Efficient recognition of trace languages defined by repeat-until loops

Luca Breveglieri[(1)] Stefano Crespi Reghizzi[(1)]
Massimiliano Goldwurm[(2)]

(1) Dipartimento di Elettronica e Informazione, Politecnico di Milano,
via Ponzio 34/5, 20133 Milano – Italy {luca.breveglieri, stefano.crespireghizzi}@polimi.it

(2) Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano
Via Comelico 39-41, 20135 Milano – Italy, goldwurm@dsi.unimi.it

November 2009

Abstract

A sequence of operations may be validly reordered, provided that only pairs of independent operations are commuted. Focusing on a program scheme, idealized as a local finite automaton, we consider the problem of checking whether a given string is a valid permutation of a word recognized by the automaton. Within the framework of trace theory, this is the membership problem for rational trace languages. Existing general algorithms, although time-polynomial, have unbounded degree related to some properties of the dependence graph. Here we present two original linear-time solutions. A straightforward algorithm is suitable for any finite automaton such that from any state only transitions labelled by dependent symbols are allowed. The second approach is currently restricted to automata representing programs of nested repeat-until loops. Using integer compositions to represent loop iterations and under suitable conditions, the algorithm constructs the syntax tree of a possible word equivalent to the input string. The same procedures show that, under our hypotheses, the uniform version of the membership problem (which is NP-complete in the general case) is solvable in polynomial time. [1]

Keywords: automata and formal languages, trace languages, local finite automata, integer compositions, dependencies checking.

1 Introduction

A sequence of operations may be validly reordered, provided that only pairs of independent operations are commuted. For instance, a computer program can be idealized as a deterministic finite automaton (DFA), recognizing a set of strings from an alphabet of abstract instructions, some of which are mutually dependent. At a coarser granularity of operations, the same problem

[1] Appeared in revised form in *Information and Computation*, vol. 208, pp. 969-981, 2010.
This work has been supported by Project MIUR PRIN 2007 "Mathematical aspects and forthcoming applications of automata and formal languages".

occurs when concurrent accesses to a database are serialized. Formally, the problem we address is the following: Is a given string a valid permutation of a word recognized by the automaton?

A motivation comes from the areas of compiler and processor design. Modern compilers [9] and processors [13] reorder ("reschedule") machine instructions, with respect to the original sequential program ordering, with the goal of minimizing program completion time by taking advantage of available hardware parallelism. This task involves the capability to check that instruction dependencies are not violated. We present two very efficient algorithms under different assumptions.

Trace theory is a convenient framework for stating and analyzing dependencies checking problems. We recall that trace languages were introduced in the '70s as a tool for the study of concurrent systems [12] and a comprehensive treatment of their properties and related theory is presented in [8].

Using concepts from formal language theory, the program scheme is the state-transition graph of a DFA, and since in a program each instruction differs from any other by its memory address, the DFA can be assumed to be a local one [2]. Then the previous problem is the membership problem for the trace language represented by a local DFA.

In the past a few algorithms [5, 3, 1] have been proposed for *rational* trace languages, i.e., partially commutative languages represented by a regular string language, and also for context-free trace languages [14, 5]. Such algorithms examine the prefixes of a trace, also using sophisticated data structures for a more efficient implementation. Their time complexity, although polynomial, has an unbounded degree, which is related to some properties of the independence graph. More precisely the degree is given by the size of the largest clique of the independence relation, a value which is likely to be too large for realistic application. Here we present two efficient linear-time solutions of the membership problem for rational trace languages represented respectively by automata with dependent transitions and by the so-called repeat-until expressions.

We also consider the uniform version of our membership problem, where both the independence alphabet and the automaton are part of the input. In this case the problem is known to be NP-complete [5]. However, our two algorithms easily show that under the given hypotheses the problem can be solved in polynomial time.

After the basic definitions of Section 2, we present in Section 3.1 a straightforward algorithm, suitable for any DFA satisfying the following hypothesis: for any pair of transitions from the same state, either the corresponding labels are dependent, or they lead to two states that are not mutually reachable. These conditions correspond to rather realistic assumptions for frequent program patterns.

The second approach is much more involved and takes the rest of the paper. The problem definition and some of the initial ideas and properties come from [7, 16], but the mathematical setting based on integer compositions and the problem solving strategy are new. This approach is currently restricted to nested repeat-until loops, a popular class of computationally intensive program structures, here represented by a family of regular expressions (called repeat-until expressions) defined in Section 4. In the same section we introduce a simple *closure assumption*, which concerns the dependencies between nested loops. Then, in Section 5, we recall the notion of integer composition with some particular properties, showing how they can be used to represent the syntax tree of a word in the language defined by a repeat-until expression.

Section 6 focuses on the membership problem for trace languages represented by such repeat-until string languages. First, we prove a theorem relating the existence of a syntax tree to certain labelled compositions, which can be derived by observing the runs of dependent letters. Based on

that, we then define and analyze an efficient algorithm, which repeatedly applies the previously introduced product and quotient operations on the integer compositions. Under the closure assumption this algorithm solves the problem in linear time.

The previous algorithms can also be used for the uniform version of the membership problem showing that, under our particular assumptions, the problem is solvable in polynomial time. The details are given in Section 3.2 for languages defined by automata with dependent transitions and in Section 6.4 for languages defined by repeat-until expressions.

At last we discuss the meaning and the limits of such a closure hypothesis, and in the conclusion we hint to possible generalizations and alternative assumptions.

2 Basic notions

Given a finite alphabet Σ and a word $x \in \Sigma^*$, $|x|$ represents the length of x while, for each $y \in \Sigma^+$, $|x|_y$ denotes the number of occurrences of y in x. Moreover, given a subset $A \subseteq \Sigma$, $\pi_A(x)$ is the projection of x over A. Further, if x is not the empty word ε, $P(x)$ and $U(x)$ denote, respectively, the first and the last symbol of x, while $S_1(x)$ is the suffix of x of length $|x| - 1$.

We recall that an independence relation on Σ is a symmetric and irreflexive relation $I \subseteq \Sigma \times \Sigma$. For every $a, b \in \Sigma$ we say that a and b are independent if $(a, b) \in I$ and in this case we also write aIb. The dependence relation D is the complement of I and also in this case aDb stands for $(a, b) \in D$. The pair (Σ, I) is called independence alphabet and it is usually represented by an undirected graph where Σ is the set of nodes and I the set of edges. An independence relation I establishes an equivalence relation $\equiv_I$ on Σ^* as the reflexive and transitive closure of the relation $\sim_I$ defined by

$$xaby \sim_I xbay \qquad \forall x, y \in \Sigma^*,\ \forall (a, b) \in I.$$

The relation $\equiv_I$ is a congruence over Σ^*, i.e. an equivalence relation preserving concatenation between words. For every $x \in \Sigma^*$ the equivalence class $[x] = \{y \in \Sigma^* \mid y \equiv_I x\}$ is called *trace*, the quotient monoid $\Sigma^*/\equiv_I$ is called trace monoid and usually denoted by $M(\Sigma, I)$. A subset $T \subseteq M(\Sigma, I)$ is called trace language and, for every $L \subseteq \Sigma^*$, we define $[L] = \{[x] \in M(\Sigma, I) \mid x \in L\}$ as the trace language represented by L. A trace language is called *rational* if it is represented by a regular language. Moreover, the rational operations on trace languages (union, product and Kleene closure) are defined as in the free monoids. It is known that the class of rational trace languages is the smallest family of trace languages including the finite sets in $M(\Sigma, I)$ and closed under the rational operations. Moreover a rational trace language $T \subseteq M(\Sigma, I)$ is called *unambiguous* if $T = [L]$ for a regular language $L \subseteq \Sigma^*$ such that, for every $t \in T$, there is exactly one string $x \in L$ belonging to t. The class of unambiguous rational trace languages can be characterized by the so-called unambiguous rational operations and it coincides with the class of all rational trace languages if and only if the independence relation I is transitive [3, 4, 15].

3 Membership problem for rational trace languages

The main topic we consider in this work is the recognition of rational trace languages. Formally, given an independence alphabet (Σ, I) and a finite automaton recognizing a language $L \subseteq \Sigma^*$,

the membership problem for $[L]$ consists of deciding, for an input $x \in \Sigma^*$, whether there exists a word in $[x] \cap L$ (i.e. whether $[x]$ belongs to $[L]$).

For this problem there is a general algorithm that works in time $O(n^c)$ for any input of size n, where c is the size of the maximum clique in (Σ, I) [5]. A similar procedure is studied in [1] in the average case, assuming that all input strings of given length are equiprobable; under this assumption the procedure works in time $O(n^k)$ in the average case, where k is the number of connected components of the dependence graph (Σ, D). Here, we first determine a family of independence alphabets and finite automata for which the corresponding membership problem can be solved in linear time.

3.1 Automata with dependent transitions

Consider a (deterministic) finite state automaton $\mathcal{A} = (Q, q_0, \delta, F)$ over an alphabet Σ, where Q is the set of states, q_0 is the initial state, $\delta : Q \times \Sigma \to Q \cup \{\perp\}$ is the (partially defined) transition function and $F \subseteq Q$ is the set of final states. For every $q \in Q$ we define the set $Tra(q) = \{a \in \Sigma \mid \delta(q, a) \neq \perp\}$.

Now, given an independence alphabet (Σ, I) with dependent relation D, we say that the above automaton $\mathcal{A} = (Q, q_0, \delta, F)$ has *dependent transitions* if, for every $q \in Q$ and every $a, b \in Tra(q)$, we have aDb. A first consequence of this definition is given by the following

Proposition 1 *For any independence alphabet (Σ, I), if $L \subseteq \Sigma^*$ is recognized by a finite automaton with dependent transitions then the rational trace language $[L]$ is unambiguous.*

Proof. In fact, let $\mathcal{A} = (Q, q_0, \delta, F)$ be a finite automaton with dependent transitions recognizing L. Assume there are two words $u, v \in L$ such that $[u] = [v]$. Consider the longest common prefix x of u and v. Then $u = xaz$ and $v = xbw$ for some aIb and $z, w \in \Sigma^*$. We have that $a, b \in Tra(q)$ where $q = \delta(q_0, x)$ and hence $\mathcal{A}$ has a pair of independent transitions, which is contradiction. □

Moreover, it turns out that if $L \subseteq \Sigma^*$ is the language accepted by an automaton $\mathcal{A}$ with dependent transitions, then the trace language $T = [L]$ can be recognized in linear time. To describe the algorithm, let us represent by Δ the set of all pairs and singletons forming a covering of the dependence graph (Σ, D):

$$\Delta \;=\; \{\{a, b\} \mid a, b \in \Sigma,\ aDb\} \;\cup\; \{\{b\} \mid b \in \Sigma,\ \forall a \in \Sigma\ a \neq b \Rightarrow aIb\}$$

Recall that, for every $x, y \in \Sigma^*$ we have $x \equiv_I y$ if and only if $\pi_\ell(x) = \pi_\ell(y)$ for all $\ell \in \Delta$. Moreover, for every $a \in \Sigma$, we denote by $\Delta(a)$ the set $\Delta(a) = \{\ell \in \Delta \mid a \in \ell\}$. It is clear that the sets Δ and $\Delta(a)$, $a \in \Sigma$, only depend on the independence alphabet (Σ, I) and can be computed in a preprocessing phase.

For a given input $x \in \Sigma^+$ the procedure computes a word $z \equiv_I x$ accepted by $\mathcal{A}$, if any, otherwise it returns 0. The procedure maintains a family of strings $\{y_\ell : \ell \in \Delta\}$, where at the beginning $y_\ell = \pi_\ell(x)$ for every ℓ, and simulates a possible computation of $\mathcal{A}$ on z. A state $q \in Q$ is updated which represents the current state of the computation. In the main iteration one looks for a letter $a \in Tra(q)$ that occurs as first symbol in all y_ℓ with $a \in \ell$: by the hypothesis of dependent transitions, there is at most one symbol a satisfying that condition; in this case $\delta(q, a)$ becomes the new current state and the projections y_ℓ with $a \in \ell$ are updated. This process is iterated until either all projections y_ℓ are empty or no new symbol a can be found which satisfies the condition above. The input is accepted if and only if, the last current state is final and all y_ℓ's are empty.

```
begin
      for ℓ ∈ Δ  do y_ℓ := π_ℓ(x)
      q := q_0
      z := ε
      while ∃a ∈ Tra(q) such that a = P(y_ℓ) for all ℓ ∈ Δ(a)
                                  do { q := δ(q, a)
                                     { z := za
                                     { for ℓ ∈ Δ(a) do y_ℓ := S_1(y_ℓ)
      if q ∈ F ∧ y_ℓ = ε for all ℓ ∈ Δ
                                  then return z
                                  else return 0
end
```

For any input x of length n the algorithm works in $O(n)$ time since the cardinality of Δ only depends on the dependence relation and hence the main iteration requires $O(1)$ time.

Proposition 2 *Given an independence alphabet (Σ, I), let $L \subseteq \Sigma^*$ be accepted by a finite automaton with dependent transitions. Then, the trace language $[L]$ can be recognized in time $O(n)$.*

Another condition on $\mathcal{A}$ and D that allows us to design a linear time algorithm to recognize $[L]$ is the following: for every $q \in Q$ and every pair of independent symbols b, c in $Tra(q)$, either state $\delta(q, b)$ or state $\delta(q, c)$ is not reachable from the other one. Thus, if q is the current state and both b and c appear as first symbol in the corresponding projections (since bIc they do not share a common projection), then the procedure is forced to choose the transition leading to the state able to reach the other one. Note that the previous condition is milder than the hypothesis of dependent transitions.

The hypotheses of dependent transitions and non-mutual reachability we consider above occur frequently in program schemes. For instance, the first condition states that the true and false successors of a conditional instruction are dependent on each other. This also happens in the common case when the successors are instructions assigning different values to the same variable, a case of write-after-write data-dependence. Also the situation where one of the successors does not reach the other is typical of rather frequent program patterns: for instance, it occurs in the case of a conditional jump raising an exception, such that the normal execution is abandoned if the exception is verified.

3.2 Uniform membership problem

The problem defined in the previous section can also be restated in a uniform version, where both the independence alphabet (Σ, I) and the (deterministic) finite automaton $\mathcal{A}$ are part of the input. In this case the problem becomes NP-complete with a reduction from the Hamiltonian Circuit Problem [5]. However, it is easy to see that if the automaton $\mathcal{A}$ has dependent transitions then the algorithm described in the previous section solves the problem in polynomial time.

To yields a more precise evaluation, let m, s, n be respectively the cardinality of Σ, the number of states of $\mathcal{A}$ and the length of x. Using matrices to represent both $\mathcal{A}$ and (Σ, I) one can verify in $O(sm^2)$ time whether $\mathcal{A}$ has dependent transitions and then apply the algorithm of the previous section to check whether $[x] \cap L(\mathcal{A})$ is empty, which in this context requires $O(m^2 n)$

time. Hence the overal time cost is $O(m^2(s+n))$, which can be considered as a cubic time with respect to the input size. Slight modifications are obtained if we use adjacency lists rather than matrices to represent the automaton $\mathcal{A}$ and the graph (Σ, I).

4 Repeat-until languages

In this section we define a family of expressions representing a program scheme consisting of nested repeat-until cycles (or loops). Intuitively, they are regular expressions without union symbols, where each letter of the alphabet occurs just once.

Given a finite alphabet Σ, let N be the set of all expressions defined as follows:

i) Every $a \in \Sigma$ belongs to N;

ii) If $\alpha, \beta \in N$ then $\alpha \cdot \beta \in N$ (also denoted by $\alpha\beta$);

iii) If α is a symbol in Σ or an expression $\beta \cdot \gamma$, for some $\beta, \gamma \in N$, then $(\alpha)^+ \in N$.

An element $\alpha \in N$ is called *repeat-until expression* over Σ if it contains just one occurrence of a for every $a \in \Sigma$. Thus, $\pi_\Sigma(\alpha)$ defines a linear order over Σ and, for every $a, b \in \Sigma$, we write $a < b$ if a occurs before b in $\pi_\Sigma(\alpha)$. The set of all repeat-until expressions over Σ will be denoted by RUE(Σ), or simply by RUE when Σ is understood.

For every $\alpha \in$ RUE, let $L(\alpha)$ be the language represented by α. Clearly, for every $x \in L(\alpha)$ and every $a, b \in \Sigma$, we have

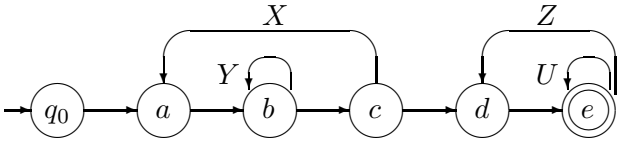
$$a < b \text{ implies } \pi_{a,b}(x) \in a\{a,b\}^*b \tag{1}$$

Moreover, we define a *cycle* of α as a subexpression $(\beta)^+$ of α such that $\beta \in N$ (note that β is not of the form $(\gamma)^+$). The string $\pi_\Sigma(\beta)$ is the *body* of the cycle, $P(\pi_\Sigma(\beta))$ and $U(\pi_\Sigma(\beta))$ are its *header* and *exit*, respectively. For instance, $((ac)^+bd(e)^+)^+$ is a cycle of $\alpha = h((ac)^+bd(e)^+)^+fg$, with header a and exit e.

Note that in every $x \in L(\alpha)$ the body of any cycle appears at least once, possibly as a subword consisting of several nonoverlapping factors. This justifies our definition: any $\alpha \in$ RUE represents a program scheme of nested repeat-until cycles and every $x \in L(\alpha)$ represents an execution of the program.

Clearly, for any $\alpha \in$ RUE, $L(\alpha)$ is a local language and the corresponding finite automaton is obtained by a standard construction [2], where there is an initial state q_0 and a state for each symbol in Σ. Here we avoid the easy formal definition and describe such automaton by an example. For our subsequent discussion, in the diagram of these automata it is convenient to represent cycles by capital letters.

Example 1 *Consider the repeat-until expression $\alpha = (a(b)^+c)^+(d(e)^+)^+$. Then, the corresponding local automaton $\mathcal{A}(\alpha)$ is defined by the set of states $Q = \{q_0, a, b, c, d, e\}$ together with following transition diagram:*



where X, Y, Z, U represent the cycles $(a(b)^+c)^+$, $(b)^+$, $(d(e)^+)^+$ and $(e)^+$, respectively.

Observe that if, for any $q \in Q$, the family $Tra(q) = \{a \in \Sigma \mid \delta(q,a) \neq \perp\}$ is a clique of a given dependence graph, then the trace language $[L(\alpha)]$ is recognizable in $O(n)$ time by the algorithm presented in Section 3.1.

4.1 Hierarchical trees

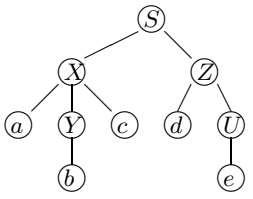
Here we describe a tree representation of a repeat-until expression. Let us first recall that a plane tree is a rooted tree where the sons of every internal node are totally ordered (usually drawn from left to right). This clearly induces a natural total order on the leaves of the tree. Now, given $\alpha \in \mathrm{RUE}(\Sigma)$, let $\mathcal{C}$ be the family of all cycles of α (denoted by capital letters) together with a special symbol S, which will represent the root of the tree. For every $X, Y \in \mathcal{C}$, we define $X \trianglelefteq Y$ if X is nested into Y or $X = Y$. We also set $X \trianglelefteq S$ for every $X \in \mathcal{C}$. Moreover, we write $X \triangleleft Y$ if $X \trianglelefteq Y$ and $X \neq Y$.

Then we define the *hierarchical tree* of α as the plane tree $T(\alpha)$ with root S, satisfying the following properties:

1. $\mathcal{C}$ is the set of internal nodes and Σ the set of leaves;
2. For any $X, Y \in \mathcal{C}$, X is son of Y if $X \triangleleft Y$ and X is immediately nested in Y (i.e. there is no $Z \in \mathcal{C}$ such that $X \triangleleft Z \triangleleft Y$) and in this case we also denote Y by $F(X)$;
3. A leaf $a \in \Sigma$ is son of a node $X \in \mathcal{C}$ if X is the smallest cycle of α including a. If a is not included in any cycle then a is son of S;
4. For every node $X \in \mathcal{C}$ and every two sons u, v of X we set $u < v$ if u (either as a cycle or as a letter in Σ) occurs before v in α.

Note that $X \trianglelefteq Y$ holds if and only if X is descendant of Y in $T(\alpha)$.

Example 2 *The hierarchical tree of the repeat-until expression α defined in Example 1 is described by the following picture.*



For every $a \in \Sigma$, let $C(a)$ be the father of a in $T(\alpha)$: thus $C(a)$ either is the smallest cycle of α containing a or $C(a) = S$ if a is not included in any cycle. Analogously, for every $a, b \in \Sigma$, $a \neq b$, let $C(a,b)$ be the root of the smallest subtree of $T(\alpha)$ including both a and b. It is easy to prove the following proposition stating that all cycles are of the form $C(a)$ or $C(a,b)$ for some $a, b \in \Sigma$.

Proposition 3 *Let $\alpha \in RUE(\Sigma)$ and let $X \in \mathcal{C}$ be a symbol different from S. Then, $X = C(a)$ for some $a \in \Sigma$ or $X = C(a,b)$ for some distinct $a, b \in \Sigma$.*

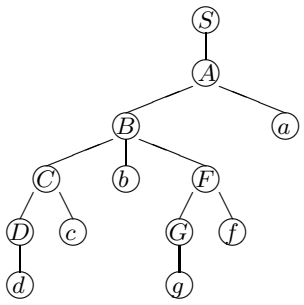
In this work we are mainly interested in those repeat-until expressions that satisfy a further "closure" condition with respect to a given independence alphabet. To state such a notion, consider an expression $\alpha \in \mathrm{RUE}(\Sigma)$ and let D be a dependence relation on Σ. We define the adjacency relation over $\mathcal{C}$ as the binary relation given by the set

$$Adj = \{(Y, X) \mid \exists aDb : Y = C(a, b), X = C(a)\}$$

Moreover we denote by Con the reflexive and transitive closure of Adj.

Definition 1 *For any expression $\alpha \in RUE(\Sigma)$ and any dependence relation D over Σ, we say that α (or $T(\alpha)$) is* closed *with respect to D if, for every $(Y, X) \in Adj$ and every cycle B such that $X \lhd B \lhd Y$, we have $(B, X) \in Con$.*

Example 3 *Let $T(\alpha)$ be the hierarchical tree defined by the following picture:*



Then $T(\alpha)$ is closed with respect to the following dependence relations: $D = \{\{a, d\}, \{b, d\},$- $\{c, d\}\}$, $D = \{\{a, d\}, \{c, f\}, \{c, d\}\}$, $D = \{\{a, g\}, \{b, f\}, \{g, f\}\}$, $D = \{\{d, c\}, \{f, g\}, \{c, g\}, \{a, d\}\}$. On the contrary, the same $T(\alpha)$ is not closed with respect to the dependence relations given by $D = \{\{a, c\}, \{a, b\}\}$, $D = \{\{a, d\}, \{b, c\}\}$, $D = \{\{c, f\}, \{c, g\}\}$, $D = \{\{a, c\}, \{b, d\}\}$.

4.2 Syntactic trees

Plane trees can also be used to describe the generation of a word in a language represented by a repeat-until expression. To describe the grammar, given $\alpha \in \mathrm{RUE}(\Sigma)$, let $\mathcal{C}$ and S be defined as in the previous section. Consider the context-free grammar with regular right parts $G(\alpha)$ defined by the tuple $(\mathcal{C}, \Sigma, S, P)$, where $\mathcal{C}$ is the set of nonterminals, S is the initial symbol, Σ is the set of terminals and P is the family of productions given by

$$\begin{aligned} P \;=\; \{&(X \to \gamma) \mid X \in \mathcal{C},\ \gamma \text{ is obtained from the list of sons of } X \text{ in } T(\alpha) \\ &\text{by replacing each nonterminal } Y \in \mathcal{C} \text{ by } Y^+\} \end{aligned}$$

Example 4 *If α is defined as in Example 2 then*

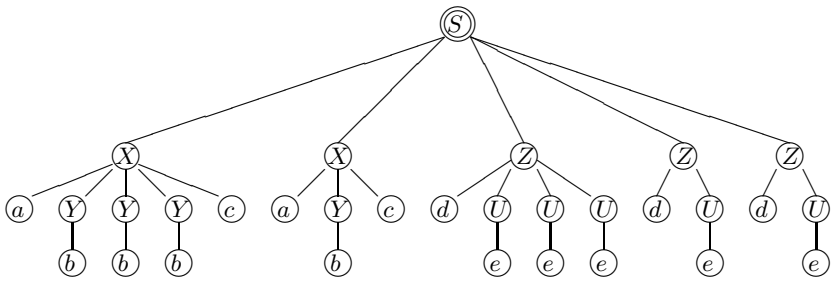
$$P = \{(S \to X^+Z^+), (X \to aY^+c), (Y \to b), (Z \to dU^+), (U \to e)\}$$

It is clear that $G(\alpha)$ generates $L(\alpha)$ in the usual way (see for instance [11]). Thus, for any $x \in L(\alpha)$ we define the *syntactic tree* of x as the derivation tree of x in $G(\alpha)$.

Example 5 *Let α be the repeat-until expression defined in Example 1 and let x be the string*

$$x \;=\; abbbcabcdeeedede$$

Then $x \in L(\alpha)$ and its syntactic tree is given by the following picture:



Proposition 4 *A word $x \in \Sigma^*$ belongs to $L(\alpha)$ if and only if there exists a syntactic tree T that generates x.*

Clearly any syntactic tree T is a plane tree. Its root is S and, for every $u \in \Sigma \cup \mathcal{C}$, T contains at least one node labelled by u: for the sake of brevity, each of them will be called *u-node*.

Observation 1 *If T is the syntactic tree of a word $w \in L(\alpha)$ then:*

1. *For every $a \in \Sigma$, $|w|_a$ equals the number of nodes of T labelled by $C(a)$;*
2. *For every $a, b \in \Sigma$ with $a < b$, $|\pi_{a,b}(w)|_{ab}$ equals the number of nodes of T labelled by $C(a, b)$.*

5 Integer compositions for tree representation

In our context, it is useful to represent syntatic trees by means of integer compositions. We recall that an *integer composition* is a finite sequence $A = (a_1, a_2, \ldots, a_h)$, where $a_i \in \mathbb{N}$ and $a_i \neq 0$ for every i, we also represent in the form $A = (a_i)_h$. The length and the sum of $A = (a_i)_h$ are defined by $\ell_A = h$ and $n_A = \sum_{i=1}^{h} a_i$, respectively.

There are two natural order relations over these structures, that we denote by $\leq$ and $\preceq$, respectively. Given $A = (a_i)_h$ and $B = (b_i)_k$, we have $A \leq B$ if $h = k$ and $a_i \leq b_i$ for each i. Moreover, $A \preceq B$ if A is finer than B, i.e. $n_A = n_B$, $k \leq h$ and there are k indices $j_1, j_2, \ldots, j_k$, $1 \leq j_1 < j_2 < \cdots < j_k = h$ such that

$$b_1 = \sum_{i=1}^{j_1} a_i \;, \quad b_2 = \sum_{i=j_1+1}^{j_2} a_i \;, \; \ldots \;, \quad b_k = \sum_{i=j_{k-1}+1}^{j_k} a_i$$

Clearly, among all integer compositions of sum n_A, the tuples $(1, 1, \ldots, 1)$ and (n_A) are the smallest and the largest element with respect to $\preceq$.

We can also define a product and a quotient operation over integer compositions. Given $A = (a_i)_h$ and $B = (b_i)_k$, if $n_A = \ell_B$ then the *product* $A \cdot B$ is given by the integer composition $C = (c_i)_h$ such that

$$c_1 = b_1 + \cdots + b_{a_1}, c_2 = b_{a_1+1} + \cdots + b_{a_1+a_2}, \; \cdots \;, c_h = b_{n_A - a_h + 1} + \cdots + b_{n_A}$$

Briefly, C is obtained from B by adding consecutive elements as indexed by the composition A. For an instance, if $A = (1, 2, 2)$ and $B = (1, 2, 1, 3, 2)$ then $C = A \cdot B = (1, 3, 5)$. It is easy to see that $n_C = n_B$, $\ell_C = \ell_A$ and $B \preceq C$. Also notice that the product is associative but not commutative. Moreover, for every composition $B = (b_j)_k$, the following identities hold:

$$(1, 1, \ldots, 1)_k \cdot B = B \qquad (k)_1 \cdot B = (n_B)_1 \qquad B \cdot (1, 1, \ldots, 1)_{n_B} = B$$

Similarly we can define the quotient of two integer compositions. Given $A = (a_i)_h$ and $B = (b_i)_k$ such that $A \preceq B$ (and hence $k \leq h$), consider the sequence of indices $j_0, j_1, \ldots j_k$ such that $0 = j_0 < j_1 < \cdots < j_k = h$ and

$$b_l = \sum_{i=j_{l-1}+1}^{j_l} a_i \qquad \text{for every } l = 1, 2, \ldots, k.$$

Then, the *quotient* B/A is the composition $C = (c_l)_k$ such that $c_l = j_l - j_{l-1}$ for every $l = 1, 2, \ldots, k$. For instance, if $B = (4, 2, 5)$ and $A = (1, 3, 2, 1, 1, 3)$ then $C = B/A = (2, 1, 3)$. Notice that we have the following special cases, for any composition $A = (a_i)_h$:

$$A/A = (1, 1, \ldots, 1)_h \qquad (n_A)_1 / A = (h)_1 \qquad A/(1, 1, \ldots, 1)_{n_A} = A$$

It is clear that if $C = B/A$ then $B = C \cdot A$, $\ell_C = \ell_B$ and $n_C = \ell_A$.

It is easy to see that both the product and the quotient of two compositions can be computed in linear time by scanning the operands from left to right.

We also observe that integer compositions can be generated by binary words. More precisely, given a word $x \in \{a, b\}^+$, a *run* of a in x is an occurrence of a maximal factor of x included in $\{a\}^+$ and an analogous definition holds for b. For instance, the word *aaabbabbbaaa* has 3 runs of a and 2 runs of b (*aaa*, *a*, *aaa* and *bb*, *bbb*, respectively). Note that two words $x, y \in \{a, b\}^+$ are equal if they have the same sequence of runs of a, the same sequence of runs of b and start with the same symbol. Moreover, every word $x \in \{a, b\}^+$, where $a \neq b$, $|x|_a \geq 1$ and $|x|_b \geq 1$, defines two compositions C_a and C_b determined, respectively, by the runs of a and the runs of b in x. In particular, $C_a = (c_1, c_2, \ldots, c_h)$ is the composition of sum $|x|_a$ such that the length h is the number of runs of a in x and each c_j is the length of the j-th run. Analogously, C_b is a composition of $|x|_b$ defined in a similar way. We also say that C_a (resp., C_b) is the composition *generated* by x on a (resp., b). Note at last that, if a and b are the first and the last symbol of x respectively, then $|x|_{ab}$ equals the length of C_a.

Now, for an arbitrary expression $\alpha \in \mathrm{RUE}(\Sigma)$, let T be the syntactic tree of word $w \in L(\alpha)$. Then the following property holds:

Observation 2 *For every $a, b \in \Sigma$, if $(c_1, c_2, \ldots, c_h)$ is the composition generated by $\pi_{a,b}(w)$ on a, then in T there are h nodes labelled by $C(a, b)$ and, for each $i = 1, \ldots, h$, there are c_i nodes of label $C(a)$ that are descendants of the i-th node of label $C(a, b)$. Moreover, an analogous statement holds for the composition generated by $\pi_{a,b}(w)$ on b.*

5.1 Labelled compositions

Now, let us see how syntactic trees can be represented by integer compositions. To this end, we introduce the notion of labelled composition. Given an expression $\alpha \in \mathrm{RUE}$ with set of cycles

$\mathcal{C}$, a *labelled composition* is an integer composition equipped with two symbols $A, B \in \mathcal{C}$ such that $B \trianglelefteq A$: we denote it by an expression of the form d_B^A, for some symbol d.

Let T be the syntactic tree of a word in $L(\alpha)$ and consider two cycles $A, B \in \mathcal{C}$ such that $B \trianglelefteq A$ and assume T has h nodes of label A and k nodes of label B. Then, define the labelled composition m_B^A by

$$m_B^A = (a_1, a_2, \ldots, a_h)$$

where, for each $i = 1, \ldots, h$, a_i is the number of B-nodes that are descendants of the i-th A-node in T. Clearly we have $k = n_{m_B^A}$, while $m_B^S = (k)$ and $m_B^B = (1, 1, \ldots, 1)_k$.

Thus, any syntactic tree T defines a family of labelled compositions $\{m_B^A \mid B \trianglelefteq A\}$ satisfying the following proposition, the proof of which follows from the definitions.

Proposition 5 *Given a syntactic tree T, let A, B, C be elements in $\mathcal{C}$ such that $C \trianglelefteq B \trianglelefteq A$. Then the following properties hold:*

1. *$m_B^A \leq m_C^A$ and $m_C^B \preceq m_C^A$;*
2. *The sum of m_B^A equals the length of m_C^B and hence $m_B^A \cdot m_C^B$ is well-defined;*
3. *$m_C^A = m_B^A \cdot m_C^B$ and hence $m_B^A = m_C^A / m_C^B$.*

Thus, properties 1, 2 and 3 above are necessary conditions for a set of labelled compositions to represent a syntactic tree (with respect to a given repeat-until expression α). Actually, they are also sufficient conditions to represent a syntactic tree. However, in order to state such a property it is convenient to restrict our reasoning to the labelled compositions corresponding to pairs of father-son cycles in $T(\alpha)$.

Proposition 6 *Given a hierarchical tree $T(\alpha)$ with set of internal nodes $\mathcal{C}$ and root S, let $N = \{d_B^A \mid A, B \in \mathcal{C}, A = F(B)\}$ be a family of labelled compositions such that:*

1. *For every $A \in \mathcal{C}$ such that $S = F(A)$, the length of d_A^S is 1;*
2. *For every $A, B, C \in \mathcal{C}$ such that $A = F(B)$ and $B = F(C)$, the sum of d_B^A equals the length of d_C^B (and hence $d_B^A \cdot d_C^B$ is well-defined).*

Then there exists a unique syntactic tree T whose family of labelled compositions includes N.

Proof. The syntactic tree T can be built as follows. First, T has a unique node of label S and, for every $X \in \mathcal{C} \backslash \{S\}$, it has k_X many nodes of label X, where k_X is the sum of d_X^Y with $Y = F(X)$. Second, for each $a \in \Sigma$, add an a-node as a son of each X-node such that $X = C(a)$. Then, for every $X, Y \in \mathcal{C}$ where $Y = F(X)$, consider the labelled composition $d_X^Y = (a_1, a_2, \ldots, a_h)$. By condition 2 it is easy to see that there are h nodes of label Y and $n = a_1 + \cdots + a_h$ nodes of label X: thus one can set the first a_1 nodes of label X as sons of the first Y-node, the subsequent a_2 nodes of label X as sons of the second Y-node, and so on till setting the last a_h nodes of label X as sons of the last Y-node. This defines a syntactic tree T and the ordered sequence of the labels of its leaves yields a string $x \in L(\alpha)$. □

Combining Propositions 6 and 5, we can state that a family M of labelled compositions (including at most one composition for each pair $A, B \in \mathcal{C}$ such that $B \trianglelefteq A$) defines a unique syntactic tree T if M includes the set N satisfying the hypothesis of Proposition 6 and all triples $m_B^A, m_B^C, m_C^A \in M$, for $C \trianglelefteq B \trianglelefteq A$, satisfy conditions 1, 2 and 3 of Proposition 5.

6 Recognition algorithm for repeat-until trace languages

Now, let us consider the membership problem for trace languages defined by repeat-until expressions. Formally, given an independence alphabet (Σ, I) and an expression $\alpha \in \mathrm{RUE}(\Sigma)$, the problem consists of deciding, for an input $x \in \Sigma^+$, whether $[x] \cap L(\alpha)$ is empty. The following theorem yields an equivalent condition.

Theorem 7 *Given an independence alphabet (Σ, I) with dependence relation D and an expression $\alpha \in RUE(\Sigma)$, for any $x \in \Sigma^+$ we have $[x] \cap L(\alpha) \neq \emptyset$ if and only if there exists $w \in L(\alpha)$ having syntactic tree T such that:*

a) *For all $a \in \Sigma$, $|x|_a$ is the number of nodes in T labelled by $C(a)$;*

b) *For every $a, b \in \Sigma$ such that aDb and $a < b$, $|\pi_{a,b}(x)|_{ab}$ equals the number of nodes of T labelled by $C(a, b)$;*

c) *For any $a, b \in \Sigma$ such that aDb and $a < b$, if $X = C(a)$, $Y = C(b)$ and $Z = C(a, b)$, then the labelled compositions m_X^Z and m_Y^Z of T coincide with the compositions generated by $\pi_{a,b}(x)$ on a and b, respectively.*

Proof. First recall that a word w belongs to $[x]$ if and only if $|x|_a = |w|_a$ for every $a \in \Sigma$ and $\pi_{a,b}(x) = \pi_{a,b}(w)$ for every pair of distinct symbols $a, b \in \Sigma$ such that aDb. Therefore, if there exists $w \in [x] \cap L(\alpha)$ then w satisfies Observations 1 and 2. Since the projections of x and w on the pairs of (possible coincident) dependent symbols are equal, the same properties hold for x, proving conditions **a)**, **b)** and **c)**.

On the other hand, if there exists $w \in L(\alpha)$ satisfying these conditions then both x and w have the same projections on the pairs of (possible coincident) dependent symbols, proving that $w \in [x]$ and hence $[x] \cap L(\alpha) \neq \emptyset$. □

Now, assuming that $T(\alpha)$ is closed with respect to D, let us define an algorithm for the recognition of $[L(\alpha)]$. The key idea of the computation is to construct, for an input $x \in \Sigma^+$, the syntactic tree T of a word $w \in [x] \cap L(\alpha)$ that satisfies conditions **a)**, **b)**, **c)** of Theorem 7. Such a tree (if any) will be defined by a family of labelled compositions $\{d_B^A \mid A, B \in \mathcal{C}, A = F(B)\}$ that satisfies Proposition 6.

The algorithm consists of three phases. In the first one, by applying conditions **a)** and **b)**, we compute the number k_A of A-nodes in T, for each $A \in \mathcal{C}$. In the second phase we compute the set of all labelled computations d_B^A of T determined by the dependency relation D, i.e. those defined by condition **c)**. In the third phase we close such a set of labelled compositions with respect to the product and the quotient, checking in particular that all products are coherent. Finally, by a suitable choice, we compute explicitely the remaining undefined compositions of the form d_B^A with $A = F(B)$.

6.1 Computing the nodes of the syntactic tree

First of all, the root is the unique node labelled by S. Then, the leaves of T are determined by the occurrences of symbols of Σ in x: for every $a \in \Sigma$ one checks that $|x|_a \geq 1$ and adds $|x|_a$ leaves labelled by a in T. Moreover, by condition **a)** of Theorem 7, the number of nodes labelled by $C(a)$ has to be equal to $|x|_a$. Thus, one has to check that $|x|_b = |x|_a$ for all $b \in \Sigma$

such that $C(b) = C(a)$. Once such a condition is guaranteed, we can assign $|x|_a$ to the required number k_X of X-nodes, where $X = C(a)$. On the contrary, if $|x|_b \neq |x|_a$ for some $b \in \Sigma$ such that $X = C(b)$, then the required syntactic tree T does not exist and hence we reject the input and stop.

A similar reasoning derives from condition **b)**, which allows us to determine the number k_Z of Z-nodes for any cycle $Z \in \mathcal{C}$ such that $Z = C(a, b)$ for some $a, b \in \Sigma$ satisfying $a < b$ and aDb. This value coincides with the number of occurrences of ab in $\pi_{a,b}(x)$. Also in this case one has to verify that k_Z equals $|\pi_{a',b'}(x)|_{a'b'}$ for every pair $a', b' \in \Sigma$ satisfying the same conditions as a, b, i.e. $Z = C(a', b')$, $a' < b'$, $a'Db'$. If that is not true, the procedure rejects the input and stops.

Then, we have to compute the number of B-nodes in T for those $B \in \mathcal{C}\backslash\{S\}$ such that $B \neq C(a)$ for all $a \in \Sigma$ and $B \neq C(a, b)$ for all $a, b \in \Sigma$ satisfying aDb. Observe that every son of such a B in $T(\alpha)$ is a cycle (it is not in Σ) and any pair of symbols $a, b \in \Sigma$ that are discendent of different sons of B are independent. As a consequence, the sons of any B-node in T can be grouped consecutively according to the order defined by $T(\alpha)$. This means we can choose the minimal k_B by setting $k_B = k_A$, where $A = F(B)$. This property can be summarized by the following proposition[2].

Proposition 8 *Given $\alpha \in RUE(\Sigma)$ and a dependence relation D on Σ, let $X \in \mathcal{C}\backslash\{S\}$ be a cycle such that $X \neq C(a)$ for all $a \in \Sigma$ and $X \neq C(a, b)$ for all $a, b \in \Sigma$ satisfying aDb. Let Y be the father of X in $T(\alpha)$, i.e. $Y = F(X)$, and consider a word $w \in L(\alpha)$. Then, there exists $z \in L(\alpha) \cap [w]$ such that every Y-node in the syntactic tree of z has just one son labelled by X (and hence the number of X-nodes equals the number of Y-nodes).*

Finally we have to check that there is at least one son for each father in T, i.e. if $Y = F(X)$ then $k_X \geq k_Y$.

To define formally the computation described above, we use the subroutine `Assign`(z, v) that assigns the value of v to the variable z, checking that the previous possible value of z is not different from v (otherwise a global variable *out* is set to 0).

```
Procedure Assign(z, v)
if z = ⊥ then z := v
         else if z ≠ v then out := 0
```

Then, the computation of the nodes of the syntactic tree is given by the following procedure:

```
begin
     for X ∈ C do k_X := ⊥
     k_S := 1
     out := 1
     for a ∈ Σ do
               begin
                    X := C(a)
                    t := |x|_a
                    if t = 0 then out := 0
                    Assign(k_X, t)
```

[2]The proof is also given in [16, Prop. 1].

```
            end
    for a, b ∈ Σ such that a < b ∧ aDb do
            begin
                Z := C(a, b)
                u := |π_{a,b}(x)|_{ab}
                Assign(k_Z, u)
            end
    for X ∈ C\{S} (in preorder) do
            begin
                Y := F(X)
                if k_X = ⊥ then k_X := k_Y
                           else if  k_X < k_Y then out := 0
            end
end
```

Thus, the variable *out* is set to 0 whenever some necessary condition for computing the nodes of T does not hold. In this case the algorithm stops and rejects the input. On the contrary, if the final value of *out* is 1, then the procedure correctly computes for every $X \in \mathcal{C}$ the number k_X of X-nodes of a possible syntactic tree.

6.2 Initial labelled compositions

In the second phase we compute a set of initial labelled compositions of the required syntatic tree. They are denoted by d_B^A, where $A, B \in \mathcal{C}$ and $B \trianglelefteq A$. Clearly, those of the form d_A^A (for $A \in \mathcal{C}$) easily derive from the values k_A computed in the previous section. Other obvious compositions are those d_B^A's such that $A = F(B)$ and $k_A = k_B$; in this case, $d_B^A = (1, 1, \ldots, 1)_{k_A}$ and this includes all compositions d_X^Y where X and Y satisfy the hypothesis of Proposition 8.

Then, we compute the labelled compositions determined by condition **c)** of Theorem 7. Also in this case a uniqueness condition has to be verified; if a pair $A, B \in \mathcal{C}$ with $B \trianglelefteq A$ is associated with two distinct pairs of dependent symbols, the corresponding labelled compositions have to be equal, otherwise there is no syntactic tree satisfying the required conditions.

The procedure below formally defines the second phase of our algorithm. Again, we use the subroutine `Assign` and, at the end of the computation, if $out = 0$ the algorithm stops and rejects the input.

```
begin
1. labelled compositions derived from nodes
    for A, B ∈ C do d_B^A := ⊥
    for A ∈ C do d_A^A := (1, 1, ..., 1)_{k_A}
    for A ∈ C such that S = F(A) do d_A^S := (k_A)
    for  A, B ∈ C such that A = F(B) do
        if k_A = k_B then d_B^A := (1, 1, ..., 1)_{k_A}
2. labelled compositions derived from dependent pairs
    for a, b ∈ Σ  such that a < b ∧ aDb do
        begin
            X := C(a)
            Y := C(b)
```

$Z := C(a,b)$
compute the composition γ generated by $\pi_{a,b}(x)$ on a
`Assign`(d_X^Z, γ)
compute the composition δ generated by $\pi_{a,b}(x)$ on b
`Assign`(d_Y^Z, δ)
`end`
`end`

6.3 Closure operations

Once the previous phase has been completed without setting *out* to 0, we have to close the set M of labelled compositions determined so far with respect to the product and the quotient. Observe that, by the procedure of Section 6.1, for any $A, B, C \in \mathcal{C}$ such that $C \lhd B \lhd A$, if $d_B^A \neq \perp \neq d_C^B$ then the product $d_B^A \cdot d_C^B$ is well-defined because $n_{d_B^A} = k_B$ equals the length of d_C^B. Then the product $d_B^A \cdot d_C^B$ can be computed and assigned to d_C^A, checking that a unique composition is assigned to the same pair A, C.

The computation is defined by the following scheme.

`repeat`
`for` $A, B, C \in \mathcal{C}$ such that $C \lhd B \lhd A$ `do`
`if` $d_B^A \neq \perp \neq d_C^B$ `then` $\left\{ \begin{array}{l} \gamma := d_B^A \cdot d_C^B \\ \texttt{Assign}(d_C^A, \gamma) \end{array} \right.$ (1)
`for` $A, B, C \in \mathcal{C}$ such that $C \lhd B$ and $A = F(B)$ `do`
`if` $d_C^A \neq \perp \neq d_C^B$ `then if` $d_C^B \preceq d_C^A$ `then` $\left\{ \begin{array}{l} \delta := d_C^A / d_C^B \\ \text{Assign}(d_B^A, \delta) \end{array} \right.$ (2)
`else` $out := 0$
`until` $out = 0$ or no new assignment is executed in commands (1) and (2)

Clearly, if $out = 0$ the input is rejected, otherwise it is accepted. Note that there could still exist pairs of father-son cycles $A, B \in \mathcal{C}$ such that $d_B^A = \perp$. However, in this case any composition of length k_A and sum k_B can be assigned to d_B^A since, by the closure hypothesis, there is no labelled composition in M connecting an ancestor of A to a descendent of B.

`for` $A, B \in \mathcal{C}$ such that $A = F(B)$ `do`
`if` $d_B^A = \perp$ `then` $\left\{ \begin{array}{l} \text{choose a composition } \gamma \text{ of length } k_A \text{ and sum } k_B \\ d_B^A := \gamma \end{array} \right.$

Thus, in case of acceptance, d_B^A is well defined for every $A, B \in \mathcal{C}$ such that $A = F(B)$ and the syntactic tree of a word $w \in [x]$ is obtained by applying Proposition 6.

Since the product and the quotient of integer compositions can be computed in linear time, the whole algorithm works in $O(n)$ time, where $n = |x|$.

Theorem 9 *For every independence alphabet (Σ, I) and every expression $\alpha \in RUE(\Sigma)$, if $T(\alpha)$ is closed with respect to the dependence relation then the trace language $[L(\alpha)]$ can be recognized in $O(n)$ time.*

If the hierarchical tree is not closed with respect to the dependence relation, the algorithm above may fail to build a syntactic tree (even if there exists one) because some father-son connection could remain undefined. This may happen when, for an adjacent pair (Y, X) and a cycle B such that $X \lhd B \lhd Y$, B is not connected to X through D and several choices for d_B^A, where $A = F(B)$, are coherent with the compositions occurring along the path from Y to X. A simple choice of one of these is not always correct, because (without the closure assumption) the resulting d_B^A might not be coherent with the initial labelled compositions occurring in other paths including B. The following example describes in detail a situation of this kind.

Consider the hierarchical tree $T(\alpha)$ defined by the picture of Example 3 and let the dependence relation be given by the pairs $\{a, d\}$, $\{b, c\}$, $\{b, f\}$, $\{a, g\}$. Clearly $T(\alpha)$ is not closed with respect to such a relation. In particular the paths from A to D and from A to G are not closed.

Now, assume the projections of the input x over $\{a, d\}$ and $\{b, c\}$ are given by $\pi_{a,d} = ddddddadddda$ and $\pi_{b,c} = ccbccbcbcb$, respectively. Here, the number of nodes of label A, B, C and D are, respectively, $k_A = 2$, $k_B = 4$, $k_C = 5$ and $k_D = 9$, while the initial compositions defined by $\{a, d\}$ and $\{b, c\}$ are $d_D^A = (4, 5)$ and $d_C^B = (2, 2, 1, 1)$. There are two possible choices for d_C^A coherent with d_D^A and d_C^B, i.e. satisfying $d_C^A \leq d_D^A$ and $d_C^B \preceq d_C^A$; they are $d_C^A = (4, 2)$ and $d_C^A = (2, 4)$, which produce, by the quotient operation, the labelled compositions $d_B^A = (2, 2)$ and $d_B^A = (1, 3)$, respectively.

However, an analogous reasoning based on dependence pairs $\{b, f\}$ and $\{a, g\}$ may yield a partially different set of possible values for d_B^A. In fact, assume $\pi_{b,f} = bffbfbffbf$ and $\pi_{a,g} = gggggaggga$. In this case we have $k_F = 6$, $k_G = 8$, $d_F^B = (2, 1, 2, 1)$ and $d_G^A = (5, 3)$. The possible values for d_F^A are $(3, 3)$ and $(5, 1)$, which implies the compositions $d_B^A = (2, 2)$ and $d_B^A = (3, 1)$, respectively.

Thus, the only value of d_B^A that is coherent with both paths $A - D$ and $A - G$ is $d_B^A = (2, 2)$. Therefore, in the general case, for computing a labelled composition d_B^A one should determine the set of admissible values for each including path and compute the intersection of all these sets. However, such a computation does not seem to be feasible as the number of compositions of given sum is exponential.

6.4 Uniform problem for repeat-until expressions

The algorithm described above can also be applied to solve the uniform version of our problem. In this case, the input is given by an independence alphabet (Σ, I), a string $x \in \Sigma^*$ and an expression $\alpha \in \mathrm{RUE}(\Sigma)$, which is assumed to be closed with respect to the dependence relation $D = I^c$. Here, a natural size of the input is the pair (m, n), where m is the cardinality of Σ and $n = |x|$. Note that the size of the graph (Σ, I) is $O(m^2)$, while $|\alpha| = O(m)$ and also the size of $T(\alpha)$ is $O(m)$.

It is easy to check that the procedure of Section 6.1 for computing the nodes of a possible syntactic tree requires $O(m^2 + mn)$ steps. Similarly, the second phase of the algorithm described in Section 6.2 can be done in $O(m^2 n)$ time. In the last phase, it is clear that each iteration of the main loop of the procedure presented in Section 6.3 can be executed in $O(m^3 n)$ time. Since there are at most $O(\log m)$ iterations the overal time complexity of the algorithm is $O(nm^3 \log m)$, proving that under the closure assumption the uniform membership problem for repeat-until trace languages is solvable in polynomial time.

Note at last that also the closure of an expression $\alpha \in \mathrm{RUE}(\Sigma)$ with respect to a dependence relation D can be tested in $O(m^3)$ time. In fact this computation reduces to compute relation

Con and then verify for every triple of nested cycles $C \lhd B \lhd A$ with $(A, C) \in Adj$, whether $(B, C) \in Con$.

6.5 On the closure assumption

To conclude this section we spend some words to discuss the closure assumption for nested repeat-until programs. We recall that the innermost loop containing instruction a is "adjacent" to an outer loop containing instructions a and b, if the two instructions are dependent. Here the adjacency relation can be seen as a directed edge from the outer loop to the inner one. Also, an innermost loop X and an outer loop Y are "connected" if there is a path of adjacences from Y to X. The closure hypothesis says that, in any chain of nested loops, such that the outermost and the innermost one are adjacent, each intermediate loop is connected to the innermost one.

For instance, it is easy to see that the standard procedure for matrix multiplication has three nested cycles, where the header of each loop increments a control variable and is dependent on the innermost instruction.

Even if the closure assumption is not satisfied by all repeat-until expressions, however we think it covers a significant part of these languages, those for which the reconstruction of the syntactic tree (from the projections of the input string on the dependence pairs) can be done univocally by using the operations of product and quotient between compositions.

Moreover, the existence of linear time recognition algorithms under this hypothesis supports the conjecture that also for more general expressions there exist efficient procedures for the membership problem, with time complexity independent of the clique size of the independence relation. A first attempt in this direction is proposed in [6] where some procedures, working in quadratic time, are described for specific examples of repeat-until expressions without closure assumption. That approach however is not based on a general property relating the repeat-until expression to the dependence relation and involves operations over integer compositions, other than product and quotient, which do not seem to be computable in linear time.

7 Conclusion

An important problem in program optimization and in other computer applications is the schedule checking problem. It consists of checking whether a given sequence of operations is a permutation of any sequence defined by a finite-state machine, obeying a given dependence relation. We have presented two linear-time algorithms that solve the problem under certain assumptions, which we believe to be not restrictive for certain realistic cases. This may open the way to the experimentation of our algorithms, in contrast to previous procedures for the general problem, which have too high time complexity to be practical.

Analysing general iterative computations, as we did for nested repeat-until cycles, is rather complicated. In our case we have overcome the difficulty by introducing the labelled integer compositions in this context, and we have shown that they are quite expressive and convenient mathematical structures. Their use has allowed us to clarify and improve on previous efforts to solve the schedule checking problem, determining precisely the time complexity of the algorithm in several significant cases. In our opinion, it should be possible to apply similar methods based on integer compositions to more general cases, such as programs of loops of type *while* ... *do* ... and others.

References

[1] A. Avellone, M. Goldwurm. Analysis of algorithms for the recognition of rational and context-free trace languages. *RAIRO Theoretical Informatics and Applications* 32: 141-152, 1998.

[2] J. Berstel, J.-E. Pin. Local languages and the Berry-Sethi algorithm. *Theoret. Comput. Sci.* 155:439-446, 1996.

[3] A. Bertoni, M. Goldwurm, G. Mauri, N. Sabadini. Counting techniques for inclusion, equivalence and membership problems, in *The book of traces*, V. Diekert and G. Rozenberg Editors, World Scientific, 131-163, 1995.

[4] A. Bertoni, G. Mauri, N. Sabadini. Unambiguous regular trace languages. Proc. Coll. on Algebra, Combinatorics and Logic in Computer Science, Colloquia Mathematica Soc. J. Bolyai, 42: 113-123, North-Holland, 1985.

[5] A. Bertoni, G. Mauri, N. Sabadini. Membership problems for regular and context-free trace languages. *Information and Computation* 82 (2): 135-150, 1989.

[6] L. Breveglieri, S. Crespi Reghizzi, M. Goldwurm. Integer compositions and syntactic trees of repeat-until programs. Tech. Rep. n. 323-08, Dip. Scienze dell'Informazione, Università degli Studi di Milano. Presented at the Workshop DNTTT08, Developments and New Tracks in Trace Theory, Cremona, 9-11 October 2008.

[7] L. Breveglieri, S. Crespi Reghizzi, A. Savelli. Efficient word recognition of certain locally defined trace languages. Proc. 5th Int. Conf. on Words, Montreal (Canada), September 2005.

[8] V. Diekert and G. Rozenberg (editors). *The book of traces*, World Scientific, 1995.

[9] J. A. Fisher, P. Faraboschi, C. Young. *Embedded computing: a VLIW approach to architecture, compilers and tools*, Morgan-Kaufmann Publishers, 2005.

[10] P. Flajolet. Mathematical methods in the analysis of algorithms and data structures, in *Trends in Theoretical Computer Science*, E. Börger Ed., Comp. Science Press, 225–304, 1988.

[11] W. LaLonde. Regular right part grammars and their parsers. *Communications of the ACM* 20 (10): 731–741, 1977.

[12] A. Mazurkiewicz, Concurrent program schemes and their interpretations, DAIMI Rep. PB 78, Aarhus University, Aarhus, 1977.

[13] D. A. Patterson, J. L. Hennessy. *Computer Organization and Design*, Morgan-Kaufmann Publishers, San Francisco, 1998.

[14] W. Rytter. Some properties of trace languages *Fund. Inform.* 7:117-127, 1984.

[15] J. Sakarovitch. On regular trace languages. *Theoret. Comput. Sci.* 52: 59-75, 1987.

[16] A. Savelli, Two contributions to automata theory on parallelization and data compression, Doctoral Thesis, Politecnico di Milano, Université de Marne-la-Vallée, June 2007.