value than than the other positions. We also use the successor relation $x = y + 1$. Although it can be defined in terms of the order, a third variable z is required, which is too much for the two-variable fragments we considered. We mainly consider satisfiability: given a formula, determine if there is a nested data word that satisfies it. This problem is undecidable in general, but by restricting the formulas, we can obtain decidable fragments.

Satisfiability is undecidable already for the following fragments, see [3]:

- There is only one layer of data. The formulas can use three variables; but the order on word positions can be accessed only via $x = y + 1$ and not $<$.
- There are two layers of data, but these are not necessarily nested. The formulas can use only two variables, $x = y + 1$, $x = y + 2$ and not $<$.

The largest known decidable fragment was presented in [3]:

- There is only one layer of data. The formulas can use only two variables, and the positions can be accessed by both $x = y + 1$ and $<$.

We want to generalize the above result to words with multiple layers of nested data. The result from [3] fails already for two layers:

Theorem 21
With two layers of nested data, satisfiability is undecidable for two-variable first-order logic with the relations $x = y + 1$ and $x < y$.

Proof. We encode computations of a two-counter machine with zero tests. Before presenting the full construction, we begin with an important component. Consider words with two layers of nested data, where the labels are

$$A = \{start, end, inc, dec\}$$

and the positions are labeled according to the regular language:

$$(start(inc + dec)^* end)^* \ .$$

This is equivalent to the conjunction of the following two-variable formulas:

$$\begin{array}{c} \forall x(\forall y\ y \neq x + 1) \Rightarrow end(x) \qquad \forall y(\forall x\ y \neq x + 1) \Rightarrow start(x) \\ \forall x \forall y\ y = x + 1 \Rightarrow (start(x) \iff end(y)) \ . \end{array} \tag{1}$$

We now say that each block of the form $start(inc + dec)^* end$ corresponds to a single data value on layer 1:

$$\begin{array}{cc} \forall x\ start(x) \Rightarrow \forall y < x\ x \not\sim_1 y & \forall x\ end(x) \Rightarrow \forall y > x\ x \not\sim_1 y \\ \forall x \exists y \geq x\ x \sim_1 y \wedge end(y) & \forall x \exists y \leq x\ x \sim_1 y \wedge start(y) \end{array} \tag{2}$$

Finally, the following formula uses equivalence classes (sets of positions with the same data value) on layer two to ensure that the operations *inc* and *dec* are balanced by each other:

$$\begin{array}{l} \forall x \forall y \ (x \sim_2 y \wedge inc(x) \wedge dec(y)) \Rightarrow x < y \\ \forall x \forall y \ (x \sim_2 y \wedge inc(x) \wedge inc(y)) \Rightarrow x = y \\ \forall x \forall y \ (x \sim_2 y \wedge dec(x) \wedge dec(y)) \Rightarrow x = y \end{array} \tag{3}$$

Consider a word with nested data that satisfies the above properties (1), (2) and (3). This word can be seen as a computation of a one counter machine without states, where *inc* corresponds to a counter increment, *dec* corresponds to a counter decrement, while *start*, *end* correspond to zero tests.

The above construction contains the basic idea for encoding the two counter machine. To get two counters instead of one, we expand the label alphabet from A to $\{1,2\} \times A$, with 1 corresponding to the first counter, and 2 corresponding to the second counter. The formula (1) is adapted to describe the language

$$start_1 start_2 (inc_1 + inc_2 + dec_1 + dec_2 + end_1 start_1 + end_2 start_2)^* end_2 end_1$$

while the formulas (2) and (3) are written twice, with one copy for each counter. The formulas corresponding to the bottom row of (2) have to say that *if* x is a counter 1 operation, then it should be followed (preceeded) by a counter 1 *end* label (*start* label) in the same $\sim_1$ class. Finally, states Q of the two-counter machine are encoded by further expanding the alphabet to

$$Q \times \{1,2\} \times A \ .$$

A two-variable formula with $y = x + 1$ can verify that states in neighboring positions are consistent with the transition relation of the two-counter machine. □

Even with the above result, however, not all hope is lost. Satisfiability is decidable if we lose the order $<$, even with arbitrarily many layers of data.

Definition 1. *$FO^2(+1, \sim_1, \ldots, \sim_k)$ is the fragment of first-order logic that uses only the two variables x and y and the following predicates.*

$a(x)$	*x has label a, where a is a label from A*
$y = x + 1$	*y is the position directly to the right of x*
$x \sim_i y$	*x and y have the same layer i data value, with $i \in \{1, \ldots, k\}$*

Theorem 22
Over words with nested data, satisfiability is decidable for $FO^2(+1, \sim_1, \ldots, \sim_k)$.

Our proof strategy for Theorem 22 is to first transform a logical formula into a normal form very similar to the one used in [3]—see Definition 2—and then to decide emptiness for formulas in the normal form. The normal form is described in terms of formulas with existential second-order monadic prefixes, which are described below.

Since the signature contains unary predicates $a(x)$ for labels from A, and since the decision problem concerned is satisfiability, we can consider without loss of generality formulas from a slightly more expressive logic, which we call $EMSO^2(+1, \sim_1, \ldots, \sim_k)$. These are obtained by prefixing a formula of $FO^2(+1, \sim_1, \ldots, \sim_k)$ with an existential second-order monadic prefix, i.e. formulas of the form

$$\psi = \exists R_1 \ldots \exists R_m \varphi$$

where φ is a formula from $FO^2(+1, \sim_1, \ldots, \sim_k)$ and $R_1, \ldots, R_n$ are unary predicate symbols. The formula φ is called the *(first-order) core*. Apart from the signature, φ can also use the unary predicates $R_1, \ldots, R_m$. It is easy to see that ψ is satisfiable if and only if its core is, so satisfiability is no more difficult after adding the second-order existential quantifiers.

Before presenting the normal form, we define two concepts used therein: type and profile. A *type* is a conjunction of unary predicates or their negations, all with the same variable. A type is called *complete* if it contains every predicate in the signature, as well as all the predicates $R_1, \ldots, R_n$ quantified in the second-order prefix. For instance, $a(x) \wedge \neg R_1(x)$ is a type, but it is not complete if the second-order prefix is $\exists R_1 \exists R_2$. The *profile* of a position j of a word is the information, for each $i = 1, \ldots, k$, about whether the predecessor and successor of j have the same level i data value as j. There are $(k+1)^2$ possible profiles: for the predecessor (and similarly for the successor) we need to say what is the largest layer $i = 0, \ldots, k$ on which the two positions agree, 0 meaning there is no predecessor.

Definition 2. *An $EMSO^2(+1, \sim_1, \ldots, \sim_k)$-formula is in* data normal form *if its core is a conjunct of formulas of one of the following forms.*

1. *Data blind property (i.e. a formula not referring to $\sim_1, \ldots, \sim_k$).*
2. *"Each $\sim_i$ class contains at most one node of type α."*
3. *"Each $\sim_i$ class with at least one node of type α has no node of type β."*
4. *"Each $\sim_i$-class with at least one node of type α also has a node of type β."*
5. *"Each node of type α has profile p."*

(Actually, conjuncts of the 3rd kind are redundant, but keep them to clarify our presentation.) The following proposition shows that the above is indeed a normal form.

Proposition 23 Every formula of $EMSO^2(+1, \sim_1, \ldots, \sim_k)$ can be effectively transformed into data normal form.

In the following example, we illustrate one of the techniques used in the proof of Proposition 23:

Example 1. Consider the following property: "each $\sim_1$ class contains at most three nodes with label a." The appropriate formula in data normal form is $\exists R_1 \exists R_2 \exists R_3 \varphi$, where φ says that: each node with label a satisfies exactly one of R_1, R_2, R_3 (a data blind formula); and each $\sim_1$ class contains at most one node with R_1, R_2 and R_3 respectively (three formulas of kind 2).

Theorem 22 will follow if show that satisfiability can be decided for formulas in data normal form. We will show this by compiling the formulas into a kind of shuffle expressions. This mirrors the approach in [3]. There are two key differences, however. First the shuffle expressions need to be nested—to account for multiple layers of data values—and second, the shuffle operation needs to be restricted—to account for the lack of order $\leq$. The appropriate definitions are presented in the Section 3.

To prove Proposition 23, we first rewrite the formula into an intermediate normal form.

Below, we will write $x \simeq_i y$ for $x \sim_i y \wedge x \not\sim_{i+1} y$. For $i = 1$, we let $x \simeq_i y$ mean $x \not\sim_1 y$. Also, for $i = k$, $x \simeq_i y$ means $x \sim_k y$.

Definition 3. *An $EMSO^2(+1, \sim_1, \ldots, \sim_k)$-formula is in* intermediate normal form *if its core is a conjunction of formulas of one of the following forms.*

1 $\forall x \forall y[\alpha(x) \wedge \beta(y) \wedge x \simeq_i y] \rightarrow d_{\leq 1}(x,y)$ *(In words: if* x, y *have types* α *and* β*, respectively, and* $x \simeq_i y$*, then they must be the same or neighbors.)*

2 $\forall x \exists y\ \alpha(x) \rightarrow [\beta(y) \wedge x \simeq_i y \wedge \varepsilon(x,y)]$ *(In words: if* x *is of type* α*, then there is a* y *of type* β *such that* $x \simeq_i y$ *and* y *is the successor of* x *(alternatively: predecessor of* x */ same as* x */ at least two positions away from* x*).)*

where

- α, β *are types;*
- $d_{\leq 1}(x,y)$ *is the disjunction* $x = y+1 \vee y = x+1 \vee x = y$*;*
- $\varepsilon(x,y)$ *is one of* $x = y+1$*,* $y = x+1$*,* $x = y$*, or* $\neg d_{\leq 1}(x,y)$*.*

Using a more or less straightforward adaptation of the proof from [3], one can show that each $FO^2(+1, \sim_1, \ldots, \sim_k)$-formula can be transformed into an $EMSO^2(+1, \sim_1, \ldots, \sim_k)$-formula in intermediate normal form.

It remains to show that formulas of types 1 and 2, where $1 \leq i \leq k-1$ can be expressed in data normal form. (For conjuncts in which the i in $x \simeq_i y$ is either 0 or k, the proof that they can be expressed in data normal form is the same as in [3].)

We begin by showing that a number of useful counting predicates can be defined in data normal form. Below we show that, using a formula in data normal form, we can verify that a predicate $\alpha^i_{\geq j}$—which can be guessed in the existential second-order prefix—is true for exactly one node in each $\sim_i$-class that contains at least j nodes of type α, with pairwise different data values on layer $i+1$. (When i is clear from the context we will sometimes omit it.)

To define $\alpha^i_{\geq 1}$ we can use one formula of type (2) and one of type (4). Once $\alpha^i_{\geq j}$ is defined, we can define $\alpha^i_{\geq j+1}$ in the following way.

1. First, we introduce two new predicate β_j, and γ_j. With a formula of type (3) we say that each $\sim_{i+1}$-class that has a node of type $\alpha^i_{\geq j}$, has no node of type $\neg\beta_j$.
2. With a formula of type (4) we say that each $\sim_{i+1}$-class with a β_j also has an $\alpha^i_{\geq j}$.
3. With a formula of type (4), we say that each $\sim_{i+1}$-class that has a node of type $\alpha \wedge \bigwedge_{l=1}^{j} \neg\beta_l$ has a node of type γ_j.
4. With a formula of type (4), we say that each $\sim_i$-class with a node of type γ_j has a node of type $\alpha^i_{\geq j+1}$.
5. With j formulas of type (3), we say that no $\sim_{i+1}$-class with an $\alpha^i_{\geq l}$, for $1 \leq l \leq j$, has a node of type $\alpha^i_{\geq j+1}$.

We also define predicates $\alpha^i_{=j}$, true for exactly one node in each $\sim_i$-class that contains exactly j $\sim_{i+1}$-classes that contain an α. It is defined by saying that each $\sim_i$-class with an $\alpha^i_{=j}$ has exactly one node of type $\alpha^i_{\geq j}$ (type (4)), and that each $\sim_i$-class that has an $\alpha^i_{=j}$ has no node of type $\alpha^i_{\geq j+1}$.

Naturally, we can define similar predicates for types β, γ, etc.

We now deal with formulas of type 1:

$$\forall x \forall y[\alpha(x) \wedge \beta(y) \wedge x \simeq_i y] \rightarrow d_{\leq 1}(x,y)$$

Since x and y belong to different $\sim_{i+1}$-classes ($1 \leq i \leq k$), $d_{\leq 1}$ can be replaced with $x = y+1 \vee y = x+1$, or in other words, x and y are neighbors.

A consequence of the above formula is that no $\sim_i$-class can contain more than two $\sim_{i+1}$-classes that contain a node of type α *and* more than two $\sim_{i+1}$-classes that contain

a node of type β. This is expressible in the following way. We mark all positions in $\sim_i$-classes that contain a position of type $\alpha^i_{\geq 3}$ with predicate R_1 and those in classes that contain a $\beta^i_{\geq 3}$ with R_2. The correctness of these markings can be expressed by formulas of type (2). We then say, using a formula of type (3) that no class that contains R_1 also contains R_2.

If a class has exactly 2 subclasses with an α (it has at least one node with $\alpha^i_{=2}$), we mark all nodes in these subclasses with predicates R_3 and R_4 respectively. Note that if some $\sim_i$-class has two α of type R_3 appearing in different $\sim_i$-zones (an $\sim_i$-zone is a contiguous set of positions that are in the same $\sim_i$-class), then all β in that class must be of type R_3. We encode this by requiring that each $\sim_i$-zone that contains an α of type R_3 should begin with a position of type P_3, and symmetrically for R_4 and P_4. Both properties can be expressed as a data-blind property that uses the profile marking (for the data-blind parts, we can use full monadic second-order logic, since the run of a finite automaton, without data, can be encoded by a data-blind formula in data normal form). Then we say that each class with at least two P_3 can contain no β that is not of type R_3. This can be done using two new predicate symbols. Next, if all α of type R_3 belong to the same $\sim_i$-zone, we enforce that all β that are not of type R_3 also belong to this zone. This is done in a similar way.

Finally, we say that if there is a β and an α which have different status w.r.t. R_3, R_4 (note that the β can be marked by neither), that are in the same $\sim_i$-zone, then they are neighbors. This is a data-blind property.

The only case left to explore is when there is exactly one subclass which contains an α. This can be managed in a similar, but slightly less complicated, way as the case with two such subclasses.

We now have to deal with formulas of type 2:

$$\forall x \exists y\ \alpha(x) \rightarrow [\beta(y) \wedge x \simeq_i y \wedge \varepsilon(x,y)]$$

The only interesting case for $\varepsilon(x,y)$ is $\neg d_{\leq 1}(x,y) \equiv d_{>1}(x,y)$.

Thus our formula says that for every α there is a β in the same $\sim_i$-class, which is not a neighbor, and which belongs to a different $\sim_{i+1}$-class. We first note that if there are at least 4 $\sim_{i+1}$-classes that contain a β, the condition is always fulfilled. Thus we have to handle the cases with 1,2, or 3 such subclasses.

If there are 3 subclasses with a β, say with data values d_1, d_2, d_3 (these are data values on layer $\sim_{i+1}$). We have to make sure that the following does not occur: data values d_2, d_3 contain one β each, there is a position x with α, whose $\sim_{i+1}$-class is d_1, such that position $x-1$ contains the unique β in subclass d_2, and position $x+1$ contains the unique β in subclass d_3. This can be described by a predicate Q, and we then say that each $\sim_i$-class with a $\beta^i_{=3}$ has no Q.

Consider next the case with 2 subclasses with a β. Just as in the proof for formulas of type 1(a), we mark these two $\sim_{i+1}$-classes with R_5 and R_6, respctively. Next, we mark all $\sim_i$-zones that contain a β of type R_5 with P_5. All α that are not of type R_5 and not in a zone P_5 are now taken care of, and we mark them with a new predicate. Next, we need to differentiate the P_5 zones into the ones with 1, 2, or 3 β of type R_5.

1. If there is only one such β, no α of type R_6 can be it's neighbor.
2. If there are two such β, no α of type R_6 can be a neighbor of both of them.
3. If there are 3 or more such β, everything is fine.

These three conditions can all be expressed in data normal form.

Finally, if there is only 1 subclass with a β, there can be no α in this subclass, and we distinguish three cases.

1. If there is only one β in the subclass, it can have no α neighbors.
2. If there is exactly two β in the subclass, they cannot be two positions apart with an α between them.
3. If there is more then two β everything is fine.

Each of the cases is expressible in data normal form.

We have shown that each formula in the logic from Theorem 22 has an equivalent formula in normal form. Theorem 22 now follows from Theorems 34 and 61.

3 Shuffle expressions

Recall that a word $w \in A^*$ is called a *shuffle* of words $w_1, \cdots, w_m \in A^*$ if the positions of w can be colored using m colors so that the positions with color $i \in \{1, \ldots, m\}$, when read from left to right, form the word w_i. If $K \subseteq A^*$ is a (possibly infinite) set of words, then *shuffle*$(K) \subseteq A^*$ is defined as

$$\{w : w \text{ is a shuffle of some } w_1, \ldots, w_m \in K, \text{ for some } m \in \mathbb{N}\} .$$

Note that the words $w_1, \ldots, w_m$ above may include repetitions. For instance, *shuffle*$(\{a\})$ contains all words a^*. Just as finite automata are connected to regular expressions, multicounter automata are connected to shuffle expressions. This is witnessed by the following result, essentially due to [6]:

Theorem 31
The following language classes are equal, modulo morphisms:

1. *Languages recognized by multicounter automata;*
2. *Languages of the form $L \cap$ shuffle(K), where L, K are regular.*

The qualification "modulo morphisms" means that any language from class 1 is a morphic image of a language in class 2. (Class 2 is simply contained in class 1, without need for morphisms.) The point of the morphism is to erase bookkeeping, such as annotations with accepting runs.

We can now ask what we get if we add intersection and *shuffle* to regular expressions. The answer is that we get more than we want:

Theorem 32
If regular expressions are extended with shuffle and intersection, all recursively enumerable languages can be defined modulo morphisms.

In particular, emptiness is undecidable for such extended regular expressions. Disallowing intersection trivializes the emptiness problem — which is the problem we are most interested in here — since *shuffle*(K) is nonempty if and only if K is. Theorem 32 follows directly from Theorems 41 and 42 below.

In this paper, however, we are most interested in decidability results, especially for logics with data values. It turns out that decidability can be recovered, if we consider a weaker form of shuffling, which is partly commutative.

3.1 Cutting and combining

To express our modification of the shuffle operation, it is most convenient to decompose *shuffle*(L) into two operations:

$$\mathit{shuffle}(L) = \mathit{combine}(\mathit{cut}(L)) \ . \tag{4}$$

The first *cut* operation sends a set of words $L \subseteq A^*$ to the set of traces obtained by cutting a word from L into pieces:

$$\mathit{cut}(L) = \{w_1|\cdots|w_k : w_1 \cdots w_k \in L\} \subseteq (A^*)^* \ .$$

In the above a *trace* is a sequence of finite words, which is written as $w_1|\cdots|w_k$, with $|$ separating consecutive words, called *segments*. Traces are denoted by θ or σ, and will be heavily used later on.

The second operation is called *combine*, and it sends a set of traces $L \subseteq (A^*)^*$ to the set of words that can be combined from these traces:

$$\mathit{combine}(L) = \{w : w \text{ is a combination of } \theta_1, \ldots, \theta_m \in L \ \} \subseteq A^* \ .$$

By saying that w is a *combination* of traces $\theta_1, \ldots, \theta_m$ we mean that positions of w can be colored with m colors, so that for each color $i = 1, \ldots, m$, the positions with color i give the trace θ_i. In the trace $\theta_i = w_1|\cdots|w_n$, the segments $w_1, \ldots, w_n$ correspond to *maximal* subwords of w that are assigned color i. Consider the following example.

$$\begin{array}{l} a\ b\ c\ c\ c\ b\ a\ c \\ 1\ 1\ 2\ 1\ 3\ 1\ 2\ 2 \end{array} \qquad \begin{array}{l} \theta_1 = ab|c|b \\ \theta_2 = c|ac \\ \theta_3 = c \end{array}$$

By the maximality requirement, the trace θ_2 cannot be replaced by $c|a|c$ Given the above definitions of *cut* and *combine*, it should be fairly clear that equation (4) holds. However, by tweaking the definitions of *cut* and *combine*, we will arrive at variants of the shuffle operation that are decidable and, more importantly, relevant to our investigation of nested data values.

The first modification gives us more control on the way words from $L \subseteq A^*$ are cut into traces. Let $K \subseteq A^*$ be a language. We define

$$\mathit{cut}_K(L) = \{w_1|\cdots|w_k : w_1 \cdots w_k \in L, w_1, \ldots, w_k \in K\} \subseteq (A^*)^* \ .$$

In other words, words from L are cut into traces where each segment belongs to K. Setting $K = A^*$ allows us to recover the standard cut operation, so cut_K is a generalization of *cut*. We only consider the case when K is regular.

The second modification concerns the operation *combine*. An *unordered trace* is a multiset of words, i.e. a trace where the order of segments is not important (however, the ordering of letters inside the segments is). The operation *ucombine*(L) treats the set of traces L as unordered traces:

$$\mathit{ucombine}(L) = \{w : w \text{ is an unordered combination of } \theta_1, \ldots, \theta_m \in L \ \} \subseteq A^* \ .$$

In the above, w is an *unordered combination* of $\theta_1, \ldots, \theta_m$ if w is a combination of traces $\sigma_1, \ldots, \sigma_m$, such that σ_i is obtained from θ_i by rearranging the order of segments. Thus abc is an unordered combination of traces $\theta_1 = c|a$ and $\theta_2 = b$.

3.2 Four kinds of shuffle expressions

From the above, we obtain four variants of the shuffle operation:

- Shuffle: $\mathit{shuffle}(L) = \mathit{combine}(\mathit{cut}(L))$.
- Controlled shuffle: $\mathit{cshuffle}_K(L) = \mathit{combine}(\mathit{cut}_K(L))$.
- Unordered shuffle: $\mathit{ushuffle}(L) = \mathit{ucombine}(\mathit{cut}(L))$.
- Unordered controlled shuffle: $\mathit{ucshuffle}_K(L) = \mathit{ucombine}(\mathit{cut}_K(L))$.

Each such operation gives rise to its own flavor of extended regular expressions. We will investigate and compare these flavors with respect to decidability and expressive power.

Definition 33 *Controlled shuffle expressions (CSE)* denote languages obtained by nesting the following operations:

- Standard regular expression operations: single letters $a \in A$, the empty word ϵ, concatenation, union and Kleene star.
- Intersection with regular languages.
- Controlled shuffle $\mathit{cshuffle}_K(L)$, where L is defined by a CSE, but K is a regular word language.
- Images under morphisms $f : A^* \to B^*$.

Shuffle expressions (SE), unordered shuffle expressions (USE) and *unordered controlled shuffle expressions (UCSE)* are defined analogously, by replacing the type of shuffle operation allowed. All types of operations can be freely nested.

The point of adding morphic images is to have a form of nondeterministic guessing in the expressions (and therefore more power). This will be illustrated in the following example. Note also that morphic images are necessary due to adding the intersection and shuffle operations; in standard regular expressions the projection operation does not add any power and can be eliminated.

Example 2. In the shuffle operation $\mathit{shuffle}(L)$, we have no control over the number of times the language L is used. In this example we show that we can enforce that it is used an even number of times. Let then $L \subseteq A^*$ be defined by an SE. The idea is to expand the alphabet A with a new symbol *start*; each word from L will be prefixed by this symbol. Consider then the expression:

$$K = \mathit{start} \cdot L \ .$$

If we now take the expression $\mathit{shuffle}(K)$, we can use the marker *start* to see how many times K was used. By intersecting with the regular language "even number of occurrences of *start*", we can make sure that it was used an even number of times. Finally, the markers can be removed using the erasing morphism $f : (A \cup \{\mathit{start}\})^* \to A^*$ defined by $f(\mathit{start}) = \epsilon$ and $f(a) = a$ for $a \in A$.

Example 3. Unordered shuffling is enough to express some counting properties: $\mathit{ushuffle}(ab)$ describes words in $(a+b)^*$ with the same number of a's and b's. Using intersection with the regular language a^*b^*, we get the language $\{a^n b^n\}$.

Example 4. Using the same idea as in the previous example, we can also get the language $L = \{a^n\#b^n\#\}$. Consider now the following expression:

$$(a^*\#)^*(b^*\#)^* \cap \mathit{ucshuffle}_K(L) \qquad \text{where } K = a^*\# + b^*\#$$

This expression defines the set of words

$$a^{n_1}\# \cdots \#a^{n_k}\#b^{m_1}\# \cdots \#b^{m_k}\#,$$

such that $n_1, \ldots, n_k$ is a permutation of $m_1, \ldots, m_k$.

3.3 From logic to shuffle expressions

In this section we state the reduction of satisfiability for $FO^2(+1, \sim_1, \ldots, \sim_k)$ to the emptiness problem for unordered controlled shuffle expressions.

Theorem 34
For every $FO^2(+1, \sim_1, \ldots, \sim_k)$-formula ϕ, a UCSE r can be effectively computed such that the language of r is non-empty if and only if ϕ is satisfiable.

Proof. We assume that ϕ is in *data normal form*; see Definition 2. The conjuncts of type (1) from the definition are easily taken care of. The properties they express are regular, so we can ensure them by intersecting with the appropriate regular expression after everything else has been taken care of.

Conjuncts of type (2)-(4) express properties of classes by referring to types. Notice that, when talking of types, these conjuncts only use the expressions "at least one node", "has a node", and "at most one node". Thus, the only relevant information about a class string (where the class can be w.r.t. any of the k equivalence relations), is the number of times each type appears. Furthermore, if a type appears at least twice, the exact number of times is irrelevant.

We start by considering only conjuncts of type (2)-(4) where the equivalence relation used is $\sim_k$. If we have a footprint mapping $f : T \to \{0, 1, 2\}$, where T is the set of types appearing in ϕ, that tells us if a type appears 0, 1, or 2 or more times, we know everything we need about the class string. We can easily compute the set F of those footprints that are allowed by the conjuncts from ϕ. If we construct, for each $f \in F$, a shuffle expression that accepts exactly those strings that have footprint f, we can combine these expressions (using the + operator) into one that accepts all correct class strings. Using the shuffle operator on this expression gives an expression whose language is such that every word can be extended with (level k) data values in such a fashion that the conjuncts of ϕ that use $\sim_k$ are satisfied.

The idea is to use this construction inductively, starting with level k, until, after using k shuffle operations, all conjuncts of type (2)-(4) are taken care of.

It remains to explain how to handle conjuncts of type (5). This is done using an extended alphabet, where each symbol, in addition to the type, contains a *profile*. The control of the shuffle expressions is used to ensure that the profiles are assigned correctly. That is, the level i unordered controlled shuffle operation uses a regular cut language K_i such that

- the first letter of each word must have a profile indicating that the previous position belongs to a different $\sim_i$-class;
- the last letter of each word must have a profile indicating that the next position belongs to a different $\sim_i$-class;

– all other positions have profiles indicating that the previous and next positions belong to the same $\sim_i$-class.

□

The corresondence between r and ϕ is actually stronger: r contains words obtained from models of ϕ by erasing data values. We do not know if the converse translation—from expressions to logic—can be done; possibly the expressions are strictly stronger than the logic. A similar reduction, from logic with order to CSE is possible, but the proof is omitted.

4 Ordered shuffle expressions

The following theorem relates CSE, SE and higher-order multicounter automata. The latter, to our best knowledge, are a new model.

Theorem 41
The following language classes are equal:

1. *Languages defined by controlled shuffle expressions (CSE);*
2. *Languages defined by shuffle expressions (SE);*
3. *Languages defined by higher-order multicounter automata;*
4. *Recursively enumerable languages.*

We define higher-order multicounter automata in Section 4.1, and prove their Turing completeness. The remaining three inclusions from Theorem 41 are shown in Sections 4.2, 4.3, and 4.4, respectively.

4.1 Higher-order multicounter automata

A multiset over A is a function $m : A \to \mathbb{N}$. We only consider finite multisets here, where all but a finite number of elements in A are assigned 0. We also consider higher-level multisets (which are also multisets). A level 1 multiset over A is a finite multiset over A. A level $k+1$ multiset over A is a finite multiset of level k multisets over A.

A level k multicounter automaton is defined as follows. It has a state space Q, an input alphabet Σ, and a multiset alphabet A. All of these are finite. The automaton reads an input word $w \in \Sigma^*$ from left to right. At each moment, its memory is a tuple $(q, m_1, m_2, \ldots, m_k)$, where q is one of the states in Q, and each m_i is a level i multiset over A, possibly undefined $\perp$. (We distinguish an empty multiset $\emptyset$ from an undefined one $\perp$.) The initial configuration is $(q_I, \perp, \perp, \ldots, \perp)$, where q_I is some designated initial state.

There is a finite set of transition rules, which say how the machine can modify its memory upon reading an input symbol (or doing an ϵ-transition). Each such transition rule is of the form: when in state q and upon reading the label $a \in \Sigma \cup \{\epsilon\}$, assume state p and do counter operation x. The counter operations are:

new_i Change m_i from $\perp$ to $\emptyset$.
inc_a Add $a \in A$ to the level 1 multiset m_1.
dec_a Remove $a \in A$ from the level 1 multiset m_1.
$store_i$ Add m_i to the level $i+1$ multiset m_{i+1}; then set m_i to $\perp$.
$load_i$ Remove nondeterministically some element m from m_{i+1} and store it in m_i. This transition is enabled only when $m_1, \ldots, m_i$ are all $\perp$.

We use $Counterops_k$ to denote the possible counter operations in a level k automaton. Note here that the automaton knows which m_i are undefined, since this is controlled by transitions new_i and $store_i$. On the other hand, the automaton does not know if a defined multiset m_i is empty, or not.

What is the accepting condition? We say a level k multiset is *hereditarily empty* if it is empty, or it consists only of hereditarily empty level $k-1$ multisets. The automaton accepts if $m_1, \ldots, m_k$ are all hereditarily empty multisets in all memory cells; and the control state belongs to a designated accepting set.

The above definition is similar, but not identical, to the notion of *nested Petri nets* from, e.g., [9].

Here we show that the machines are Turing complete, already on level 2.

Theorem 42
Level 2 *multicounter automata recognize all recursively enumerable languages.*

Proof. We show that a level 2 multicounter automaton can simulate a two-counter machine with zero tests. Since the latter type of machine is capable of recognizing all recursively enumerable languages, the statement follows.

A configuration of the two-counter machine, where counter 1 has value i and counter 2 has value j, will be represented by the following level 2 multiset:

$$\{\{\underbrace{x, \ldots, x}_{i \text{ times}}, a\}, \{\underbrace{x, \ldots, x}_{j \text{ times}}, b\}, \underbrace{\emptyset, \ldots, \emptyset}_{k \text{ times}}\} \ . \tag{5}$$

The occurrences $\emptyset$ are used for bookkeeping; the number k will correspond to the number of zero-tests that have been carried out in the run leading to this configuration. A configuration as above is called *proper*. Our automaton will have the property that improper configurations always lead to improper configurations; furthermore, a failed zero-test will lead to an improper configuration.

We now show how to represent the operations of the simulated machine:

- Zero test on counter 1. We do the following sequence of operations:

  $$load_1 \quad dec_a \quad store_1 \quad new_1 \quad inc_a \quad store_1 \ .$$

  If the configuration was improper, it will remain so. If it was proper, the level 2 from (5) multiset will become:

  $$\{\{a\}, \{\underbrace{x, \ldots, x}_{j \text{ times}}, b\}, \underbrace{\emptyset, \ldots, \emptyset}_{k \text{ times}}, \{\underbrace{x, \ldots, x}_{i \text{ times}}\}\} \ .$$

  If i was not 0, the above configuration will be improper.
- Increment on counter 1. We do the following sequence of operations:

  $$load_1 \quad dec_a \quad inc_x \quad inc_a \quad store_1 \ .$$

  A decrement is done the same way.
- The operations on counter 2 are as above, except b is used instead of a.

One can easily see that the automaton can reach a proper configuration as in (5) if and only if the simulated two-counter machine could have counter values (i, j). Furthermore,

the simulating machine can test (once, at the end of its run), if it has reached a proper configuration of the form:

$$\{\{a\}, \{b\}, \underbrace{\emptyset, \ldots, \emptyset}_{k \text{ times}}\} .$$

This is done by $load_1$ dec_a $store_1$ $load_1$ dec_b $store_1$ and testing if all memory cells are hereditarily empty. □

4.2 CSE are captured by SE.

We show that given a language L and a regular language K over alphabet A, we can define the language $cshuffle_K(L)$ using a shuffle expression (without controlled cut). The basic idea is to *color* the start and end of each subword from K with 0 or 1, using a scheme from [1]. We assume without loss of generality, that K does not contain the empty word ϵ.

First, we expand the alphabet to contain some bookkeping information. Let then A' be $A \cup \{0, 1\}$ and let

$$K' = 0 \cdot K \cdot 0 \ + \ 0 \cdot K \cdot 1 \ + \ 1 \cdot K \cdot 0 \ + \ 1 \cdot K \cdot 1.$$

Second, let $f : (A')^* \to A$ be the morphism that erases the bookkeeping information, defined by $f(a) = a$ if $a \in A$ and $f(a) = \epsilon$ if $a \in \{0, 1\}$.

Third, let L' be the set of words w that are of the form $(0+1)(A^*(01+01))^*A^*(0+1)$ and whose image $f(w)$ belongs to L.

Finally, let M be the regular language $((0 + 1) \cdot (A^* \cdot (00 + 11))^* \cdot A^* \cdot (0 + 1))$.

Now consider the SE

$$r = f(shuffle(L') \cap (K')^* \cap M).$$

We claim that the languages defined by r and $cshuffle_K(L)$ are the same.

Suppose a nonempty word w belongs to the language of $cshuffle_K(L)$. Then w can be cut into nonempty segments $w_1|\ldots|w_k$ such that each w_i belongs to K, and there is an equivalence relation $\sim$ on $\{w_1, \ldots, w_k\}$ such that for each equivalence class of $\sim$, the sequence of all subwords belonging to the class, read from left to right, form a word from L. (In the equivalence relation $\sim$, any two consecutive segments w_i and w_{i+1} are non-equivalent, for $i = 1, \ldots, k-1$.) Let w' be a word

$$c_1 w_1 c_2 c_2 w_2 c_3 c_3 \ldots c_k c_k w_k c_{k+1} \qquad c_1, \ldots, c_{k+1} \in \{0, 1\}$$

such that $c_{i+1} \neq c_j$ holds whenever $i < j$ are such that $w_i \sim w_j$ holds, and are chosen closest to each other for this property. Using a coloring argument, one can show that such a word always exists; see [1]. Clearly, w' belongs to $shuffle(L') \cap (K')^* \cap M$, and after f is applied, it is identical to w.

For the other inclusion, consider a word w described by the expression r. By definition of r, there must be a way of inserting zeroes and ones into w such that the resulting word w' belongs to $shuffle(L') \cap (K')^* \cap M$. Since K is a language over A, not A', there is a unique way of cutting w' into subwords $w'_1|\ldots|w'_k$ from K'. Since w' belongs to M, the last symbol of w'_i must be the same as the first symbol of w'_{i+1}, for all $i = 1, \ldots, k-1$. Let $T = \{\theta_1, \ldots, \theta_m\}$ be a set of traces witnessing the membership of $w' \in shuffle(L')$, i.e., $w' \in combine(T)$. We must show that for each trace

$\theta_i = u_{i,1}|\ldots|u_{i,n_i} \in T$, every segment $u_{i,j}$ must belong to K'. We know that $u_{i,j}$ cannot contain 00 or 11, since this is disallowed by L'. Furthermore, $u_{i,j}$ cannot also contain 01 or 10, since otherwise the resulting shuffle w' would also contain $01, 10$, contradicting to $w' \in M$. It remains to show that each segment $u_{i,j}$ has length at least three, and begins and ends with one of $\{0,1\}$. Suppose this is not the case, and let $u_{i,j}$ be the leftmost segment in w' that violates this property. We consider only the case when w' does not begin with $\{0,1\}$, the other two are similar. Let then x be the position in the word w' that corresponds to the beginning of the segment $u_{i,j}$; this position has a label $a \in A' \setminus \{0,1\} = A$. If x is the first position in the word, then the trace θ_i containing x starts with a, a contradiction with L'. Otherwise, consider the position $x-1$. Since x was chosen leftmost, then the positions $x-2$ and $x-1$ belong to some non-violating segment, and therefore have labels in A and $\{0,1\}$ respectively. This gives a subword of w' of the form $A\{0,1\}A$, a contradiction with M.

4.3 SE captured by automata.

The proof is by induction on the structure of a shuffle expression. The cases of union, intersection with a regular language and projection are easy. The only nontrivial part is the following lemma:

Lemma 1. *If L is recognized by a level k multicounter automaton, then shuffle(L) is recognized by a level $k+1$ multicounter automaton.*

Proof. Let $\mathcal{A}$ be the level k automaton for L. The general idea is that to accept

$$\mathit{shuffle}(w_1, \ldots, w_n)$$

we simulate in parallel the runs of $\mathcal{A}$ on all the words $w_1, \ldots, w_n \in L$. In order to do this, we use the multiset at level $k+1$ to store multiple configurations of $\mathcal{A}$. The idea is to define a folding operation, which takes a configuration $(q, m_1, \ldots, m_k)$, and folds it into a single level k multiset m. A dual unfolding operation goes from m back to the configuration.

First, we define the folding operation. Then we show how it is used to get an automaton for *shuffle*(L).

Let a^k be the level k multiset defined by $a^0 = a$ and $a^{i+1} = \{a^i\}$. The folding of a sequence $m_1, \ldots, m_k$ with respect to a is a level k multiset, denoted $[m_1, \ldots, m_k]_a$, defined by

$$[m_1]_a = m_1$$
$$[m_1, \ldots, m_k]_a = m_k + \{[m_1, \ldots, m_{k-1}]_a + a^{k-1}\}.$$

In case m_i is undefined, we use $\perp^i$ instead of m_i in the above construction, where $\perp$ is a special symbol.

It is easily seen that the folding can be realized by an automaton. That is, we can use a macroinstruction "a-fold", which transforms a configuration with memory

$$m_1, \ldots, m_k$$

into one with memory

$$\perp, \ldots, \perp, [m_1, \ldots, m_k]_a.$$

Dually, we use a macroinstruction "a-unfold", which does the inverse.

Now, we can use the folding operation to encode n level k configurations as one level $k+1$ multiset. We encode

$$(q_1, m_1^1, \ldots, m_k^1), \ldots, (q_n, m_1^n, \ldots, m_k^n)$$

in the single configuration

$$\bot, \ldots, \bot, [m_1^1, \ldots, m_k^1]_{q_1} + \cdots + [m_1^n, \ldots, m_k^n]_{q_n}.$$

The simulating automaton guesses when to switch from one $\mathcal{A}$-configuration to another. It folds the current configuration, stores it in the $k+1$-multiset, loads the new configuration, and unfolds it. □

4.4 Automata are captured by CSE.

In the following proof, it will be convenient to assume that each transition has its own name. That it, the automaton has a set T (names of transitions), and a mapping

$$T \to Q \times A \times \mathit{Counterops}_k \times Q \tag{6}$$

that says what each transition does. We will write $\mathcal{A}(t)$ to denote the result of this mapping for a transition name t in an automaton $\mathcal{A}$. A run is a sequence of transitions $t_1 \cdots t_n$ that can be decorated with configurations $c_1 \cdots c_{n+1}$ so that transition t_i takes the automaton from c_i to c_{i+1}. The run is accepting if it admits a decoration where c_1 is an initial and c_{n+1} is an accepting configuration. (Note that the decoration with $c_1 \cdots c_{n+1}$ need not be unique.)

Proposition 43 Let $\mathcal{A}$ be a higher-order multicounter automaton. There is a controlled shuffle expression, which describes all accepting runs of $\mathcal{A}$.

Clearly, once we have an expression for the transitions, we can use a projection to obtain the set of input words that are accepted. Therefore, this proposition will show the inclusion of 3 in 1 in Theorem 41. The proof of the Proposition is by induction on the level k of $\mathcal{A}$. For $k = 1$, the result is known [3].

Let us fix a level $k+1$ automaton $\mathcal{A}$, described as in (6). We begin by defininining a level k automaton $\mathcal{B}$, which is going to correspond to what $\mathcal{A}$ does to a single level k multiset. Then we will show that the accepting runs can be obtained by shuffling accepting runs of $\mathcal{B}$.

The automaton $\mathcal{B}$ has the same states as $\mathcal{A}$; except that it also has a new "limbo" state X. The automaton $\mathcal{B}$ has the same transition names as $\mathcal{A}$, but the corresponding actions are changed:

$$\mathcal{B}(t) = \begin{cases} (p, a, \epsilon, X) \text{ if } \mathcal{A}(t) = (p, a, \mathit{load}_k, q) \\ (X, a, \epsilon, q) \text{ if } \mathcal{A}(t) = (p, a, \mathit{store}_k, q) \\ \mathcal{A}(t) \qquad \text{otherwise} \end{cases}$$

By inductive assumption, the set $L \subseteq T^*$ of accepting runs of $\mathcal{B}$ can be described by a cut shuffle expression. Therefore, Proposition 43 will follow once we establish the following Lemma:

Lemma 2. *There are regular languages $S, K \subseteq T^*$ such that the set of accepting runs of $\mathcal{A}$ is exactly $\mathit{cshuffle}_K(L) \cap S$.*

Proof. Let $T' \subseteq T$ be the transitions of $\mathcal{A}$ that do a load_k. Then we set K to be $(T \setminus T') \cdot T'$. The language S is defined to be the those $t_1 \cdots t_n \in T^*$ where for each $i = 1, \cdots, n-1$ the source state of $\mathcal{A}(t_{i+1})$ is the same as the target state of $\mathcal{A}(t_i)$. □

5 Unordered shuffle expressions

In this section, we state the decidability of the emptiness problems for unordered shuffle expressions, controlled (UCSE) or not (USE). Since $ushuffle(L) = ucshuffle_{A^*}(L)$ if A is the alphabet of L, it is clear that USE is a special case of UCSE. Nevertheless, we chose to state the following independently:

Theorem 51
Emptiness for unordered shuffle expressions is decidable.

The reason is that the proof is considerably less involved than for the controlled case. It uses a reduction to finite word automata equipped with a Presburger counting condition.

We show a decision procedure for emptiness of unordered (uncontrolled) shuffle expressions USE. Furthermore, we show that these are weaker than unordered controlled shuffle expressions.

Before we describe the decision procedure in Section 5.2, we recall the basic tools that will be used in this section and the next.

5.1 Semilinear sets and Presburger automata

A set $X \subseteq \mathbb{N}^k$ is called *linear*, if there are vectors $\overline{v}, \overline{v}_1, \ldots, \overline{v}_n \in \mathbb{N}^k$ such that

$$X = \{\overline{v} + i_1\overline{v}_1 + \cdots + i_n\overline{v}_n : i_1, \ldots, i_n \in \mathbb{N}\} .$$

The vector $\overline{v}$ is called the *base*, while the vectors $\overline{v}_1, \ldots, \overline{v}_n$ are called the *periods*. A set is called *semilinear*, if it is a finite union of linear sets.

There are two important properties of semilinear sets. The first is their connection with Presburger formulas; while the second is their connection with Parikh images.

We begin by recalling the connection of semilinear sets with Presburger formulas. A *Presburger formula* is a formula that can use addition $+$, the constants $0, 1$ and quantification over natural numbers $\exists, \forall$. Multiplication is not allowed, but free variables are allowed. When there are free variables $x_1, \ldots, x_k$, a Presburger formula can be seen as defining a set of vectors in $\mathbb{N}^k$. For instance, the formula

$$\varphi(x) \quad = \quad (\exists y.\ x = y + y) \wedge (\exists z.\ x = 1 + 1 + 1 + z)$$

defines the set of even natural numbers not smaller than 3. This set is semilinear, which is no coincidence:

Theorem 52
The set of vectors in $\mathbb{N}^k$ defined by a Presburger formula with k free variables is described by an effectively obtained semilinear set.

We now proceed to define Parikh images. Let $w \in A^*$ be a finite word. The *Parikh image* $\pi_A(w) \in \mathbb{N}^A$ of this word is a vector, which on coordinate $a \in A$ has the number of occurrences of the letter a in the word w. When the alphabet A will be understood from the context, we will sometimes omit the A subscript and simply write $\pi(w)$. The Parikh image can be also applied to languages: $\pi(L) = \{\pi(w) : w \in L\}$.

Theorem 53 ([11])
The Parikh image of a regular language is semilinear.

Definition 54 A *Presburger automaton* $(\mathcal{A}, X)$ is a finite nondeterministic automaton $\mathcal{A}$ with states Q, along with a semilinear set $X \subseteq \mathbb{N}^Q$. The automaton accepts a word if it has a run that ends in an accepting state, and where the number of times each state is used in the run is consistent with X.

Presburger automata can be used to accept languages such as $\{a^n b^m c^{m+n}\}$.

Presburger automata can also be defined as a type of counter automata. In this alternative definition, the automaton has a finite set C of counters, which have integer (possibly negative) values. In each transition (ϵ-transitions are allowed), the counters can be incremented or decremented (but they can not be tested). The automaton accepts if at the end of the word, it reaches an accepting state and all counters have value 0. One can easily show that the two definitions are equivalent.

5.2 Decision procedure for USE

The general idea is that since there is no control over the cutting, only simple counting is sufficient. We would like to use Presburger automata. However, this will not work for a rather shallow reason: Presburger automata are not closed under Kleene star, which is needed to recognize the language

$$\{a^n b^n : n \in \mathbb{N}\}^* \ .$$

Our approach is to consider Presburger automata extended with Kleene star. Fortunately, it turns out that this construction does not need to be nested (i.e. we need not consider Presburger automata over Kleene stars of Presburger automata).

An *extended Presburger* automaton is defined as follows. (We base this definition on the alternative definition of Presburger automata, where counters are used). In the extended automaton, there is a limited form of zero-test allowed during the run (and not only at the end of the run). In the limited form of zero-test, the automaton can test if *all* counters are zero.

Lemma 3. *The Parikh image of a language recognized by an extended Presburger automaton is semilinear.*

Proof. In the first step, one can easily show that the Parikh image of a (non-extended) Presburger automaton is semilinear. This is because semilinear sets are closed under intersection, projection and contain all Parikh images of regular languages.

By redoing the proof of Kleene's theorem for regular word languages, one can see that the Parikh image of an extended Presburger automaton can be obtained by taking unions and Kleene stars of Parikh images of non-extended Presburger automata.

Once this has been shown, the result follows, since semilinear sets are closed under Kleene star

$$X^* = \bigcup_{i \in \mathbb{N}} \{\overline{x}_1 + \cdots + \overline{x}_n : \overline{x}_1, \ldots, \overline{x}_n \in X\} \ ,$$

which follows from Theorem 53. □

Theorem 55
Every language defined by a USE is recognized by some extended Presburger automaton.

Proof. The proof is by induction on the size of the USE. One can easily see that extended Presburger automata are closed under union and projection (thanks to nondeterminism); under concatenation (using a different two sets of counters for the two concatenated words); and under Kleene star (thanks to the zero-test).

The only more difficult part is the unordered shuffle operation. But this is easy, if there is no cutting involved. Indeed, since we have no control over where the words will be cut, one can easily see that

$$\mathit{ushuffle}(L) = \{w : \pi(w) \in \pi(L^*)\} \ .$$

Therefore, in order to determine if a word w belongs to $\mathit{ushuffle}(L)$, the automaton need only verify a semilinear property of the Parikh image $\pi(w)$. This can be easily done using the counters. □

To give a feeling of what Presburger automata cannot do, we present here a negative result.

Proposition 56 Extended Presburger automata are stronger than Presburger automata. Furthermore, even extended Presburger automata do not recognize all languages defined by UCSE.

Before we prove this proposition, we state an important Corollary, which follows from the above proposition and from Theorem 55:

Corollary 57 USE are weaker than UCSE.

Proof (of Proposition 56). The first part of the statement is witnessed by the language:

$$\{a^n b^n : n \in \mathbb{N}\}^* \ .$$

We do not show that this language cannot be recognized by Presburger automata. The reader can fill in this proof, based on the more difficult proof of the second part of the statement.

The language witnessing the second part of the statement is the language M from Example 4: we claim it is not recognized by any extended Presburger automaton. The general idea is that languages accepted by extended Presburger automata satisfy a certain swapping condition, which is not satisfied by the language M.

Let $\mathcal{A}$ be an extended Presburger automaton accepting M, with states Q. For each pair of states $p, q \in Q$, let $L_{p,q}$ be the set of words, which admit a run that takes $\mathcal{A}$ from p to q, and does not change the counter values (i.e. for each counter, the number of increments is balanced by the number of decrements). Some of the languages $L_{p,q}$ are finite, some are infinite. Let N be the length of the longest word in the finite languages among $L_{p,q}$.

Let n be larger than both N and $|Q|^2 + 1$. Consider the word

$$w = a^{2^n} \# a^{2^{n-1}} \# \cdots \# a^4 \# a^2 \# a^1 \# b^1 \# b^2 \# b^4 \# \cdots \# b^{2^{n-1}} \# b^{2^n} \# \ .$$

This word belongs to the language M, and therefore should be accepted. Let ρ be an accepting run witnessing this acceptance. First we show that the only zero-tests the automaton can do are in the prefix a^{2^n} and in the suffix b^{2^n}. Indeed, assume that there is a zero-test outside this prefix and suffix. We assume without loss of generality that it is in the first half of the word (the other case is analogous). Let q be the state in which

this zero-test is done; and let p be the initial state. This means that some prefix of the word w belongs to $L_{p,q}$. Since the zero-test was done after reading the first $2^n > N$ positions, the set $L_{p,q}$ must be infinite. This can be easily exploited to form a word outside M that is accepted by the automaton.

Therefore, the accepting run does no zero-tests in the fragment

$$a^{2^{n-1}} \# \cdots \# a^4 \# a^2 \# a^1 b^1 \# b^2 \# b^4 \# \cdots \# b^{2^{n-1}} \ .$$

For $i = n-1, \ldots, 1$ let p_i be the state assumed by ρ in the first position of the block a^{2^i} and let q_i be the state assumed by ρ in the middle – that is 2^{i-1}-st – position of the block a^{2^i}. If $n-1$ is larger than the square of the state space of $\mathcal{A}$, there must be some $i < j$ with $p_i = p_j$ and $q_i = q_j$. We can therefore swap the subword that leads from p_i to q_i with the subword that leads from p_j to q_j; this will not affect the number of states used in the run. Therefore, the word after the swap will also be accepted by the automaton. This is a contradiction, since after the swap, both the i-th and j-th block of a's will have length $2^{i-1} + 2^{j-1}$. □

The above result shows that even extended Presburger automata are not strong enough to recognize language defined by UCSE. In [2] it was shown that (unextended) Presburger automata are strong enough the recognize all UCSE where the shuffle operation is on the top level. Therefore, the above lemma shows that the approach from [2] breaks down when arbitrary UCSE are considered.

6 Unordered controlled shuffle expressions

As stated in the introduction, the main goal of this paper is to show decidability of satisfiability for the 2-variable logic from Definition 1 over words with nested data. Theorem 34 shows that this problem reduces to emptiness for UCSE. We are now ready to complete the proof of Theorem 22, by stating the main combinatorial result of the paper:

Theorem 61
Emptiness for unordered controlled shuffle expressions is decidable.

The proof is rather involved, and is based on a study of the Parikh images [11] of languages defined by UCSE. Due to space limitiations, we can can only present here a very brief outline of the key ideas.

We actually focus on proving the following:

Theorem 62
The Parikh image of a language defined by a UCSE is semilinear.

Theorem 61 then follows as a corollary, since, as we will see, the Parikh image is effectively obtained, and can therefore be tested for emptiness.

The following easy technical lemma will be convenient:

Lemma 4. *Every UCSE is equivalent to one where intersection with regular languages is used only next to the unordered controlled shuffle operation.*

Note however, that the unordered controlled shuffle operation can be nested, and therefore so can intersection with regular languages.

The proof of Theorem 62 is by induction on the number of nested unordered controlled shuffle operations in the UCSE. We assume without loss of generality that the expression is as in Lemma 4. All operations of the UCSE are handled as in the Parikh theorem, the only new step is the case:

$$\mathit{ucshuffle}_M(L) \cap K \ , \tag{7}$$

where L is defined by a UCSE and K, M are regular languages.

We will reduce the statement to the following result:

Proposition 63 Let $L \subseteq A^*$ be a language with a semilinear Parikh image. The language $\mathit{ucshuffle}_A(L)$ is definable by a Presburger automaton.

The assumptions in the above proposition are very strong, since we assume that the words are cut into traces with one letter segments. In particular, a word from L can be identified with its Parikh image, since unordered shuffles are involved here. On the other hand, the conclusion is stronger, since we define a Presburger automaton for the whole language, and not just to describe its Parikh image. (The Parikh image of a Presburger automaton is semilinear.) The above Proposition 63 is related to a result in [2]. The result in [2] is stronger in that it considers trees and not words, on the other hand it is weaker in that L is a finite language in [2].

In Section 6.1, we show how the crucial inductive step (7) from Theorem 62 can be reduced to Proposition 63. Then, in Section 6.2, we prove the proposition.

6.1 Reduction to Proposition 63

We now proceed to show how Theorem 62 follows from Proposition 63. As noted previously, the crucial step is showing that the Parikh image

$$\pi(\mathit{ucshuffle}_M(L) \cap K) \subseteq \mathbb{N}^A$$

is semilinear, as long as $L \subseteq A^*$ is defined by an UCSE and $K, M \subseteq A^*$ are regular languages. Our proof is a long sequence of algebraic manipulations. In the end, we will refer to Lemma 3, which says that the Parikh image of an extended Pressburger automaton is semilinear.

By unraveling the definition, the above set becomes

$$\begin{aligned}\{\pi(\theta_1) + \cdots + \pi(\theta_k) :\ & \theta_1, \ldots, \theta_k \in \mathit{cut}_M(L), \\ & K \text{ contains an unordered combination of } \theta_1, \ldots, \theta_k\} \ .\end{aligned} \tag{8}$$

Let $\mathcal{A}$ be a finite deterministic automaton recognizing the language K; with states Q. Each word $w \in A^*$ induces a state transformation $f_w : Q \to Q$. Given a trace $\theta = w_1|\cdots|w_k \in (A^*)^*$, we denote by $\mathit{short}(w)$ the trace $f_{w_1}|\cdots|f_{w_k}$. This new trace is over a large alphabet Q^Q, however, each word in the trace has only one letter. Furthermore, let $K' \subseteq (Q^Q)^*$ be the set of those sequences $f_1 \ldots f_k$ of state transformations, such that the composition $f_k \circ \cdots \circ f_1$ maps the initial state of $\mathcal{A}$ to an accepting state. Clearly, for any traces $\theta_1, \ldots, \theta_k$ over A, the following two conditions are equivalent:

- K contains an unordered combination of $\theta_1, \ldots, \theta_k$;
- K' contains an unordered combination of $\mathit{short}(\theta_1), \ldots, \mathit{short}(\theta_k)$).

Therefore, the set (8) becomes:

$$\{\pi(\theta_1) + \cdots + \pi(\theta_k) : \begin{array}{l} \theta_1, \ldots, \theta_k \in cut_M(L), \\ K' \text{ contains an unordered combination} \\ \quad \text{of } short(\theta_1), \ldots, short(\theta_k))\}. \end{array} \tag{9}$$

This can be restated in the following way:

$$\{\overline{y}_1 + \cdots + \overline{y}_k : \begin{array}{l} \text{there exist } \theta_1, \ldots, \theta_k \in cut_M(L) \text{ such that} \\ K' \text{ contains an unordered combination} \\ \quad \text{of } short(\theta_1), \ldots, short(\theta_k)) \text{ and} \\ \overline{y}_1 = \pi(\theta_1), \ldots, \overline{y}_k = \pi(\theta_k)\} \ . \end{array} \tag{10}$$

The point of restating the set (9) in the form (10) is to switch from quantifying over $\theta_1, \ldots, \theta_k$ to quantifying over $short(\theta_1), \ldots, short(\theta_k)$:

$$\{\overline{y}_1 + \cdots + \overline{y}_k : \begin{array}{l} \text{there exist } \sigma_1, \ldots, \sigma_k \in short(cut_M(L)) \text{ such that} \\ K' \text{ contains an unordered combination of } \sigma_1, \ldots, \sigma_k \text{ and} \\ \text{for each } i = 1, \ldots, k \text{ there exists } \theta_i \in cut_M(L) \\ \text{such that } \overline{y}_i = \pi(\theta_i) \text{ and } \sigma_i = short(\theta_i) \ .\} \end{array} \tag{11}$$

We want to quantify over $\sigma_1, \ldots, \sigma_k$, because these are traces where each word has a single letter (a state transformation); and as such can be identified with their Parikh images $\pi(\sigma_1), \ldots, \pi(\sigma_k)$. In particular, the last two lines in (11) can be seen as expressing a property of two vectors: $\overline{y}_i$ and $\pi(\sigma_i)$. In the following lemma, we show that this vector property is semilinear:

Lemma 5. *The following relation is semilinear:*

$$R = \{(\pi(\theta), \pi(short(\theta)) : \theta \in cut_M(L)\} \subseteq \mathbb{N}^A \times \mathbb{N}^{Q^Q} \ .$$

Proof. We consider a new language P. The idea is that each word from P contains information about both θ and $short(\theta))$. The language P contains words

$$w \in (A^* Q^Q)^Q$$

such that: 1) if the labels from Q^Q are removed, the words belongs to L; 2) every maximal subword in A^+ belongs to M; and 3) if u is a maximal subword of w in A^+Q^Q, then the last letter describes the state transformation of the rest of the word. To obtain the language P from L, the only necessary operations are expanding the alphabet and intersecting with a regular language (the intersection checks properites 2) and 3)). Therefore, the UCSE for P has the same nesting depth of the unordered controlled shuffle operation as the UCSE for L. Therefore, by inductive assumption from Theorem 62, the Parikh image $\pi(P)$ is semilinear. Finally, the relation R in the statement of the lemma can be computed from $\pi(P)$ by using projections; therefore R is semilinear. □

A consequence of the above lemma is that the Parikh image of

$$short(cut_M(L))$$

is semilinear, since it is the projection of the relation R onto the $\pi(short(\theta))$ coordinate. Therefore, the set from (11) can be restated as

$$\{\overline{y}_1 + \cdots + \overline{y}_k : \begin{array}{l} \text{there exist one-letter traces } \sigma_1, \ldots, \sigma_k \text{ over } Q^Q \text{ such that} \\ K' \text{ contains an unordered combination of } \sigma_1, \ldots, \sigma_k \text{ and} \\ (\overline{y}_1, \pi(\sigma_1)), \ldots, (\overline{y}_k, \pi(\sigma_k)) \in R\} \ . \end{array} \tag{12}$$

Recall that our goal is to prove that the above set is semilinear. We will prove a slightly more general result, which only uses the assumption that R is semilinear and K' is regular (and not necessarily of the special form that appears in this proof). To clean up the notation, in the below proposition we replace K' by K, replace Q^Q by A and use the name B for the coordinates of the vectors $\overline{y}_1, \ldots, \overline{y}_k$:

Proposition 64 Let $X \subseteq \mathbb{N}^A$ and $R \subseteq \mathbb{N}^A \times \mathbb{N}^B$ be semilinear sets and let $K \subseteq A^*$ be a regular language. The following subset of $\mathbb{N}^B$ is semilinear:

$$\{\overline{y}_1 + \cdots + \overline{y}_k : \text{ there exist one-letter traces } \sigma_1, \ldots, \sigma_k \text{ over } A \text{ such that } K \text{ contains an unordered combination of } \sigma_1, \ldots, \sigma_k \text{ and } (\overline{y}_1, \pi(\sigma_1)), \ldots, (\overline{y}_k, \pi(\sigma_k)) \in R\} .$$

Before we proceed with the proof of this proposition, we would like to alert the reader to an important aspect of the clause "K contains an unordered combination of $\sigma_1, \ldots, \sigma_k$". The reader will recall that in an unordered combination (or any combination, for that matter), the segments of each trace must be separated by segments from other traces. For instance, *abc* is not an unordered combination of traces $a|b$ and c; the only unordered combinations are *acb* and *bca*. This requirement will pose difficulties, and it makes the easy proofs from Section 5 inapplicable to this case.

Our proof of Proposition 64 consists of two steps. First we show in Lemma 6, that without loss of generality we may assume that the relation R in the statement above can be assumed to be in a certain normal form, and prove a key property of this normal form. Second, we use this property to prove Proposition 64, under the assumption that Proposition 63 holds. This finishes the reduction from Theorem 62 to Proposition 63.

The *nonzero domain* of a vector is the set of coordinates where it has nonzero values. Two vectors are *independent* if they have disjoint nonzero domains. The *defining vectors* of a semilinear set are all the base and period vectors of its linear components. A semilinear set is in *normal form* if a) all its defining vectors are pairwise independent; and b) every linear component has a nonzero base vector.

Lemma 6. *Without loss of generality, we may assume that the relation R is in normal form.*

Proof. The general idea is that for each base and period vector, we add new coordinates R, so that the vectors use disjoint coordinates. We now give a more formal argument. Let $f : C \to D$ be a mapping. This mapping can be naturally carried over to vectors: $[f] : \mathbb{N}^C \to \mathbb{N}^D$, with $[f](v)$ having on coordinate $d \in D$ the sum of all values of v on coordinates $c \in C$ with $f(c) = d$.

It is fairly easy to see that for any semilinear relation R, in particular the semilinear relation $R \subseteq \mathbb{N}^A \times \mathbb{N}^B$ in Proposition 64, one can find

$$C, \quad f : C \to A \cup B, \quad S \subseteq \mathbb{N}^C \text{ in normal form,}$$

such that R is the image of S under $[f]$. By basic properties of regular languages, $f^{-1}(K)$ is a regular language. Using f and S, the set from Proposition 64 can be written as:

$$\{[f](\overline{y}_1) + \cdots + [f](\overline{y}_k) : \text{ there exist one-letter traces } \sigma_1, \ldots, \sigma_k \text{ over } C \text{ such that } f^{-1}(K) \text{ contains an unordered combination of } \sigma_1, \ldots, \sigma_k \text{ and } (\overline{y}_1, \pi(\sigma_1)), \ldots, (\overline{y}_k, \pi(\sigma_k)) \in S\} .$$

The reduction then follows, since $[f]$ can be moved outside the whole expression, thanks to the equality:

$$[f](\overline{y}_1) + \cdots + [f](\overline{y}_k) = [f](\overline{y}_1 + \cdots + \overline{y}_k)$$

□

Normal form of semilinear relations. In this section we analyze the structure of semilinear sets in normal form. Using the assumption that R is in normal form, we will show that the set

$$\{\overline{y}_1 + \cdots + \overline{y}_k : (\overline{x}_1, \overline{y}_1), \ldots, (\overline{x}_k, \overline{y}_k) \in R\} \subseteq \mathbb{N}^B$$

only depends only on the sum $\overline{x}_1 + \cdots + \overline{x}_k \in \mathbb{N}^A$ and not the individual components $\overline{x}_1, \ldots, \overline{x}_k$ of the sum. In fact, we will show an even stronger result, where the relationship between $\overline{x}_1 + \cdots + \overline{x}_k$ and $\overline{y}_1 + \cdots + \overline{y}_k$ is a semilinear relation:

Lemma 7. *Let $R \subseteq \mathbb{N}^A \times \mathbb{N}^B$ be in normal form. There are semilinear relations*

$$S \subseteq \mathbb{N}^A \times \mathbb{N}^B \qquad T \subseteq \mathbb{N}^A$$

such that for any $\overline{x}_1, \ldots, \overline{x}_n \in \mathbb{N}^A$ and $\overline{y} \in \mathbb{N}^B$, we have

$$\begin{array}{l} \overline{x}_1, \ldots, \overline{x}_n \in T \text{ and} \\ (\overline{x}_1 + \cdots + \overline{x}_k, \overline{y}) \in S \end{array} \iff \begin{array}{l} \exists \overline{y}_1, \ldots, \overline{y}_k.\ \overline{y}_1 + \cdots + \overline{y}_k = \overline{y} \text{ and} \\ \qquad (\overline{x}_1, \overline{y}_1), \ldots, (\overline{x}_1, \overline{y}_k) \in R \end{array} \tag{13}$$

Proof. We set T to be the projection of R onto $\mathbb{N}^A$, i.e. T is the set of vectors $\overline{x} \in \mathbb{N}^A$ such that $(\overline{x}, \overline{y}) \in R$ holds for some $\overline{y} \in \mathbb{N}^B$.

We now define what vectors $\overline{z}$ belong to S. Let the linear sets comprising R be $R_1, \ldots, R_k$. Let $\overline{v}_1, \ldots, \overline{v}_m$ an enumeration of all the defining vectors of R, i.e. the defining vectors of $R_1, \ldots, R_k$. By assumption on R being in normal form, these vectors have disjoint nonzero domains. In particular, every vector $\overline{z} \in \mathbb{N}^A \times \mathbb{N}^B$ has at most one decomposition:

$$\overline{z} = a_1\overline{v}_1 + \cdots + a_m\overline{v}_m \ . \tag{14}$$

In the above decomposition, we say a vector $\overline{v}_i$ is *used* if a_i is nonzero. In order to satisfy the right hand side of (13), a vector must admit such a decomposition. Furthermore, it must also satisfy:

> (*) Let R_i be one of the linear sets $R_1, \ldots, R_k$. If a period vector from R_i is used, then the base vector of R_i is also used.

The above two conditions form the definition of the set S from (13): it is defined to be the set of vectors that admit a decomposition as in (14), and where property (*) is satisfied. The set S is semilinear, since its definiton can be formalized by a Presburger formula.

We now proceed to show that the above defined set satisfies the equivalence (13). The right-to-left implication is immediate; we only do the left-to-right implication here.

Let then $\overline{x}_1, \ldots, \overline{x}_n \in \mathbb{N}^A$ and $\overline{y} \in \mathbb{N}^B$ be such that $S(\overline{x}_1 + \cdots + \overline{x}_n, \overline{y})$ is satisfied. We need to find a decomposition

$$\overline{y}_1 + \cdots + \overline{y}_n = \overline{y}$$

such that the left-hand side of (13) is satisfied. The proof is by induction on n. For $n = 0$ the statement is trivially true.

Consider now the vector $\overline{x}_1 \in \mathbb{N}^A$. By assumption on R being in normal form, there is at most one linear set R_i such that $(\overline{x}_1, \overline{y}_1) \in R_i$ may hold, for some $\overline{y}_i \in \mathbb{N}^B$. We partition the period vectors of R_i into two sets: W_i containing the defining vectors with nonzero values on coordinates from A; and V_i containing the remaining defining vectors of R_i. Consider now a vector $\overline{y}_1 \in \mathbb{N}^B$ such that $(\overline{x}_1, \overline{y}_1) \in R_i$ holds, and its (unique) decomposition

$$(\overline{x}_1, \overline{y}_1) = b_1\overline{v}_1 + \cdots + b_m\overline{v}_m \ . \tag{15}$$

We partition the coordinates $b_1, \ldots, b_m$ into three groups. First there are the coordinates corresponding to vectors from W_i, these are uniquely determined by the value $\overline{x}_1$ and do not depend on the choice of $\overline{y}_1$. Then there are the coordinates corresponding to vectors from V_i, these are dependent on both $\overline{x}_1$ and $\overline{y}_1$. Finally, there are the coordinates corresponding to defining vectors of the linear components other than R_i, these have all value 0 regardless of the choice of $\overline{y}_1$. Choose then $\overline{y}_1$ so that the decomposition as in (15) of the vector

$$(\overline{x}_1 + \cdots + \overline{x}_m, \overline{y}) - (\overline{x}_1, \overline{y}_1)$$

does not use any vectors from V_i. This way, the above vector satisfies (*), and therefore it belongs to S, so the inductive assumption can be applied. □

Proof of Proposition 64 We now conclude the proof of Proposition 64, under the assumption that the relation R is in normal form. Since R is in normal form, we can apply Lemma 7, and replace the set in the statement of Proposition 64 with

$$\begin{aligned}\{\overline{y} :\ & \text{there exist } \sigma_1, \ldots, \sigma_k \text{ such that} \\ & K \text{ contains an unordered combination of } \sigma_1, \ldots, \sigma_k \text{ and} \\ & \pi(\sigma_1), \ldots, \pi(\sigma_k) \in T \text{ and } (\pi(\sigma_1) + \cdots + \pi(\sigma_k), \overline{y}) \in S\}\end{aligned}$$

Since the last line above is a condition on the Parikh images of $\sigma_1, \ldots, \sigma_n$ and their combination, the above set is the same as:

$$\{\overline{y} : \exists w \in \mathit{ucshuffle}_A(L) \cap K \ .(\pi(w), \overline{y}) \in S\} \ , \tag{16}$$

where $L \subseteq A^*$ is the set of words with a Parikh image in T. We will show that the set

$$\{\pi(w) : w \in \mathit{ucshuffle}_A(L) \cap K\}$$

is semilinear, and therefore so is (16), its image under the semilinear relation S. (Recall that the subscript A in $\mathit{ucshuffle}_A(L)$ means that each segment must belong to A, i.e. have only one letter.)

But this last step immediately follows from Proposition 63. Indeed, by this proposition, the language $\mathit{ucshuffle}_A(L)$ is recognized by a Presburger automaton. Since Presburger automata are closed under intersection with regular languages, then the language

$$\mathit{ucshuffle}_A(L) \cap K$$

is also recognized by a Presburger automaton and therefore has a semilinear Parikh image.

6.2 UCSE for Parikh languages

In this section, we prove Proposition 63. Recall the statement of that result:

> Let $L \subseteq A^*$ be a language with a semilinear Parikh image. The language $\mathit{ucshuffle}_A(L)$ is definable by a Presburger automaton.

The following lemma shows that Proposition 63 is closed under finite unions:

Lemma 8. *Let $L_1, \ldots, L_k \subseteq A^*$ be Parikh languages. If each $\mathit{ucshuffle}_A(L_i)$ is definable by a Presburger automaton, then so is $\mathit{ucshuffle}_A(L_1 \cup \cdots \cup L_k)$.*

Proof. The Presburger automaton simulates all the automata for $\mathit{ucshuffle}_A(L_i)$ in parallel. It guesses for each position in the word which language L_i corresponds to that position. □

By the above lemma, it is enough to consider the case when the Parikh image of L is linear. Our objective is to define a Presburger automaton for $\mathit{ucshuffle}_A(L)$. Since L is linear and only the Parikh image of L is relevant, we may assume that L is of the form:

$$w_0(w_1 + \cdots + w_n)^* \qquad \text{with } w_0, w_1, \ldots, w_n \in A^* \ .$$

(Here the word w_0 corresponds to the base vector, and the words $w_1, \ldots, w_n$ correspond to the period vectors.) We first define the automaton, and then prove it's correctness.

Lemma 9. *Without loss of generality, we may assume that all the words $w_0, w_1, \ldots, w_n$ use disjoint alphabets, and no letter is used twice.*

Proof. The technique is the same as in the proof of Lemma 6. Another argument is that in our construction, Proposition 63 is only applied when the Parikh image of L is in normal form, which assures the statement of the lemma. □

We use A_i to denote the labels occurring in the word w_i. We use the name *dog labels* for the letters in the word w_0 and the name *sheep labels* for the letters from the words $w_1, \ldots, w_n$.

Consider the following constant:

$$N = 7|A|^2 \tag{17}$$

Using the same type of argument as in the proof of Lemma 8, it is sufficient to construct a Presburger automaton $\mathcal{A}$ that recognizes those words in $\mathit{ucshuffle}_A(L)$ where each label from A is used at least N times. Indeed, when some label is used at most N times, then the at most N words from L that use this label can be specially marked by the Presburger automaton and treated separately.

The Presburger automaton $\mathcal{A}$ we define works as follows: it checks that for each $i = 0, \ldots, n$, every two letters from w_i occur the same number of times (and at least N times).

Clearly the above automaton accepts all words from $\mathit{ucshuffle}_A(L)$ where each label occurs at least N times. We need to show that it does not accept anything else. Let then $a_1 \cdots a_k$ be a word accepted by $\mathcal{A}$. We claim, that for each $i = 0, \ldots, k$ we can find a coloring

$$\alpha : \{1, \ldots, k\} \to \Delta \qquad |\Delta| \geq N$$

of the positions $a_1, \ldots, a_k$ such that the following invariant is satisfied:

– For each color $d \in \Delta$, the positions assigned label d are a permutation of some word in L.
– At most N colors $d \in \Delta$ are used for positions with sheep labels. Moreover, no two successive positions with sheep labels have the same color.
– No two successive positions in $\{1, \ldots, i\}$ are assigned the same color.

For $i = k$, the invariant clearly implies that the word belongs to $\mathit{ucshuffle}_A(L)$. Before we show that the invariant can be satisfied, we need to introduce some auxiliary notation and results.

Clusters For a set $B \subseteq A$ of labels, we say a set of positions $V \subseteq \{1, \ldots, k\}$ in a word $a_1 \cdots a_k$ word *forms a B-cluster*, if the labels of these positions are exactly B (without repetitions). The first condition in the invariant can be restated in terms of clusters:

Fact 65 The set of positions $V \subseteq \{1, \ldots, k\}$ that are assigned a color $d \in \Delta$ are a permutation of some word in L if and only if V can be partitioned into a disjoint union $V_0, V_1, \ldots, V_k$, where V_0 is an A_0-cluster, while each of $V_1, \ldots, V_k$ is an A_i-cluster for some $i = 1, \ldots, m$.

Recall the constant N defined in (17).

Lemma 10. *Let $B \subseteq A$ and let $Z \subseteq \{1, \ldots, k\}$ be a union of at least N disjoint B-clusters. One can assign at most N colors to the positions in Z so that each two successive positions in Z have different colors, and each class is a disjoint union of B-clusters.*

Proof. We will construct a partial coloring $\beta : Z \to \{1, \ldots, N\}$, so that each two successive positions in the domain of Z have different colors, and each class (i.e. set of positions from Z that are assigned the same color) is a disjoint union of B-clusters. At the end, the domain of β will be all of Z, thereby proving the lemma.

Claim. Let $Y \subseteq Z$ be a union of more than $2|B|$ disjoint B-clusters. There is a B-cluster $X \subseteq Y$ that does not contain two successive positions.

Proof. By successively adding positions to X, and avoiding neighbors of the already chosen positions. □

Since Z contains $N > 2|B| + 4|B|$ clusters, we can iterate the above claim $4|B|$ times, and find a partial coloring β whose domain has at $4|B|$ clusters, each of which is assigned a different color.

To this assignment β we will now successively add B-clusters, until its domain exhausts all of Z. During the process, we will always make sure that β uses at least $4|B|$ colors. Let $Y \subseteq Z$ be the domain of β, and let $X \subseteq Z$ be a B-cluster disjoint with Y. We will add X to the domain of β. The problem is that X might contain two successive positions, so some swapping with elements already in Y might be necessary.

Let Γ be the set of colors that are used for neighbors of the positions in X. This set has at most $2|B|$ colors.

Claim. We can find nonneighboring positions y_b, one for each label b, such that each y_b has a label b and a color outside Γ.

Proof. We use the assumption that β uses many colors. We successively chose the positions y_b, avoiding neighbors of the already chosen positions. □

Let y_b be as in the claim above and let d be a color that does not occur in any of the at most $2|B|$ neighbors of the nodes y_b. By assumption on $N > 2|B|$, the color d can be found. We modify the coloring β in the following way. We add X to domain of β. For each $b \in B$, we assign d to the position y_b, while to the (unique) b-labeled position in the cluster X we assign the old color of y_b. □

Proof of the invariant We first show the invariant for $i = 0$, and then show how to inductively advance it from i to $i + 1$.

Thanks to the counting properties verified by $\mathcal{A}$, we know that a coloring α can be found so that the first and third properties of the invariant are satisfied for $i = 0$. The second condition is a consequence of Lemma 10.

We now proceed to show the inductive step, and assume that $\alpha(i) = \alpha(i + 1)$ (otherwise we keep α as it is and proceed with $i+1$). Thanks to the second item in the invariant, either i or $i+1$ must have a dog label. We assume that i has a dog label, the other case is resolved in the same way. Let d be the color $\alpha(i)$. In a first step, we make sure that no position with a sheep label is assigned the color d:

Lemma 11. *The coloring α can be modified so that the invariant is still satisfied for i, and moreover the class of position i contains no sheep labels.*

Before we proceed with a proof of this lemma, we show how it can be used to complete the induction step. Thanks to the above lemma, the class of d is simply an A_0 cluster V. If after applying Lemma 11, the colors in position i and $i+1$ are still different, then the label a of position $i + 1$ must be a dog label. Let W be all the occurrences of a in the word. Since V contains $|A_0|$ positions and W contains at least $N > 2|A_0| + 1$ positions, there must be some position $j \in W$ that is not adjacent to a position in V, and is not assigned the same color as position i. We swap the colors for $i + 1$ and j. By choice of j, the invariant is no satisfied for $i + 1$.

We now show Lemma 11.

Proof (of Lemma 11). Let d be the color $\alpha(i)$. Assume that i contains sheep labels. We partition the class $\alpha^{-1}(d)$ into clusters $V_0, \ldots, V_n$ as in Fact 65. Here V_0 are the dog labels, and $V_1, \ldots, V_n$ are the sheep labels.

The general idea is that we pick any other color e that does not have sheep letters, and assign the sheep letters in the class $\alpha^{-1}(d)$ to the color e. In the process, however, we need to be careful not to introduce any violations of the neighborhood condition (i.e. conditions 2 and 3 of the invariant).

Let then e be some color such that the class $\alpha^{-1}(e)$ does not contain any sheep labels, and is therefore an A_0-cluster W. Such a color must exists by the assumption that few colors are used for sheep labels. Let $\mathcal{V}$ be all the clusters among $V_1, \ldots, V_n$ that contain a position adjacent to one of the positions in W. For positions from $V_1, \ldots, V_n$ that are not from $\mathcal{V}$, we pick the color e.

It remains to pick a color for the positions from $\mathcal{V}$. Clearly, $\mathcal{V}$ contains at most $2|A_0|$ clusters, and therefore involves at most $2|A_0||A|$ positions, with at most $4|A_0||A|$ neighbors. As long as we do not use the colors from the $4|A_0||A|$ neighbors, we will not violate the neighborhood condition. Since there are at least

$$N \geq 4|A_0||A| + 3$$

colors available for sheep labels, we can therefore pick a new color other than d, e for the positions in $\mathcal{V}$ so that none of the positions in $\mathcal{V}$ violate the neighborhood conditions,

and the second item of the invariant is satisfied. (Note that no two positions from $\mathcal{V}$ are neighboring, since otherwise the original assignment α would not have satisfied condition 2 of the invariant.) □

We conclude by summarizing the expressive power of the expressions:

$$\text{USE} \subsetneq \text{UCSE} \subsetneq \text{SE} = \text{CSE} .$$

The strictness of the first inequality is not shown due to lack of space. The second inequality follows by undecidability of SE, while the equality was mentioned in Theorem 41.

References

1. H. Björklund and T. Schwentick. On notions of regularity for data languages. In *FCT'07*, 2007. To appear.
2. M. Bojańczyk, A Muscholl, T. Schwentick, L. Segoufin, and C. David. Two-variable logic on data trees and xml reasoning. In *PODS'06*, pages 10–19, 2006.
3. M. Bojańczyk, A Muscholl, T. Schwentick, L. Segoufin, and C. David. Two-variable logic on words with data. In *LICS'06*, pages 7–16, 2006.
4. P. Bouyer, A. Petit, and D. Thérien. An algebraic approach to data languages and timed languages. *Information and Computation*, 182(2):137–162, 2003.
5. S. Demri and R. Lazić. LTL with the freeze quantifier and register automata. In *LICS'06*, pages 17–26, 2006.
6. J. Gischer. Shuffle languages, petri nets, and context-sensitive grammars. *Communications of the ACM*, 24(9):597–605, 1981.
7. J. Jedrzejowicz and A. Szepietowski. Shuffle languages are in P. *TCS*, 250:31–53, 2001.
8. M Kaminski and N. Francez. Finite-memory automata. *TCS*, 132(2):329–363, 1994.
9. I.A. Lomazova and P. Schnoebelen. Some decidability results for nested petri nets. In *PSI'99*, pages 208–220, 2000.
10. F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM transactions on computational logic*, 15(3):403–435, 2004.
11. R. Parikh. On context-free languages. *Journal of the ACM*, pages 570–581, 1966.
12. L. Segoufin. Automata and logics for words and trees over an infinite alphabet. In *Computer Science Logic (CSL)*, volume 4207 of *LNCS*, pages 41–57, 2006.