LALBLC
a program testing the equivalence of dpda's

P. Henry and G. Sénizergues

LaBRI and Université de Bordeaux, Talence, France {Patrick.Henry,ges}@labri.fr

Abstract. We describe the program LALBLC which tests whether two deterministic pushdown automata recognize the same language.

keywords: Deterministic pushdown automata ; deterministic context-free grammars ; equivalence problem.

1 Introduction

The so-called "equivalence problem for deterministic pushdown automata" is the following decision problem:

$$\text{INSTANCE} : \text{two dpda } A, B; \quad \text{QUESTION} : \mathrm{L}(A) = \mathrm{L}(B)?$$

i.e. do the given automata recognize the *same* language ? This problem was shown to be decidable in ([Sén97],[Sén01a, sections 1-9])[1]. Beside crude decidability, the intrinsic complexity of this problem is far from being understood. A progress in this direction has been achieved in [Sti02] by showing that the general problem is *primitive recursive* while subclasses with complexity in P (resp. co-NP) have been discovered in [BCFR06,BG11,BGJ13,CCD13](resp. [Sén03]). Any further progress in this direction is likely to have some impact on other areas of computer science, as is shown by the numerous applications that were found even before proving decidability of the problem (see [Sén01b] for a survey and [CCD13] for a recent new connection).

The contribution presented here consists in showing that it is *practically feasable* to solve the equivalence problem for general dpda on non-trivial examples (see section 5). We have implemented, and to some extent refined, the main ideas of [Sén01a]. On every pair (A, B) the program returns, either a proof of $\mathrm{L}(A) = \mathrm{L}(B)$ (see section 4 for a precise notion of proof) or a terminal word witnessing the fact that $\mathrm{L}(A) \neq \mathrm{L}(B)$.

The sources of our (Python) program can be uploaded from `http://dept-info.labri.u-bordeaux.fr/~ges`; the program can also be run on predefined (or user-defined) examples on the server `lalblc.labri.fr`.

2 Automata, grammars

We introduce here the notions of automata and grammars that are manipulated by the program.

[1] a similar method is exposed within the framework of term root-rewriting in [Jan12]

2.1 Deterministic matricial fa

A *finite automaton* is, as usual, a tuple, $\mathcal{A} =< X, Q, Q_-, Q_+, \delta >$ where X is the input alphabet, Q is the set of states, $Q_- \subset Q$ is the set of initial states, $Q_+ \subset Q$ is the set of terminal states, $\delta \subset Q \times X \times Q$ is the set of transitions, and all of these five sets are finite.
The main object that our program handles is a *matrix* of languages which is defined by some finite automaton with one set of initial states for each line and one set of terminal states for each column.
Let us recall that a language $L \subset X^*$ is a *prefix language* iff, $\forall u, v \in L, u \preceq v \Rightarrow u = v$. The line-vectors of the matrices we are interested in are *prefix vectors* in the following sense:

Definition 1 *A vector $(L_1, \ldots, L_i, \ldots, L_n) \in \mathcal{P}(X^*)^n$ is said to be* prefix *iff it fulfills: $\forall i, j \in [1, n], i \neq j \Rightarrow L_i \cap L_j = \emptyset$ and $\bigcup_{i=1}^{n} L_i$ is a prefix language.*

We thus consider the following variant of the notion of d.f.a. which recognizes a *prefix* matrix of languages i.e. where each row-vector is prefix.

Definition 2 (complete deterministic matricial f.a.) *A finite* complete deterministic matricial finite automaton *is a tuple, $\mathcal{A} =< X, Q, Q_{1,-}, \ldots, Q_{n,-}, Q_{1,+}, \ldots, Q_{m,+}, \delta >$ where X is the input alphabet, Q is the set of states, $Q_{i,-} \subseteq Q$ is the set of initial states of the i-th line, $Q_{j,+} \subseteq Q$ is the set of final states of the j-th column,*

$$\forall i \in [1, n], \mathrm{Card}(Q_{i,-}) = 1$$
$$\forall j, k \in [1, m], j \neq k \Rightarrow Q_{j,+} \cap Q_{k,+} = \emptyset$$

$\delta : (Q \times X) \to Q$ is a total map, which is called the transition map. All the items of this tuple are assumed to be finite sets.

Such an automaton defines a prefix *matrix of languages* $\mathrm{L}(\mathcal{A}) := (L_{i,j})_{(i,j)\in[1,n]\times[1,m]}$ where $L_{i,j} := \{u \in X^* \mid q_{i-} \in Q_{i,-}, \delta^*(q_{i,-}, u) \in Q_{j,+}\}$.
The usual theory of recognizable languages, complete deterministic automata and residuals can be adapted to matrices of languages, cdmfa's and residuals of matrices.

Implementation Our module `fautomata` deals with f.automata and their analogues. The class `dr-matrix` implements the notion of cdmfa. The program stores every rational prefix matrix under the form of a *canonical* dcmfa i.e. a minimal dcmfa, in which the states are integers that are completely determined by some depth-first traversal of the minimal automaton. The equality of two rational (prefix) matrices is implemented as an isomorphism-test for the corresponding canonical dcmfa.

2.2 Pushdown automata and context-free grammars

The notions of *pushdown automaton* and *context-free grammars* are well-known. A pda is said *deterministic* if, informally, on every triple (state, stack-contents, tape-contents), *at most one* transition is applicable. It is called *strict* if it recognizes by empty stack and a finite set of final sates and *normal* if every ϵ-transition is popping.

Definition 1. *([Har78, Definition 11.4.1 p.347]) Let $G =< X, V, P >$ be a context-free grammar. G is said* strict-deterministic *iff there exists an equivalence relation $\smile$ over V fulfilling the following condition:*
1- X is a class $(\mathrm{mod}\ \smile)$
2-for every $v, v' \in V, \alpha, \beta, \beta' \in (X \cup V)^*$, if $v \longrightarrow_P \alpha \cdot \beta$ and $v' \longrightarrow_P \alpha \cdot \beta'$ and $v \smile v'$, then either:
2.1- both $\beta, \beta' \neq \epsilon$ and $\beta[1] \smile \beta'[1]$
2.2- or $\beta = \beta' = \epsilon$ and $v = v'$.

(In the above definition, for every word γ, $\gamma[1]$ denotes the first letter of the word γ). Any equivalence $\smile$ satisfying the above condition is said to be a *strict equivalence* for the grammar G. It is known that, given a strict dpda $\mathcal{M}$, one can construct, in polynomial time, an associated grammar $G_{\mathcal{M}} =< X, V_{\mathcal{M}}, P_{\mathcal{M}} >$ which is strict-deterministic and generates the language recognized by $\mathcal{M}$.

Implementation Our module `grammars` deals with dpda and dcf grammars. The translation of a dpda into a dcf grammar is realized by the `autotogram(A)` function; some routine functions around these notions are implemented (test for determinism of a cf grammar, elimination of non-productive non-terminals and reduction in Greibach normal-form, for grammars translating a normal strict dpda),see Figure 1.

```
Non-terminal symbols :
[<q2-A-q4> <q2-A-qb> ][<q4-O-q3> ][<q1-O-q3> ]
[<q2-O-q3> ][<q3p-O-q3> ][<q4-A-q4> ][<q3-O-q3> ]
[<q3-A-q3> ][<q1-A-q3> <q1-A-q5> ][<q3b-O-q3> ]

Terminal symbols : # a b x
Rewriting rules:
 <q1-A-q3>::=a
 <q1-A-q3>::=x<q1-A-q3><q3-A-q3>
 <q1-A-q5>::=b
 <q1-A-q5>::=x<q1-A-q5>
 <q1-O-q3>::=#<q1-A-q3><q3-O-q3>
 <q1-O-q3>::=#<q1-A-q5>
 <q2-A-q4>::=a<q4-A-q4><q4-A-q4>
 <q2-A-q4>::=x<q2-A-q4><q4-A-q4>
 <q2-A-qb>::=b
 <q2-A-qb>::=x<q2-A-qb>
 <q2-O-q3>::=#<q2-A-q4><q4-O-q3>
 <q2-O-q3>::=#<q2-A-qb>
 <q3-A-q3>::=a
 <q3-O-q3>::=a<q3p-O-q3>
 <q3b-O-q3>::=a
 <q3p-O-q3>::=a<q3b-O-q3>
 <q4-A-q4>::=a
 <q4-O-q3>::=a

Axiom: <q1-O-q3>
```

Fig. 1. A dcf grammar G2 (obtained from some dpda)

3 Algebraic framework

We recall here the algebraic framework which is the base of our program (see [Sén01a, sections 2,3] for more details).

3.1 Semi-rings and right-actions

Semi-ring $\mathbb{B}\langle\langle\ W\ \rangle\rangle$ Let $(\mathsf{B}, +, \cdot, 0, 1)$ where $\mathsf{B} = \{0, 1\}$ denote the semi-ring of "booleans". Let W be some alphabet. By $(\mathbb{B}\langle\langle\ W\ \rangle\rangle, +, \cdot, \emptyset, \epsilon)$, we denote the semi-ring of *boolean series* over W (which is, up to isomorphism, nothing else than the semi-ring of subsets of W^*, $(\mathcal{P}(W^*), \cup, \cdot, \emptyset, \{\epsilon\})$.

Right-actions over $\mathbb{B}\langle\langle\ W\ \rangle\rangle$ We recall the following classical right-action $\bullet$ of the monoid W^* over the semi-ring $\mathbb{B}\langle\langle\ W\ \rangle\rangle$: for all $S, S' \in \mathbb{B}\langle\langle\ W\ \rangle\rangle, u \in W^*$

$$S \bullet u = S' \Leftrightarrow \forall w \in W^*, (S'_w = S_{u \cdot w}),$$

(i.e. $S \bullet u$ is the *residual* of S by u). Let $(V, \smile)$ be the structured alphabet associated with a strict-deterministic grammar (see paragraph §2.2). We define the right-action $\odot$ over non-terminal words by:

$$(v \cdot \beta) \odot x = (\sum_{(v,h) \in P} h \bullet x) \cdot \beta, \ \ \epsilon \odot x = \emptyset.$$

The action is then extended to arbitray boolean series (on the left) and to arbitray terminal words (on the right) by:

$$(\sum_{w \in W^*} S_w \cdot w) \odot x := \sum_{w \in W^*} S_w (w \odot x), \ \ S \odot \varepsilon := S, \ \ S \odot wx := (S \odot w) \odot x$$

3.2 Deterministic matrices

We recall here the notion of *deterministic* series and, more generally, deterministic matrices [2] [3]. Let us consider a pair $(W, \smile)$ where W is an alphabet and $\smile$ is an equivalence relation over W. We call $(W, \smile)$ a *structured* alphabet.
Let us denote by $\mathbb{B}_{n,m}\langle\langle\ W\ \rangle\rangle$ the set of (n, m)-matrices with entries in the semi-ring $\mathbb{B}\langle\langle\ W\ \rangle\rangle$ (the index (m, n) will continue to mean "of dimension (m, n)" for all subsequent subsets of matrices).

Definition 3 *Let* $m \in \mathbb{N}, S \in \mathbb{B}_{1,m}\langle\langle\ W\ \rangle\rangle : S = (S_1, \cdots, S_m)$. *$S$ is said* left-deterministic *iff either* $\forall i \in [1, m], S_i = \emptyset$ *or*
$\exists i_0 \in [1, m], S_{i_0} = \epsilon$ *and* $\forall i \neq i_0, S_i = \emptyset$ *or*
$\forall w, w' \in W^*, \forall i, j \in [1, m], (S_i)_w = (S_j)_{w'} = 1 \Rightarrow [\exists A, A' \in W, w_1, w'_1 \in V^*, A \smile A', w = A \cdot w_1$ *and* $w' = A' \cdot w'_1]$.

Both right-actions $\bullet, \odot$ on $\mathbb{B}\langle\langle\ W\ \rangle\rangle$ are extended componentwise to $\mathbb{B}_{n,m}\langle\langle\ W\ \rangle\rangle$.

Definition 4 *A row-vector* $S \in \mathbb{B}_{1,m}\langle\langle\ W\ \rangle\rangle$. *$S$ is said* deterministic *iff, for every* $u \in W^*$, *$S \bullet u$ is left-deterministic.*
A matrix $S \in \mathbb{B}_{n,m}\langle\langle\ W\ \rangle\rangle$. *$S$ is said* deterministic *iff, for every* $i \in [1, n]$, $S_{i,\cdot}$ *is a deterministic row-vector.*

The classical definition of *rationality* of series in $\mathbb{B}\langle\langle\ W\ \rangle\rangle$ is extended componentwise to matrices. Given $A \in \mathbb{B}_{1,m}\langle\langle\ W\ \rangle\rangle$ and $1 \leq j_0 \leq m$, we define the vector $\nabla^*_{j_0}(A) := A$ by:
if $A = (a_1, \ldots, a_j, \ldots, a_m)$ then $A' := (a'_1, \ldots, a'_j, \ldots, a'_m)$ where

$$a'_j := a^*_{j_0} \cdot a_j \text{ if } j \neq j_0 \ , \ \ a'_j := \emptyset \text{ if } j = j_0.$$

[2] these series play, for dcf grammars the role that *configurations* play for a dpda
[3] it extends the notion of (finite) *set of associates* defined in [HHY79, definition 3.2 p. 188]

Note that every deterministic matrix is prefix; it follows that every deterministic rational matrix is recognized by some cdmfa. The main closure poperties of deterministic rational matrices are summarized below.

Proposition 1. *Let* $S \in \mathsf{DR}\mathbb{B}_{n,m}\langle\langle\ W\ \rangle\rangle, T \in \mathsf{DR}\mathbb{B}_{m,s}\langle\langle\ W\ \rangle\rangle$,$w \in W^*$, $u \in X^*$, *Then*
$S \cdot T \in \mathsf{DR}\mathbb{B}_{n,s}\langle\langle\ W\ \rangle\rangle, S \bullet w \in \mathsf{DR}\mathbb{B}\langle\langle\ W\ \rangle\rangle, S \odot u \in \mathsf{DR}\mathbb{B}\langle\langle\ W\ \rangle\rangle$
If $n = 1, 1 \leq j_0 \leq m$,*then* $\nabla^*_{j_0}(S) \in \mathsf{DR}\mathbb{B}_{1,m}\langle\langle\ W\ \rangle\rangle$.

These closure properties are effective.

Terminal matrices versus non-terminal matrices Let us denote by $\mathrm{L} : \mathsf{D}\mathbb{B}\langle\langle\ V\ \rangle\rangle \to \mathsf{D}\mathbb{B}\langle\langle\ X\ \rangle\rangle$ the map sending every deterministic series S on the language $\mathrm{L}(S) := \{u \in X \mid S \odot u = \varepsilon\}$ (i.e. the set of terminal words generated from all the non-terminal words of S via the derivation w.r.t. the rules of G). For every integers $n, m \geq 1$, L is extended componentwise as a map $\mathsf{D}\mathbb{B}_{n,m}\langle\langle\ V\ \rangle\rangle \to \mathsf{D}\mathbb{B}_{n,m}\langle\langle\ X\ \rangle\rangle$.

Lemma 1. *For every* $S \in \mathsf{D}\mathbb{B}_{n,m}\langle\langle\ V\ \rangle\rangle, T \in \mathsf{D}\mathbb{B}_{m,s}\langle\langle\ V\ \rangle\rangle, u \in X^*$,
$\mathrm{L}(\varepsilon) = \varepsilon,\ \ \mathrm{L}(S \cdot T) = \mathrm{L}(S) \cdot \mathrm{L}(T),\ \ \mathrm{L}(S \odot u) = \mathrm{L}(S) \bullet u$.

Implementation The module `fautomata` implements the matricial product (`prod`), the right-actions `bullet`, `odot` and the operation `nablastar`.

3.3 Linear combinations

Let us call *linear combination* of the series $S_1, \ldots, S_j, \ldots, S_m$ any series of the form $\sum_{1 \leq j \leq m} \alpha_j \cdot S_j$ where $\boldsymbol{\alpha} \in \mathsf{DR}\mathbb{B}_{1,m}\langle\langle\ V\ \rangle\rangle$. Let $S_1, \ldots, S_j, \ldots, S_m \in \mathsf{DR}\mathbb{B}\langle\langle\ V\ \rangle\rangle$. We call *dependency* of order 0 between the S_j's, an equality of the form:

$$S_{j_0} = \sum_{1 \leq j \leq m} \gamma'_j \cdot S_j, \tag{1}$$

where $j_0 \in [1, m]$, $\boldsymbol{\gamma}' \in \mathsf{DR}\mathbb{B}_{1,m}\langle\langle\ V\ \rangle\rangle$ and $\gamma'_{j_0} = \emptyset$. [4] Analogously, we call *dependency* of order 1 between the S_j's, an equality of the form (1), but where the symbol "=" is replaced by the symbol "$\equiv$". It is clear that the homomorphism L maps every dependency of order 1 between the S_j's onto a dependency of order 0 between the $L(S_j)$.

Canonical coordinates Let $S, T_1, T_2, \ldots, T_n \mathsf{D}\mathbb{B}\langle\langle\ V\ \rangle\rangle$. We assume that $i \neq j \Rightarrow T_i \neq T_j$. For every $i \in [1, n]$ we define
$\alpha_i := \{u \in V^* \mid S \bullet u = T_i \text{ and } \forall u' \prec u, \forall j \in [1, n], S \bullet u' \neq T_j\}$ and
$\alpha_{n+1} := \{u \in S \mid \forall u' \preceq u, \forall j \in [1, n], S \bullet u' \neq T_j\}$.

Lemma 2. *The vector* $\boldsymbol{\alpha}$ *of canonical coordinates fullfils:*
1- $\boldsymbol{\alpha} \in \mathsf{D}\mathbb{B}_{1,n+1}\langle\langle\ V\ \rangle\rangle,\ S = \sum_{i=1}^{n} \alpha_i \cdot T_i + \alpha_{n+1}$
2- S *is a linear combination of the* T_i*, with a vector of coefficients in* $\mathsf{D}\mathbb{B}_{1,n}\langle\langle\ V\ \rangle\rangle$*, iff* $\alpha_{n+1} = \emptyset$.

[4] This terminology originates in [Mei89].

Unifiers The following notion was implicit in [Sén01a, section 5] and explicited in [Sén05, section 11]. It turns out to be central in our implementation. Let $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathsf{D}\mathbb{B}_{1,q}\langle\langle\ X\ \rangle\rangle$.
A *unifier* of $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ is any matrix $U \in \mathsf{D}\mathbb{B}_{q,q}\langle\langle\ X\ \rangle\rangle$ such that: $\boldsymbol{\alpha} \cdot U = \boldsymbol{\beta} \cdot U$.
U is a *Most General* Unifier iff, every unifier of $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ has the form $U \cdot T$ for some $T \in \mathsf{D}\mathbb{B}_{q,q}$. This notion is lifted to $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathsf{DR}\mathbb{B}_{1,q}\langle\langle\ V\ \rangle\rangle$ via the map L.

Theorem 5. *1- Every pair $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathsf{D}\mathbb{B}_{1,q}\langle\langle\ V\ \rangle\rangle$ has a MGU (up to $\equiv$)*
2- This MGU is unique, *up to $\equiv$ and up to some right-product by a permutation matrix.*
3- For pairs $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathsf{DR}\mathbb{B}_{1,q}\langle\langle\ V\ \rangle\rangle$ the MGU has some representative which belongs to $\mathsf{DR}\mathbb{B}_{q,q}\langle\langle\ V\ \rangle\rangle$ and is computable from $\boldsymbol{\alpha}, \boldsymbol{\beta}$.

In other words, the MGU of two algebraic row-vectors defined by det. rational vectors over a s.d. grammar G is itself algebraic and definable by a det. rational-matrix over the grammar G. The MGU of $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathsf{DR}\mathbb{B}_{1,q}\langle\langle\ V\ \rangle\rangle$ can be computed along the following algorithm scheme:

$M \leftarrow \mathrm{Id}_q;\ cost \leftarrow 0$
while (not $\boldsymbol{\alpha} \cdot M \equiv \boldsymbol{\beta} \cdot M$) **do**
  find $j \in [1, q], w \in X^*$, prefix-minimal, such that:
  $((\boldsymbol{\alpha} \cdot M) \odot w = \varepsilon_j^q)$ iff $((\boldsymbol{\beta} \cdot M) \odot w \neq \varepsilon_j^q)$
  $\boldsymbol{\gamma} \leftarrow (\boldsymbol{\alpha} \cdot M) \odot w$ (if it is equal to ε_j^q) or $\boldsymbol{\gamma} \leftarrow (\boldsymbol{\beta} \cdot M) \odot w$ (if it is equal to ε_j^q)
  $\boldsymbol{\gamma} \leftarrow \nabla_j^*(\boldsymbol{\gamma})$)
  $D \leftarrow \mathrm{Id}_q$; $D_{j,*} \leftarrow \boldsymbol{\gamma}$ {D is the dependency matrix associated to $\boldsymbol{\gamma}$ and j}
  $M \leftarrow M \cdot D;\ cost \leftarrow cost + |w|$
end while
return [M,$cost$]

(See on figure 2 an example of mgu computation, where $q = 4$).
The integer $cost$ is useful for a proper use of M leading to an equivalence proof (i.e. for ensuring property (3) of §4.5).

```
v1: list of states [0, 1, 2, 3]
sets of initial states [[0]]
sets of final states [[1], [3], [], []]
list of (non-sink) transitions:
( 0 <q1-A-q3> )--> 1
( 0 <q1-A-q5> )--> 3

v2: list of states [0, 1, 2, 3]
sets of initial states [[0]]
sets of final states [[], [], [2], [3]]
list of (non-sink) transitions:
( 0 <q2-A-q4> )--> 2
( 0 <q2-A-qb> )--> 3
```

```
mgu list of states [0, 1, 2, 3, 4]
sets of initial states [[0], [4], [3], [4]]
sets of final states [[], [], [3], [4]]
list of (non-sink) transitions:
( 0 <q4-A-q4> )--> 2
( 2 <q4-A-q4> )--> 3

cost_mgu 2
```

Fig. 2. A mgu w.r.t. grammar G2

Implementation The module `fautomata` implements the function `coords` that computes the canonical coordinates of a d.r. series over a finite family of d.r. series.
The module `equations` defines a functional `mgu(f-equiv,f-op,vec1,vec2)`:

it computes the MGU of two row-vectors by the above algorithm where `f-equiv` is used for the testing the equivalence (or returning a witness) of two row-vectors and `f-op` is the right-action used for computing the dependency $\boldsymbol{\gamma}$. The MGU' s of order 0 or approximate [5] MGU' s at order 1 are obtained by application of this functional.

4 Logics

4.1 The deduction relation

We denote by $\mathcal{A}$ the set $\mathsf{DR}\mathbb{B}\langle\langle\ V\ \rangle\rangle \times \mathsf{DR}\mathbb{B}\langle\langle\ V\ \rangle\rangle$. An element $(S,T) \in \mathcal{A}$ is called an equation while a triple (p,S,T) where $p \in \mathbb{N}$ is called a *weighted* equation. The *divergence* of (S,T), denoted by $\mathrm{Div}(S,T)$, is defined by:

$$\mathrm{Div}(S,T) := \inf\{|u| \ \mid u \in X^*, (S \odot u = \varepsilon) \Leftrightarrow (S \odot u \neq \varepsilon)\}$$

The map Div is extended to sets of equations by: $\mathrm{Div}(P) := \inf\{\mathrm{Div}(p) \mid p \in P\}$. Let $\mathcal{B}, \mathcal{C}$ be the sets of meta-rules described in figure 3. We define the binary rela-

$$\begin{array}{llll}
(R0) & \emptyset & |\!\vdash\!- & (T,T) \\
(R1) & \{(T,T')\} & |\!\vdash\!- & (T',T) \\
(R2) & \{(T,T'),(T',T'')\} & |\!\vdash\!- & (T,T'') \\
(R3) & \{(S_1,T_1),(S_2,T_2)\} & |\!\vdash\!- & (S_1+S_2,T_1+T_2) \\
(R4) & \{(T,T')\} & |\!\vdash\!- & (T\cdot U,T'\cdot U) \\
(R5) & \{(T,T')\} & |\!\vdash\!- & (U\cdot T,U\cdot T') \\
(R6) & \{(U_1\cdot T+U_2,T)\} & |\!\vdash\!- & (U_1^*\cdot U_2,T)
\end{array}
\qquad
\begin{array}{llll}
(W0) & \emptyset & |\!\vdash\!- & (0,T,T) \\
(W0') & \{(p,S,T)\} & |\!\vdash\!- & (p+1,S,T) \\
(W1) & \{(p,T,T')\} & |\!\vdash\!- & (p,T',T) \\
(W2) & \{(p,T,T'),(p,T',T'')\} & |\!\vdash\!- & (p,T,T'') \\
(W3) & \{(p,S_1,T_1),(p,S_2,T_2)\} & |\!\vdash\!- & (p,S_1+S_2,T_1+T_2) \\
(W4) & \{(p,T,T')\} & |\!\vdash\!- & (p,T\cdot U,T'\cdot U) \\
(W5) & \{(p,T,T')\} & |\!\vdash\!- & (p,U\cdot T,U\cdot T') \\
(W6) & \{(p,U_1\cdot T+U_2,T)\} & |\!\vdash\!- & (p,U_1^*\cdot U_2,T)
\end{array}$$

Fig. 3. Systems $\mathcal{B}$ (R0-R6) and $\mathcal{C}$ (W0-W6)

tion $|\!\vdash\!-_{\mathcal{B}} \subseteq \mathcal{P}(\mathcal{A}) \times \mathcal{A}$, as the set of all the instances of meta-rules of $\mathcal{B}$ where $S,T,T',T'',U \in \mathsf{D}\mathbb{B}\langle\langle\ V\ \rangle\rangle, (S_1,S_2),(T_1,T_2),(U_1,U_2) \in \mathsf{D}\mathbb{B}_{1,2}\langle\langle\ V\ \rangle\rangle, U_1 \neq \epsilon$. The binary relation $\vdash\!-_{\mathcal{B}}$ over $\mathcal{P}(\mathcal{A})$ is defined by: $\forall P,Q \in \mathcal{P}(\mathcal{A})$

$$P \vdash\!-_{\mathcal{B}} Q \Leftrightarrow (\forall q \in Q - P, \exists P' \subseteq P, \text{ such that } P' \,|\!\vdash\!-_{\mathcal{B}} q).$$

The relation $\overset{p}{\vdash\!-}_{\mathcal{B}}$ (for $p \in \mathbb{N}$) and $\overset{*}{\vdash\!-}_{\mathcal{B}}$ are then deduced from $\vdash\!-_{\mathcal{B}}$ as usual (and likewise the binary relations $|\!\vdash\!-_{\mathcal{C}}$, $\vdash\!-_{\mathcal{C}}$, $\overset{p}{\vdash\!-}_{\mathcal{C}}$, $\overset{*}{\vdash\!-}_{\mathcal{C}}$).

Lemma 3. *: For every $P,Q \in \mathcal{P}(\mathcal{A})$, $P \overset{*}{\vdash\!-}_{\mathcal{B}} Q \Rightarrow \mathrm{Div}(P) \leq \mathrm{Div}(Q)$.*

[5] i.e. up to some length for the terminal words

4.2 Self-provable sets

A subset $P \subseteq \mathcal{A}$ is said *self-provable*[6] iff:
$\forall (S,T) \in P, (S = \varepsilon) \Leftrightarrow (T = \varepsilon)$ and $\forall x \in X, P \overset{*}{\vdash\!\!-}_{\mathcal{B}} P \odot x$.

Lemma 4. *If P is self-provable then,* $\forall (S,T) \in P,\ S \equiv T$.

This follows easily from Lemma 3.

4.3 Comparison-forest

A *comparison-forest* is, informally speaking, a set of oriented trees labeled by weighted equations such that:
- a distinguished root, the *starting-node*, has a label of the form $(0, S, T)$, where $S, T \in \mathsf{DR}\mathbb{B}\langle\langle\ V\ \rangle\rangle$
- all other roots, the *unifier-nodes* have labels of the form $(0, u.M, v.M)$ where u, v are det. rat. row-vectors of dimension $(1, d)$ and M is a det rat matrix of dimension (d, d)
- non-root nodes have labels of the form (p, U, U') where $U, U' \in \mathsf{DR}\mathbb{B}\langle\langle\ V\ \rangle\rangle$.

Every node can have the status "open" or "closed". In case it is closed, property (3) of §4.5 is satisfied. Open nodes are leaves.

4.4 Tactics and strategies

The program maintains, at each step of the computation, a comparison-forest.
The program starts from the comparison-forest consisting of just one node, labeled by $(0, S, T)$. Then it iteratively modifies this c.f. by either:
1-closing an open node and adding new sons (the number of new sons ranges from 0 to the maximum cardinality of some class (modulo $\backsim$); at this stage, the sons are open.
2-discovering that an open node is obviously false (e.g (p, ε, S) where $S \neq \varepsilon$); a witness $u \in X^*$ of non-equivalence is thus propagated to the root r above this node

2-a if r is a unifier-node, this unifier is improved and all nodes of the forest that are below some node "using" the unifier are destroyed.

2-b if r is the starting-node, the witness u is thus a *witness of falsity* for the initial equation (S, T). The algorithm stops and returns the witness.
3-discovering that the forest has no open node. The set of equations of the forest is thus a *self-provable set*. The algorithm stops and returns the self-provable set.

The precise sequence of actions of the program will be determined by a *strategy*; in turn, the strategy will call *tactics* that are able to perform, given an open node of the current comparison-tree, one of the above kind of actions.

Tactics The main tactics yet implemented are summarized in table 1. The four last tactics lean on the notions exposed in section 3. Note that TCM implements the "triangulation process" described in [Sén01a, section 5].

[6] translation into our framework of the notion of "self-proving set of pairs" from [Cou83, p.162]

Trep	**argument**-node: n, open, labeled by (p, S, T) **context**:n',closed, labeled by (p', S, T) where $p' \leq p$. **action**: n is closed, "leaning on" n'.
Teq	**argument**-node: n, open, labeled by (p, T, T) **action**: n is closed.
TA	**argument**-node: n, open, labeled by (p, S, T) **action**: n is closed "leaning on his new sons". $\mathrm{Card}(X)$ sons are created, x-ith son is labeled by $(p+1, S \odot x, T \odot x)$
TD	**argument**-node: n, open, labeled by $(p, \sum_{j=1}^{d} A_j \cdot S_j, \sum_{j=1}^{n} A_j \cdot T_j)$, where A_j are $\smile$-equivalent non-terminals. **action**: n is closed, "leaning on his new sons". d sons are created, j-ith son is labeled by $(p+1, S_j, T_j)$
TCM	**argument**-node: n, open **context**:$n_0, n_1, \ldots, n_\ell$ is a path with $n_\ell = n$, n_i is labeled by $E_i = (\boldsymbol{\alpha}_i S, \boldsymbol{\beta}_i S)$ with a weight π_i where $\alpha_i, \beta_i \in \mathsf{DR}\mathbb{B}_{1,d}\langle\langle\ V\ \rangle\rangle, S \in \mathsf{DR}\mathbb{B}_{d,1}\langle\langle\ V\ \rangle\rangle$, **action**: a subsequence $n_0, n_{i_1}, \ldots, n_{i_r}$ is selected and r series S_j are eliminated as follows (w.l.o.g. we assume the eliminated indices are $1, \ldots, r$) $E_0 \odot w_1 = (S_1, \boldsymbol{\gamma_1} \cdot S),\ E_{i_1} D_1 \odot w_2 = (S_2, \boldsymbol{\gamma_2} \cdot S), \ldots,\ E_{i_{r-1}} D_1 \cdots D_{r-1} \odot w_r = (S_r, \boldsymbol{\gamma_r} \cdot S)$ each D_i is the dependency matrix associated to line i and vector $\boldsymbol{\gamma}_i$ Successive indices are chosen in such a way that $\pi_j \geq \pi_{j-1} + \lvert w_j \rvert + 1$. the sub-tree strictly beneath n_{i_r} is destroyed. $M := D_1 D_2 \cdots D_r$, n_r is given d new open sons n'_j labeled by: $(\pi_{i_r}, (\boldsymbol{\alpha}_{i_r} \cdot M)_j, (\boldsymbol{\beta}_{i_r} \cdot M)_j)$.
TCJ	**argument**-node: n, open **context** :idem as for TCM. In addition, $\forall i < \ell, \exists u_i \in X^*, (\boldsymbol{\alpha}_i \odot u_i, \boldsymbol{\beta}_i \odot u_i) = (\boldsymbol{\alpha}_{i+1}, \boldsymbol{\beta}_{i+1})$. **action**:a candidate mgu M for the vectors $\boldsymbol{\alpha}_0, \boldsymbol{\beta}_0$ is computed together with its cost c. Smallest index i such that $\pi_i \geq \pi_0 + c + 1$ is selected. Subtree strict. beneath n_i is destroyed. n_i is given d new open sons n'_j labeled by: $(\pi_i, (\boldsymbol{\alpha}_i \cdot M)_j, (\boldsymbol{\beta}_i \cdot M)_j)$
TCR	**argument**-node: n, open, labeled by (p, S, T) **context** :idem as for TCJ. **action**: M,$cost$,i computed and subtree destroyed as in TCJ. A new root n' is created, it is closed , n' is given d new open sons n'_j labeled by $(0, (\boldsymbol{\alpha}_0 \cdot M)_j, (\boldsymbol{\beta}_0 \cdot M)_j)$.
TSUN	**argument**-node: n, open, labeled by $(p, \boldsymbol{\alpha} S, \boldsymbol{\beta} S)$, where $\alpha, \beta \in \mathsf{DR}\mathbb{B}_{1,d}\langle\langle\ V\ \rangle\rangle, S \in \mathsf{DR}\mathbb{B}_{d,1}\langle\langle\ V\ \rangle\rangle$. and all components of $\boldsymbol{\alpha}, \boldsymbol{\beta}$ are null or have length one. **action**: n is closed.A candidate mgu M for the vectors $\boldsymbol{\alpha}, \boldsymbol{\beta}$ is computed The node n is given d new open sons, labeled by: $(p+1, S_j, (M \cdot S)_j)$.

Table 1.

Error tactics The tactics `Terror` is responsible for detecting that an open node is labeled by some trivially false equation. Then it returns "failure".
The tactics `Terror-dyn` also detects that an open node is false and then realizes action 2-a or 2-b of subsection 4.4

Strategies Two kinds of strategies have been developed. They all consist of combinations of the above tactics (or variants).The *static* strategies make only one guess of MGU (for each call to a computation of MGU) and either succeed to confirm this guess by terminating the proof, or discover an error and return "failure" as global result.
The *dynamic strategies* start each computation of mgu by a guess which might be improved by successive discoveries of errors by tactics `Terror-dyn`. Finally they return either a proof of the proposed equivalence or a witness of non-equivalence.

Implementation The module `proofs` defines a class `proof` that implements the notion of comparison-forest. The functions in charge of management of equations and MGU's are defined there. The module `tactics` implements the above defined tactics. In general we first defined abstract tactics that depend of functionnal arguments specifying which MGU-computation we use (static or dynamical, function guessing the MGU). Concrete tactics are obtained by instanciating these arguments. The module `strategies` defines a functional `make-strategy(maxsteps,error-tactics,*tactics)` which, in turn, produces concrete strategies.

4.5 Soundness

Our (meta)-proof that the program is *sound* i.e. that its positive outputs are really self-provable sets, leans on the auxiliary system $\mathcal{C}$. Let us use the following notation: for every $\pi, n \in \mathbb{N}, S, S' \in \mathsf{DRB}\langle\langle\ V\ \rangle\rangle$,

$$[\pi, S, S', n] = \{(\pi + |u|, S \odot u, S' \odot u) \mid u \in X^{\leq n}\}. \quad (2)$$

All the above tactics T enjoy the following fundamental property: if (π, S, S') is the weighted equation labelling a closed node of the forest t on which tactics T has been applied, then, for every terminal letter $x \in X$

$$\bigcup\{[p, U, U, n] \mid (p, U, U) \in \mathrm{im}(t), p + n \leq \pi\} \ \stackrel{*}{\vdash\!\!-}_{\mathcal{C}} \ \{(\pi + 1, S \odot x, S' \odot x)\} \quad (3)$$

A comparison forest is said *closed* when all its nodes are closed.

Theorem 6. *Let t be the closed forest computed by some strategy using only the tactics* $\mathrm{Trep}, \mathrm{Teq}, \mathrm{TA}, \mathrm{TD}, \mathrm{TCM}, \mathrm{TCJ}, \mathrm{TCR}, \mathrm{TSUN}$. *Then the set of equations labelling t is a* self-provable *set.*

Sketch of proof: Let us note P the set of weighted equations labelling t and let us consider the following property $\mathcal{Q}(\pi, n, p)$: $\forall S, S' \in \mathsf{DRB}\langle\langle\ V\ \rangle\rangle, P \ \stackrel{p}{\vdash\!\!-}_{\mathcal{C}} \ (\pi, S, S') \Rightarrow P \ \stackrel{*}{\vdash\!\!-}_{\mathcal{C}} \ [\pi, S, S', n]$.
Following the lines of the induction of [Sén01a, subsec. 10.2, eq (136)], one can prove by lexicographic induction over $(\pi+n, n, p)$ the statement: $\forall(\pi, n, p) \in \mathbb{N}^3, \mathcal{Q}(\pi, n, p)$.
□

5 Experiments

Out of 17 strategies yet experimented, let us show the behaviour of 5 typical ones over 7 positive examples and 5 negative examples. The selected strategies are characterized by 3 parameters: their algebraic tactics [TCM (*triangulation*) or TCJ (*jump*) or TSUN (*quasi division*)], the *connectedness* property for the forests they produce[7] and their *static* (versus *dynamic*) character (see section 4). The tests have been run on a computer `Intel(R) Xeon(R) CPU X5675 @ 3.07GHz`. In each positive example we show the number of nodes of the final proof, the number of tactics calls and the CPU-time (number of seconds or "oot" if $>= 3600$).

$pos\ example$	$ex0$	$ex1$	$ex2$	$ex3$	$ex4$	$ex5$	$ex6$
$size$	36	51	34	86	179	253	525
$trg, c, stat$	44/44/0.88	75/121/10	99/145/11	oot	oot	oot	oot
jp, c, dyn	44/44/0.79	75/117/4	67/123/8	100/1206/88	707/1067/476	oot	oot
jp, nc, dyn	44/44/0.8	60/102/3.5	61/117/7	64/1104/83	251/467/117	732/4220/1245	oot
$qdiv, nc, stat$	51/51/0.87	54/54/1	54/84/4	25/25/15	134/134/180	149/149/80	502/502/977
$qdiv, nc, dyn$	51/60/1	54/70/1	60/140/7	25/39/0.23	132/177/3	149/191/9	520/779/66

In each negative example, we show the length of the witness (for dynamic strategies[8]) and the CPU-time (in s.); we mention the performance of an exhaustive search, for comparison.

$neg\ example$	$ex2n$	$ex4n$	$ex4nn$	$ex4nnn$	$ex6n$
$size$	34	168	175	171	525
$trg, c, stat$	$-/2$	$-/2.7$	$-/52$	$-/17$	oot
jp, c, dyn	2/2.9	8/3.1	13/69	13/20	19/1609
jp, nc, dyn	2/2.8	8/3.2	11/60	13/20	19/2062
$qdiv, nc, stat$	$-/1.3$	$-/0.6$	$-/61$	$-/2.8$	$-/128$
$qdiv, nc, dyn$	4/4	7/15	13/35	13/29	23/170
$ex - srch$	4/0.02	4/0.08	7/2	7/1.3	oot

The size is the sum of the lengths of the rhs of the grammar.

6 Conclusion and perspectives

The program realized so far is a prototype where the low-level functions are far from being optimized. Its performance on grammar examples of 20 to 100 rules (and size in [30,500]) seems to show that the equivalence problem for dpda (and the computation of algebraic mgu's) is not out of reach from a practical point of view.
Among our perspectives of development we plan: to improve the core of the program by using rewriting techniques; to devise an example-generation module; to extend the program by modules implementing the reductions described in [Sén01b].
The program is open-source and we hope other authors will write their own complementary modules (e.g. the authors of [CCD13] are already implementing their reduction).

[7] depending on the fact that they launch a new tree for each new mgu-computation or not
[8] recall that the " failure" message sent by static strategies is unconclusive

Aknowledgments We thank I. Durand for her continuous advices concerning programming and X. Blanc for his lecture on program-testing.

References

[BCFR06] Cédric Bastien, Jurek Czyzowicz, Wojciech Fraczak, and Wojciech Rytter. Prime normal form and equivalence of simple grammars. *Theoret. Comput. Sci.*, 363(2):124–134, 2006.

[BG11] Stanislav Böhm and Stefan Göller. Language equivalence of deterministic real-time one-counter automata is NL-complete. In *Mathematical foundations of computer science 2011*, volume 6907 of *Lecture Notes in Comput. Sci.*, pages 194–205. Springer, Heidelberg, 2011.

[BGJ13] Stanislav Böhm, Stefan Göller, and Petr Jancar. Equivalence of deterministic one-counter automata is NL-complete. *CoRR*, abs/1301.2181, 2013.

[CCD13] R. Chrétien, V. Cortier, and S. Delaune. From security protocols to pushdown automata. *manuscript, submitted to ICALP 13*, 2013.

[Cou83] B. Courcelle. Fundamental properties of infinite trees. *Theoretical Computer Science 25*, pages 95–169, 1983.

[Har78] M.A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley, Reading, Mass., 1978.

[HHY79] M.A. Harrison, I.M. Havel, and A. Yehudai. On equivalence of grammars through transformation trees. *TCS 9*, pages 173–205, 1979.

[Jan12] Petr Jancar. Decidability of dpda language equivalence via first-order grammars. In *LICS*, pages 415–424, 2012.

[Mei89] Y.V. Meitus. The equivalence problem for real-time strict deterministic pushdown automata. *Kibernetika 5 (in russian, english translation in Cybernetics and Systems analysis)*, pages 14–25, 1989.

[Sén97] G. Sénizergues. The Equivalence Problem for Deterministic Pushdown Automata is Decidable. In *Proceedings ICALP 97*, pages 671–681. Springer, LNCS 1256, 1997.

[Sén01a] G. Sénizergues. L(A) = L(B) ? decidability results from complete formal systems. *Theoretical Computer Science*, 251:1–166, 2001.

[Sén01b] G. Sénizergues. Some applications of the decidability of dpda's equivalence. In *Proceedings MCU'01*, volume 2055 of *LNCS*, pages 114–132. Springer-Verlag, 2001.

[Sén03] G. Sénizergues. The equivalence problem for t-turn dpda is co-NP. In *Proceedings ICALP'03*, volume 2719 of *LNCS*, pages 478–489. Springer-Verlag, 2003.

[Sén05] G. Sénizergues. The bisimulation problem for equational graphs of finite out-degree. *SIAM J. Comput.*, 34(5):1025–1106 (electronic), 2005.

[Sti02] C. Stirling. Deciding DPDA Equivalence is Primitive Recursive. In *Proceedings ICALP 02*, pages 821–832. Springer, LNCS 2380, 2002.

Annex

Let us describe here the examples that section 5 is referring to.

6.1 Positive examples

Examples 0,1,2 were constructed in parallel with a hand-made proof, at the time G.S. was writing paper [Sén01a]. Example 1 is treated in [Sén01a, section 12.1], example 2 is treated in [Sén01a, section 12.2]. Examples 3,5,6 were constructed in a somewhat technical way. The leading idea is that, starting from a system of equations and a solution in words, we can define a d.c.f. grammar G that makes this system of equations a self-provable set (w.r.t. G).
Example 4 was obtained from an easier example by folding-unfolding.

```
A0=auto(
            ('x','a','b','c','d','z','#'),
            ('q0','q1','q2','q3','q4','p','qb'),
            ('O','A','B','C','Z','Ab','Zb'),
            (
                ('q0','O','#','q0','Z'),
                ('q1','O','#','q1','Zb'),
                ('q0','Z','x','q0','A','Z'),
                ('q0','A','a','q2'),
                ('q0','A','b','q0','B'),
                ('q2','Z','z','p'),
                ('q0','B','a','q2'),
                ('q1','Zb','x','q1','Ab','Zb'),
                ('q1','Ab','a','q3'),
                ('q1','Ab','b','q4'),
                ('q3','Zb','z','p'),
                ('q4','Zb','a','q3','Zb'),
                ('q0','A','c','q0','C','A'),
                ('q0','C','c','q0','C','C'),
                ('q0','C','d','qb'),
                ('qb','C','d','qb'),
                ('qb','A','a','q2'),
                ('q1','Ab','c','q0','C','Ab'),
                ('qb','Ab','a','q3')
            ),
            ("q0"),
            ("p")
        )

#G0
Non-terminal symbols : [<q0-B-q2> ][<qb-A-q2> ][<q1-Ab-q3> <q1-Ab-q4> ]
```

```
[<qb-C-qb> ][<q0-Z-p> ][<q0-A-q2> ][<q1-O-p> ][<q0-O-p> ][<q1-Zb-p> ]
[<q0-C-qb> ][<qb-Ab-q3> ][<q3-Zb-p> ][<q4-Zb-p> ][<q2-Z-p> ]

Terminal symbols : # a b c d x z
Rewriting rules:
<q0-A-q2>::=a
 <q0-A-q2>::=b<q0-B-q2>
 <q0-A-q2>::=c<q0-C-qb><qb-A-q2>
 <q0-B-q2>::=a
 <q0-C-qb>::=c<q0-C-qb><qb-C-qb>
 <q0-C-qb>::=d
 <q0-O-p>::=#<q0-Z-p>
 <q0-Z-p>::=x<q0-A-q2><q2-Z-p>
 <q1-Ab-q3>::=a
 <q1-Ab-q3>::=c<q0-C-qb><qb-Ab-q3>
 <q1-Ab-q4>::=b
 <q1-O-p>::=#<q1-Zb-p>
 <q1-Zb-p>::=x<q1-Ab-q3><q3-Zb-p>
 <q1-Zb-p>::=x<q1-Ab-q4><q4-Zb-p>
 <q2-Z-p>::=z
 <q3-Zb-p>::=z
 <q4-Zb-p>::=a<q3-Zb-p>
 <qb-A-q2>::=a
 <qb-Ab-q3>::=a
 <qb-C-qb>::=d

Axiom: <q0-O-p>

S=word_to_mat(["<q0-O-p>"],G0)
T=word_to_mat(["<q1-O-p>"],G0)
```

```
A1=auto(
            (’x’,’a’,’b’,’c’,’d’,’t’,’tb’),
            (’q0’,’q1’,’q2’,’q3’,’qb’),
            (’O’,’A’,’B’,’D’,’T’),
            (
                (’q0’,’O’,’x’,’q0’,’A’,’O’),
                (’q1’,’O’,’b’,’q1’,’O’),
                (’q2’,’O’,’d’,’qb’),
                (’q0’,’T’,’t’,’q0’,’T’,’T’),
                (’q0’,’A’,’a’,’q1’),
                (’q1’,’O’,’c’,’q1’,’D’),
                (’q0’,’A’,’t’,’q0’,’T’,’A’),
                (’q0’,’A’,’c’,’q2’),
                (’q1’,’D’,’d’,’qb’),
                (’q0’,’T’,’tb’,’q0’),
                (’q1’,’O’,’x’,’q1’,’A’,’O’),
                (’q1’,’A’,’c’,’q2’),
                (’q1’,’A’,’t’,’q1’,’T’,’A’),
                (’q1’,’A’,’a’,’q1’,’B’),
                (’q1’,’B’,’x’,’q1’,’A’),
                (’q1’,’T’,’tb’,’q1’),
                (’q1’,’B’,’b’,’q1’),
                (’q1’,’B’,’c’,’q2’),
                (’q1’,’T’,’t’,’q1’,’T’,’T’),
            ),
            ("q0"),
            ("qb")
        )
#G1
Non-terminal symbols : [<q1-A-q1> <q1-A-q2> ][<q0-T-q0> ][<q0-A-q1> <q0-A-q2>
[<q1-T-q1> ][<q1-D-qb> ][<q0-O-qb> ][<q2-O-qb> ][<q1-O-qb> ][<q1-B-q1> <q1-B-

Terminal symbols : a b c d t tb x
Rewriting rules:
<q0-A-q1>::=a
 <q0-A-q1>::=t<q0-T-q0><q0-A-q1>
 <q0-A-q2>::=c
 <q0-A-q2>::=t<q0-T-q0><q0-A-q2>
 <q0-O-qb>::=x<q0-A-q1><q1-O-qb>
 <q0-O-qb>::=x<q0-A-q2><q2-O-qb>
 <q0-T-q0>::=t<q0-T-q0><q0-T-q0>
 <q0-T-q0>::=tb
 <q1-A-q1>::=a<q1-B-q1>
 <q1-A-q1>::=t<q1-T-q1><q1-A-q1>
 <q1-A-q2>::=a<q1-B-q2>
```

```
 <q1-A-q2>::=c
 <q1-A-q2>::=t<q1-T-q1><q1-A-q2>
 <q1-B-q1>::=b
 <q1-B-q1>::=x<q1-A-q1>
 <q1-B-q2>::=c
 <q1-B-q2>::=x<q1-A-q2>
 <q1-D-qb>::=d
 <q1-O-qb>::=b<q1-O-qb>
 <q1-O-qb>::=c<q1-D-qb>
 <q1-O-qb>::=x<q1-A-q1><q1-O-qb>
 <q1-O-qb>::=x<q1-A-q2><q2-O-qb>
 <q1-T-q1>::=t<q1-T-q1><q1-T-q1>
 <q1-T-q1>::=tb
 <q2-O-qb>::=d

Axiom: <q0-O-qb>

S=word_to_mat(["<q0-O-qb>"],G1)
T=word_to_mat(["<q1-O-qb>"],G1)
```

```
A2=auto(
            (’x’,’a’,’b’,’#’),
            (’q1’,’q2’,’q3’,’q3p’,’q3b’,’q4’,’q5’,’qb’),
            (’O’,’A’,’B’),
            (
                (’q1’,’O’,’#’,’q1’,’A’,’O’),
                (’q2’,’O’,’#’,’q2’,’A’,’O’),
                (’q1’,’A’,’a’,’q3’),
                (’q3’,’A’,’a’,’q3’),
                (’q3’,’O’,’a’,’q3p’,’O’),
                (’q1’,’A’,’b’,’q5’),
                (’q5’,’A’,"eps",’q5’),
                (’q3p’,’O’,’a’,’q3b’,’O’),
                (’q1’,’A’,’x’,’q1’,’A’,’A’),
                (’q5’,’O’,"eps",’q3’),
                (’q3b’,’O’,’a’,’q3’),
                (’q2’,’A’,’a’,’q4’,’A’,’A’),
                (’qb’,’O’,"eps",’q3’),
                (’q4’,’O’,’a’,’q3’),
                (’q2’,’A’,’b’,’qb’),
                (’q2’,’A’,’x’,’q2’,’A’,’A’),
                (’qb’,’A’,"eps",’qb’),
                (’q4’,’A’,’a’,’q4’)
            ),
            ("q1"),
            ("q3")
        )
#G2
Non-terminal symbols : [<q2-A-q4> <q2-A-qb> ][<q4-O-q3> ][<q1-O-q3> ][<q2-O-q
[<q3p-O-q3> ][<q4-A-q4> ][<q3-O-q3> ][<q3-A-q3> ][<q1-A-q3> <q1-A-q5> ][<q3b-

Terminal symbols : # a b x
Rewriting rules:
<q1-A-q3>::=a
 <q1-A-q3>::=x<q1-A-q3><q3-A-q3>
 <q1-A-q5>::=b
 <q1-A-q5>::=x<q1-A-q5>
 <q1-O-q3>::=#<q1-A-q3><q3-O-q3>
 <q1-O-q3>::=#<q1-A-q5>
 <q2-A-q4>::=a<q4-A-q4><q4-A-q4>
 <q2-A-q4>::=x<q2-A-q4><q4-A-q4>
 <q2-A-qb>::=b
 <q2-A-qb>::=x<q2-A-qb>
 <q2-O-q3>::=#<q2-A-q4><q4-O-q3>
 <q2-O-q3>::=#<q2-A-qb>
```

```
 <q3-A-q3>::=a
 <q3-O-q3>::=a<q3p-O-q3>
 <q3b-O-q3>::=a
 <q3p-O-q3>::=a<q3b-O-q3>
 <q4-A-q4>::=a
 <q4-O-q3>::=a

Axiom: <q1-O-q3>
S=word_to_mat(["<q1-O-q3>"],G2)
T=word_to_mat(["<q2-O-q3>"],G2)
```

```
#G3
    #equiv over nt alphabet
    rep={}
    rep["S"]="S"
    rep["T"]="T"
    rep["A1"]="A1"
    rep["A2"]="A1"
    G3=gram(
    ["S","T","A1","A2"],
    ["x","y","z","xb","zb"],
    [["S",["x","S","S","S","S"]],
     ["S",["y","A1","S","S","S","S","S"]],
     ["S",["y","A2","S","S","S"]],
     ["S",["z","S","S"]],["S",["xb"]],
     ["S",["zb","S","S"]],
     ["A1",["x","S","A1","S","S"]],
     ["A1",["x","S","A2"]],
     ["A1",["y","S","A1","S","S"]],
     ["A1",["y","S","A2"]],
     ["A1",["z"]],
     ["A2",["xb"]],
     ["A2",["zb","A1","S","S","S"]],
     ["A2",["zb","A2","S"]],
     ["T",["x","S","A1","S","S","S","S"]],
     ["T",["x","S","A2","S","S"]],
     ["T",["y","S","A1","S","S","S","S"]],
     ["T",["y","S","A2","S","S"]],
     ["T",["z","S","S"]],
     ["T",["xb"]],
     ["T",["zb","A1","S","S","S"]],
     ["T",["zb","A2","S"]]],
    "S",rep
    )

    S=word_to_mat(["S"],G3)
    T=word_to_mat(["T"],G3)
```

```
#G4
    #equiv over nt alphabet
    rep={}
    rep["S1"]="S1"
    rep["T1"]="S1"
    rep["S2"]="S2"
    rep["S3"]="S3"
    rep["S4"]="S4"
    rep["T4"]="S4"
    rep["S5"]="S5"
    rep["T5"]="S5"
    rep["S6"]="S6"
    rep["X"]="X"
    rep["Y"]="X"
    rep["Z"]="X"
    G4=gram(
    ["S1","T1","S2","S3","S4","T4","S5","T5","S6","X","Y","Z","T"],
    ["x","y","z","t"],
    [["X",["x"]],["Y",["y"]],["Z",["z"]],

     ["S1",["y","S2","Y","S2","S1"]],
     ["S1",["y","S2","X","S3"]],
     ["S1",["x","T1","S2"]],
     ["S1",["x","S1","S2"]],

     ["T1",["y","S2","Y","S2","T1"]],
     ["T1",["y","S2","Z"]],
     ["T1",["z"]],

     ["S2",["y","S6","T1","S1"]],
     ["S2",["y","S6","T1","T1"]],
     ["S2",["z","S1"]],
     ["S2",["z","T1"]],

     ["S3",["y","S2","Y","S2","T1","T5","S1"]],
     ["S3",["y","S2","Z","T5","S1"]],
     ["S3",["z","T5","S1"]],
     ["S3",["y","S2","Y","S2","T1","T5","T1"]],
     ["S3",["y","S2","Z","T5","T1"]],
     ["S3",["z","T5","T1"]],
     ["S3",["y","S2","Y","S2","S1","S2"]],
     ["S3",["y","S2","X","S3","S2"]],
     ["S3",["x","T1","S2","S2"]],
     ["S3",["x","S1","S2","S2"]],
```

```
 ["S4",["y","S2","S1"]],
 ["S4",["x","S3"]],

 ["T4",["y","S2","T1"]],
 ["T4",["z"]],

 ["S5",["y","T1","S1","S1"]],
 ["S5",["y","T1","T1","S1"]],
 ["S5",["x","S3"]],

 ["T5",["y","T1","S1","T1"]],
 ["T5",["y","T1","T1","T1"]],
 ["T5",["z"]],

 ["S6",["y","S2","Y","S2","T1","Y","S2","S1"]],
 ["S6",["y","S2","Z","Y","S2","S1"]],
 ["S6",["z","Y","S2","S1"]],
 ["S6",["y","S2","Y","S2","T1","Y","S2","T1"]],
 ["S6",["y","S2","Z","Y","S2","T1"]],
 ["S6",["z","Y","S2","T1"]],
 ["S6",["y","S2","Y","S2","T1","X","S3"]],
 ["S6",["y","S2","Z","X","S3"]],
 ["S6",["z","X","S3"]],
 ["S6",["y","S2","Y","S2","T1","Z"]],
 ["S6",["y","S2","Z","Z"]],
 ["S6",["z","Z"]],

 ],
"S1",rep
)
S=word_to_mat(["S1"],G4)
T=word_to_mat(["S5"],G4)
```

```
#G5
    #equiv over nt alphabet
    rep={}
    rep["U1"]="U1"
    rep["U2"]="U1"
    rep["U3"]="U1"
    rep["U4"]="U1"
    rep["V1"]="V1"
    rep["V2"]="V1"
    rep["V3"]="V1"
    rep["V4"]="V1"
    rep["C1"]="C1"
    rep["C2"]="C1"
    rep["D1"]="D1"
    rep["D2"]="D1"
    rep["X"]="X"
    rep["Y"]="X"
    rep["Z"]="X"
    rep["T"]="X"
    G5=gram(
    ["U1","U2","U3","U4","V1","V2","V3","V4","C1","C2","D1","D2","X","Y","Z",
    ["x","y","z","t"],
    [["X",["x"]],["Y",["y"]],["Z",["z"]],["T",["t"]],
     ["U1",["x","U1","X","V1","C2","Y","V3"]],
     ["U1",["x","U1","X","V1","C2","X"]],
     ["U1",["x","U1","Y","Y","V3"]],
     ["U1",["x","U1","Y","X"]],
     ["U1",["y","U3","X","V1","C2","Y","V3"]],
     ["U1",["y","U3","X","V1","C2","X"]],
     ["U1",["y","U3","Y","Y","V3"]],
     ["U1",["y","U3","Y","X"]],
     ["U1",["z","X","V1","C2","Y","V3"]],
     ["U1",["z","X","V1","C2","X"]],
     ["U1",["z","Y","Y","V3"]],
     ["U1",["z","Y","X"]],
     ["U2",["x","U1","X","V1","C2","Y","V4"]],
     ["U2",["x","U1","X","V1","C2","Z"]],
     ["U2",["x","U1","Y","Y","V4"]],
     ["U2",["x","U1","Y","Z"]],
     ["U2",["y","U3","X","V1","C2","Y","V4"]],
     ["U2",["y","U3","X","V1","C2","Z"]],
     ["U2",["y","U3","Y","Y","V4"]],
     ["U2",["y","U3","Y","Z"]],
     ["U2",["z","X","V1","C2","Y","V4"]],
     ["U2",["z","X","V1","C2","Z"]],
```

```
["U2",["z","Y","Y","V4"]],
["U2",["z","Y","Z"]],

["U3",["x","U1","X","V2"]],
["U3",["x","U1","Z"]],
["U3",["y","U3","X","V2"]],
["U3",["y","U3","Z"]],
["U3",["z","X","V2"]],
["U3",["z","Z"]],

["U4",["x","U3"]],
["U4",["y","U4","C2"]],
["U4",["t"]],

["V1",["x","V1","D1","X","U2"]],
["V1",["x","V1","D1","Y"]],
["V1",["y","V2","X","U2"]],
["V1",["y","V2","Y"]],
["V1",["z","X","U2"]],
["V1",["z","Y"]],

["V2",["x","V1","D1","X","U3"]],
["V2",["x","V1","D1","Z"]],
["V2",["y","V2","X","U3"]],
["V2",["y","V2","Z"]],
["V2",["z","X","U3"]],
["V2",["z","Z"]],

["V3",["x","V2","Y","U4","D1"]],
["V3",["x","V2","X"]],
["V3",["y","V4","Y","U4","D1"]],
["V3",["y","V4","X"]],
["V3",["t","Y","U4","D1"]],
["V3",["t","X"]],

["V4",["x","V2","Y","U4","D2"]],
["V4",["x","V2","Z"]],
["V4",["y","V4","Y","U4","D2"]],
["V4",["y","V4","Z"]],
["V4",["t","Y","U4","D2"]],
["V4",["t","Z"]],

["C1",["y","U4","C1"]],
["C1",["x"]],
["C2",["y","U4","C2"]],
```

```
 ["C2",["z"]],

 ["D1",["y","V3"]],
 ["D1",["x"]],
 ["D2",["y","V4"]],
 ["D2",["z"]]
 ],
"U1",rep
)
S=word_to_mat(["U3"],G5)
T=word_to_mat(["V2"],G5)
```

```
#G6
    #equiv over nt alphabet
    rep={}
    rep["U1"]="U1"
    rep["U2"]="U1"
    rep["U3"]="U1"
    rep["U4"]="U1"
    rep["V1"]="V1"
    rep["V2"]="V1"
    rep["V3"]="V1"
    rep["V4"]="V1"
    rep["C1"]="C1"
    rep["C2"]="C1"
    rep["X"]="X"
    rep["Y"]="X"
    rep["Z"]="X"
    G6=gram(
    ["U1","U2","U3","U4","V1","V2","V3","V4","C1","C2","X","Y","Z"],
    ["x","y","z"],
    [["X",["x"]],["Y",["y"]],["Z",["z"]],
    ["U1",["x","V1","C1","Y","U2","X","U1"]],
    ["U1",["x","V1","C1","Y","U2","X","U2"]],
    ["U1",["x","V1","C1","Y","U3","X","U1"]],
    ["U1",["x","V1","C1","Y","U3","X","U2"]],
    ["U1",["x","V1","C1","Z","X","U1"]],
    ["U1",["x","V1","C1","Z","X","U2"]],
    ["U1",["x","V2","Y","U2","X","U1"]],
    ["U1",["x","V2","Y","U2","X","U2"]],
    ["U1",["x","V2","Y","U3","X","U1"]],
    ["U1",["x","V2","Y","U3","X","U2"]],
    ["U1",["x","V2","Z","X","U1"]],
    ["U1",["x","V2","Z","X","U2"]],
    ["U1",["x","V3","Y","U2","X","U1"]],
    ["U1",["x","V3","Y","U2","X","U2"]],
    ["U1",["x","V3","Y","U3","X","U1"]],
    ["U1",["x","V3","Y","U3","X","U2"]],
    ["U1",["x","V3","Z","X","U1"]],
    ["U1",["x","V3","Z","X","U2"]],
    ["U1",["y","Y","U2","X","U1"]],
    ["U1",["y","Y","U2","X","U2"]],
    ["U1",["y","Y","U3","X","U1"]],
    ["U1",["y","Y","U3","X","U2"]],
    ["U1",["y","Z","X","U1"]],
    ["U1",["y","Z","X","U2"]],
    ["U2",["x","V1","C1","Y","U2","Y","U4","C1"]],
```

```
["U2",["x","V1","C1","Y","U2","Y","U3"]],
["U2",["x","V1","C1","Y","U3","Y","U4","C1"]],
["U2",["x","V1","C1","Y","U3","Y","U3"]],
["U2",["x","V1","C1","Z","Y","U4","C1"]],
["U2",["x","V1","C1","Z","Y","U3"]],
["U2",["x","V2","Y","U2","Y","U4","C1"]],
["U2",["x","V2","Y","U2","Y","U3"]],
["U2",["x","V2","Y","U3","Y","U4","C1"]],
["U2",["x","V2","Y","U3","Y","U3"]],
["U2",["x","V2","Z","Y","U4","C1"]],
["U2",["x","V2","Z","Y","U3"]],
["U2",["x","V3","Y","U2","Y","U4","C1"]],
["U2",["x","V3","Y","U2","Y","U3"]],
["U2",["x","V3","Y","U3","Y","U4","C1"]],
["U2",["x","V3","Y","U3","Y","U3"]],
["U2",["x","V3","Z","Y","U4","C1"]],
["U2",["x","V3","Z","Y","U3"]],
["U2",["y","Y","U2","Y","U4","C1"]],
["U2",["y","Y","U2","Y","U3"]],
["U2",["y","Y","U3","Y","U4","C1"]],
["U2",["y","Y","U3","Y","U3"]],
["U2",["y","Z","Y","U4","C1"]],
["U2",["y","Z","Y","U3"]],
["U2",["x","V1","C1","Y","U2","Z"]],
["U2",["x","V1","C1","Y","U3","Z"]],
["U2",["x","V1","C1","Z","Z"]],
["U2",["x","V2","Y","U2","Z"]],
["U2",["x","V2","Y","U3","Z"]],
["U2",["x","V2","Z","Z"]],
["U2",["x","V3","Y","U2","Z"]],
["U2",["x","V3","Y","U3","Z"]],
["U2",["x","V3","Z","Z"]],
["U2",["y","Y","U2","Z"]],
["U2",["y","Y","U3","Z"]],
["U2",["y","Z","Z"]],
["U3",["x","V1","C1","Y","U4","C1"]],
["U3",["x","V1","C1","Y","U4","C2"]],
["U3",["x","V2","Y","U4","C1"]],
["U3",["x","V2","Y","U4","C2"]],
["U3",["x","V3","Y","U4","C1"]],
["U3",["x","V3","Y","U4","C2"]],
["U3",["y","Y","U4","C1"]],
["U3",["y","Y","U4","C2"]],
["U4",["x","V1","C2"]],
["U4",["x","V4"]],
```

```
["V1",["x","U1","Y","V1","C2"]],
["V1",["x","U1","Y","V2"]],
["V1",["x","U1","Z"]],
["V1",["x","U3","Y","V1","C2"]],
["V1",["x","U3","Y","V2"]],
["V1",["x","U3","Z"]],
["V1",["x","U4","C1","Y","V1","C2"]],
["V1",["x","U4","C1","Y","V2"]],
["V1",["x","U4","C1","Z"]],
["V1",["y","Y","V1","C2"]],
["V1",["y","Y","V2"]],
["V1",["y","Z"]],
["V2",["x","U1","Y","V3"]],
["V2",["x","U1","Y","V4"]],
["V2",["x","U3","Y","V3"]],
["V2",["x","U3","Y","V4"]],
["V2",["x","U4","C1","Y","V3"]],
["V2",["x","U4","C1","Y","V4"]],
["V2",["y","Y","V3"]],
["V2",["y","Y","V4"]],
["V3",["x","U2","X","V1","C1"]],
["V3",["x","U2","X","V1","C2"]],
["V3",["x","U4","C2","X","V1","C1"]],
["V3",["x","U4","C2","X","V1","C2"]],
["V4",["x","U2","Y","V2"]],
["V4",["x","U2","Y","V3"]],
["V4",["x","U2","Z"]],
["V4",["x","U4","C2","Y","V2"]],
["V4",["x","U4","C2","Y","V3"]],
["V4",["x","U4","C2","Z"]],
["C1",["x","U1"]],
["C1",["x","U2"]],
["C2",["y","U4","C1"]],
["C2",["y","U3"]],
["C2",["z"]]],
"U1",rep
)
S=word_to_mat(["U1"],G6)
T=word_to_mat(["V1","C1"],G6)
```

6.2 Negative examples

Example 2n is built from ex2 by a small random modification. The other examples were obtained casually, in the process of constructing positive ones: ex4n , ex4nn, ex4nnn are early (incorrect) versions of ex4 and ex6n is an early (incorrect) version of ex6. We have left for future work the design of negative examples, viewed as challenges for the dynamical strategies (and surely non-treatable by exhaustive comparison). Note that, by chance, ex6n already demonstrates the advantages of a dynamical strategy over an exhaustive search.

```
#G2n
Non-terminal symbols : [<q2-A-q4> <q2-A-qb> ][<q4-O-q3> ][<q1-O-q3> ]
[<q2-O-q3> ][<q3p-O-q3> ][<q4-A-q4> ][<q3-O-q3> ][<q3-A-q3> ]
[<q1-A-q3> <q1-A-q5> ][<q3b-O-q3> ]

Terminal symbols : # a b x
Rewriting rules:
<q1-A-q3>::=a
 <q1-A-q3>::=x<q1-A-q3><q3-A-q3>
 <q1-A-q5>::=b
 <q1-A-q5>::=x<q1-A-q5>
 <q1-O-q3>::=#<q1-A-q3><q3-O-q3>
 <q1-O-q3>::=#<q1-A-q5>
 <q2-A-q4>::=a<q4-A-q4><q4-A-q4>
 <q2-A-q4>::=x<q2-A-q4><q4-A-q4>
 <q2-A-qb>::=b
 <q2-A-qb>::=x<q2-A-qb>
 <q2-O-q3>::=#<q2-A-q4><q4-O-q3>
 <q2-O-q3>::=#<q2-A-qb>
 <q3-A-q3>::=a
 <q3-O-q3>::=a<q3p-O-q3>
 <q3b-O-q3>::=a
 <q3p-O-q3>::=a<q3b-O-q3>
 <q4-A-q4>::=a
 <q4-O-q3>::=a

S=word_to_mat(["<q1-O-q3>","<q1-O-q3>","<q1-O-q3>"],G2n)
T=word_to_mat(["<q2-O-q3>","<q1-O-q3>"],G2n)
```

```
#G4n
    #equiv over nt alphabet
    rep={}
    rep["S1"]="S1"
    rep["T1"]="S1"
    rep["S2"]="S2"
    rep["S3"]="S3"
    rep["S4"]="S4"
    rep["T4"]="S4"
    rep["S5"]="S5"
    rep["T5"]="S5"
    rep["S6"]="S6"
    rep["X"]="X"
    rep["Y"]="X"
    rep["Z"]="X"
    G4n=gram(
    ["S1","T1","S2","S3","S4","T4","S5","T5","S6","X","Y","Z","T"],
    ["x","y","z","t"],
    [["X",["x"]],["Y",["y"]],["Z",["z"]],

     ["S1",["y","S2","Z","S1"]],
     ["S1",["y","S2","Y"]],
     ["S1",["x","T1","S2"]],
     ["S1",["x","S1","S2"]],

     ["T1",["y","S2","Z","T1"]],
     ["T1",["y","S2","X"]],
     ["T1",["z"]],

     ["S2",["y","S6","T1","S1"]],
     ["S2",["y","S6","T1","T1"]],
     ["S2",["z","S1"]],
     ["S2",["z","T1"]],

     ["S3",["y","S2","Z","T1","T5","S1"]],
     ["S3",["y","S2","X","T5","S1"]],
     ["S3",["z","T5","S1"]],
     ["S3",["y","S2","Z","T1","T5","T1"]],
     ["S3",["y","S2","X","T5","T1"]],
     ["S3",["z","T5","T1"]],
     ["S3",["y","S2","Z","S1","S2"]],
     ["S3",["y","S2","Y","S2"]],
     ["S3",["x","T1","S2","S2"]],
     ["S3",["x","S1","S2","S2"]],
```

```
 ["S4",["y","S2","S1"]],
 ["S4",["x","S3"]],

 ["T4",["y","S2","T1"]],
 ["T4",["z"]],

 ["S5",["y","T1","S1","S1"]],
 ["S5",["y","T1","T1","S1"]],
 ["S5",["x","S3"]],

 ["T5",["y","T1","S1","T1"]],
 ["T5",["y","T1","T1","T1"]],
 ["T5",["z"]],

 ["S6",["y","S2","Z","T1","Y","S2","S1"]],
 ["S6",["y","S2","X","Y","S2","S1"]],
 ["S6",["z","Y","S2","S1"]],
 ["S6",["y","S2","Z","T1","Y","S2","T1"]],
 ["S6",["y","S2","X","Y","S2","T1"]],
 ["S6",["z","Y","S2","T1"]],

 ["S6",["y","S2","Z","T1","X","S3"]],
 ["S6",["y","S2","X","X","S3"]],
 ["S6",["z","X","S3"]],
 ["S6",["y","S2","Z","T1","Z"]],
 ["S6",["y","S2","X","Z"]],
 ["S6",["z","Z"]],

 ],
"S1",rep
)
S=word_to_mat(["S1"],G4n)
T=word_to_mat(["S5"],G4n)
```

```
#G4nn
    #equiv over nt alphabet
    rep={}
    rep["S1"]="S1"
    rep["T1"]="S1"
    rep["S2"]="S2"
    rep["S3"]="S3"
    rep["S4"]="S4"
    rep["T4"]="S4"
    rep["S5"]="S5"
    rep["T5"]="S5"
    rep["S6"]="S6"
    rep["X"]="X"
    rep["Y"]="X"
    rep["Z"]="X"
    G4nn=gram(
    ["S1","T1","S2","S3","S4","T4","S5","T5","S6","X","Y","Z","T"],
    ["x","y","z","t"],
    [["X",["x"]],["Y",["y"]],["Z",["z"]],

     ["S1",["y","S2","Y","S2","S1"]],
     ["S1",["y","S2","X","S3"]],
     ["S1",["x","T1","S2"]],
     ["S1",["x","S1","S2"]],

     ["T1",["y","S2","Y","S2","T1"]],
     ["T1",["y","S2","Z"]],
     ["T1",["z"]],

     ["S2",["y","S6","T1","S1"]],
     ["S2",["y","S6","T1","T1"]],
     ["S2",["z","S1"]],
     ["S2",["z","T1"]],

     ["S3",["y","S2","Z","T1","T5","S1"]],
     ["S3",["y","S2","X","T5","S1"]],
     ["S3",["z","T5","S1"]],
     ["S3",["y","S2","Z","T1","T5","T1"]],
     ["S3",["y","S2","X","T5","T1"]],
     ["S3",["z","T5","T1"]],
     ["S3",["y","S2","Z","S1","S2"]],
     ["S3",["y","S2","Y","S2"]],
     ["S3",["x","T1","S2","S2"]],
     ["S3",["x","S1","S2","S2"]],
```

```
 ["S4",["y","S2","S1"]],
 ["S4",["x","S3"]],

 ["T4",["y","S2","T1"]],
 ["T4",["z"]],

 ["S5",["y","T1","S1","S1"]],
 ["S5",["y","T1","T1","S1"]],
 ["S5",["x","S3"]],

 ["T5",["y","T1","S1","T1"]],
 ["T5",["y","T1","T1","T1"]],
 ["T5",["z"]],

 ["S6",["y","S2","Y","S2","T1","Y","S2","S1"]],
 ["S6",["y","S2","Z","Y","S2","S1"]],
 ["S6",["z","Y","S2","S1"]],
 ["S6",["y","S2","Y","S2","T1","Y","S2","T1"]],
 ["S6",["y","S2","Z","Y","S2","T1"]],
 ["S6",["z","Y","S2","T1"]],

 ["S6",["y","S2","Y","S2","T1","X","S3"]],
 ["S6",["y","S2","Z","X","S3"]],
 ["S6",["z","X","S3"]],
 ["S6",["y","S2","Y","S2","T1","Z"]],
 ["S6",["y","S2","Z","Z"]],
 ["S6",["z","Z"]],

 ],
"S1",rep
)
print(G)
S=word_to_mat(["S1"],G4nn)
T=word_to_mat(["S5"],G4nn)
```

```
#G4nnn
    #equiv over nt alphabet
    rep={}
    rep["S1"]="S1"
    rep["T1"]="S1"
    rep["S2"]="S2"
    rep["S3"]="S3"
    rep["S4"]="S4"
    rep["T4"]="S4"
    rep["S5"]="S5"
    rep["T5"]="S5"
    rep["S6"]="S6"
    rep["X"]="X"
    rep["Y"]="X"
    rep["Z"]="X"
    G4nnn=gram(
    ["S1","T1","S2","S3","S4","T4","S5","T5","S6","X","Y","Z","T"],
    ["x","y","z","t"],
    [["X",["x"]],["Y",["y"]],["Z",["z"]],

     ["S1",["y","S2","Y","S2","S1"]],
     ["S1",["y","S2","X","S3"]],
     ["S1",["x","T1","S2"]],
     ["S1",["x","S1","S2"]],

     ["T1",["y","S2","Y","S2","T1"]],
     ["T1",["y","S2","Z"]],
     ["T1",["z"]],

     ["S2",["y","S6","T1","S1"]],
     ["S2",["y","S6","T1","T1"]],
     ["S2",["z","S1"]],
     ["S2",["z","T1"]],

     ["S3",["y","S2","Z","T1","T5","S1"]],
     ["S3",["y","S2","X","T5","S1"]],
     ["S3",["z","T5","S1"]],
     ["S3",["y","S2","Z","T1","T5","T1"]],
     ["S3",["y","S2","X","T5","T1"]],
     ["S3",["z","T5","T1"]],
     ["S3",["y","S2","Z","S1","S2"]],
     ["S3",["y","S2","Y","S2"]],
     ["S3",["x","T1","S2","S2"]],
     ["S3",["x","S1","S2","S2"]],
```

```
 ["S4",["y","S2","S1"]],
 ["S4",["x","S3"]],

 ["T4",["y","S2","T1"]],
 ["T4",["z"]],

 ["S5",["y","T1","S1","S1"]],
 ["S5",["y","T1","T1","S1"]],
 ["S5",["x","S3"]],

 ["T5",["y","T1","S1","T1"]],
 ["T5",["y","T1","T1","T1"]],
 ["T5",["z"]],

 ["S6",["y","S2","Z","T1","Y","S2","S1"]],
 ["S6",["y","S2","X","Y","S2","S1"]],
 ["S6",["z","Y","S2","S1"]],
 ["S6",["y","S2","Z","T1","Y","S2","T1"]],
 ["S6",["y","S2","X","Y","S2","T1"]],
 ["S6",["z","Y","S2","T1"]],

 ["S6",["y","S2","Z","T1","X","S3"]],
 ["S6",["y","S2","X","X","S3"]],
 ["S6",["z","X","S3"]],
 ["S6",["y","S2","Z","T1","Z"]],
 ["S6",["y","S2","X","Z"]],
 ["S6",["z","Z"]],

 ],
"S1",rep
)
S=word_to_mat(["S1"],G4nnn)
T=word_to_mat(["S5"],G4nnn)
```

```
#G6n
    #equiv over nt alphabet
    rep={}
    rep["U1"]="U1"
    rep["U2"]="U1"
    rep["U3"]="U1"
    rep["U4"]="U1"
    rep["V1"]="V1"
    rep["V2"]="V1"
    rep["V3"]="V1"
    rep["V4"]="V1"
    rep["C1"]="C1"
    rep["C2"]="C1"
    rep["X"]="X"
    rep["Y"]="X"
    rep["Z"]="X"
    G6n=gram(
    ["U1","U2","U3","U4","V1","V2","V3","V4","C1","C2","X","Y","Z"],
    ["x","y","z"],
    [["X",["x"]],["Y",["y"]],["Z",["z"]],
    ["U1",["x","V1","C1","Y","U2","X","U1"]],
    ["U1",["x","V1","C1","Y","U2","X","U2"]],
    ["U1",["x","V1","C1","Y","U3","X","U1"]],
    ["U1",["x","V1","C1","Y","U3","X","U2"]],
    ["U1",["x","V1","C1","Z","X","U1"]],
    ["U1",["x","V1","C1","Z","X","U2"]],
    ["U1",["x","V2","Y","U2","X","U1"]],
    ["U1",["x","V2","Y","U2","X","U2"]],
    ["U1",["x","V2","Y","U3","X","U1"]],
    ["U1",["x","V2","Y","U3","X","U2"]],
    ["U1",["x","V2","Z","X","U1"]],
    ["U1",["x","V2","Z","X","U2"]],
    ["U1",["x","V3","Y","U2","X","U1"]],
    ["U1",["x","V3","Y","U2","X","U2"]],
    ["U1",["x","V3","Y","U3","X","U1"]],
    ["U1",["x","V3","Y","U3","X","U2"]],
    ["U1",["x","V3","Z","X","U1"]],
    ["U1",["x","V3","Z","X","U2"]],
    ["U1",["y","Y","U2","X","U1"]],
    ["U1",["y","Y","U2","X","U2"]],
    ["U1",["y","Y","U3","X","U1"]],
    ["U1",["y","Y","U3","X","U2"]],
    ["U1",["y","Z","X","U1"]],
    ["U1",["y","Z","X","U2"]],
    ["U2",["x","V1","C1","Y","U2","Y","U4","C1"]],
```

```
["U2",["x","V1","C1","Y","U2","Y","U2"]],
["U2",["x","V1","C1","Y","U3","Y","U4","C1"]],
["U2",["x","V1","C1","Y","U3","Y","U2"]],
["U2",["x","V1","C1","Z","Y","U4","C1"]],
["U2",["x","V1","C1","Z","Y","U2"]],
["U2",["x","V2","Y","U2","Y","U4","C1"]],
["U2",["x","V2","Y","U2","Y","U2"]],
["U2",["x","V2","Y","U3","Y","U4","C1"]],
["U2",["x","V2","Y","U3","Y","U2"]],
["U2",["x","V2","Z","Y","U4","C1"]],
["U2",["x","V2","Z","Y","U2"]],
["U2",["x","V3","Y","U2","Y","U4","C1"]],
["U2",["x","V3","Y","U2","Y","U2"]],
["U2",["x","V3","Y","U3","Y","U4","C1"]],
["U2",["x","V3","Y","U3","Y","U2"]],
["U2",["x","V3","Z","Y","U4","C1"]],
["U2",["x","V3","Z","Y","U2"]],
["U2",["y","Y","U2","Y","U4","C1"]],
["U2",["y","Y","U2","Y","U2"]],
["U2",["y","Y","U3","Y","U4","C1"]],
["U2",["y","Y","U3","Y","U2"]],
["U2",["y","Z","Y","U4","C1"]],
["U2",["y","Z","Y","U2"]],
["U2",["x","V1","C1","Y","U2","Z"]],
["U2",["x","V1","C1","Y","U3","Z"]],
["U2",["x","V1","C1","Z","Z"]],
["U2",["x","V2","Y","U2","Z"]],
["U2",["x","V2","Y","U3","Z"]],
["U2",["x","V2","Z","Z"]],
["U2",["x","V3","Y","U2","Z"]],
["U2",["x","V3","Y","U3","Z"]],
["U2",["x","V3","Z","Z"]],
["U2",["y","Y","U2","Z"]],
["U2",["y","Y","U3","Z"]],
["U2",["y","Z","Z"]],
["U3",["x","V1","C1","Y","U4","C1"]],
["U3",["x","V1","C1","Y","U4","C2"]],
["U3",["x","V2","Y","U4","C1"]],
["U3",["x","V2","Y","U4","C2"]],
["U3",["x","V3","Y","U4","C1"]],
["U3",["x","V3","Y","U4","C2"]],
["U3",["y","Y","U4","C1"]],
["U3",["y","Y","U4","C2"]],
["U4",["x","V1","C2"]],
["U4",["x","V4"]],
```

```
["V1",["x","U1","Y","V1","C2"]],
["V1",["x","U1","Y","V2"]],
["V1",["x","U1","Z"]],
["V1",["x","U3","Y","V1","C2"]],
["V1",["x","U3","Y","V2"]],
["V1",["x","U3","Z"]],
["V1",["x","U4","C1","Y","V1","C2"]],
["V1",["x","U4","C1","Y","V2"]],
["V1",["x","U4","C1","Z"]],
["V1",["y","Y","V1","C2"]],
["V1",["y","Y","V2"]],
["V1",["y","Z"]],
["V2",["x","U1","Y","V3"]],
["V2",["x","U1","Y","V4"]],
["V2",["x","U3","Y","V3"]],
["V2",["x","U3","Y","V4"]],
["V2",["x","U4","C1","Y","V3"]],
["V2",["x","U4","C1","Y","V4"]],
["V2",["y","Y","V3"]],
["V2",["y","Y","V4"]],
["V3",["x","U2","X","V1","C1"]],
["V3",["x","U2","X","V1","C2"]],
["V3",["x","U4","C2","X","V1","C1"]],
["V3",["x","U4","C2","X","V1","C2"]],
["V4",["x","U2","Y","V2"]],
["V4",["x","U2","Y","V3"]],
["V4",["x","U2","Z"]],
["V4",["x","U4","C2","Y","V2"]],
["V4",["x","U4","C2","Y","V3"]],
["V4",["x","U4","C2","Z"]],
["C1",["x","U1"]],
["C1",["x","U2"]],
["C2",["y","U4","C1"]],
["C2",["y","U2"]],
["C2",["z"]]],
"U1",rep
)
S=word_to_mat(["U1"],G6n)
T=word_to_mat(["V1","C1"],G6n)
```