Word Equations and SLPs: Assignement

Comment: I do not expect extremely formal and detailed solutions, in particular, if something was sketched on the lecture and you are to show a variant/generalization, then I expect you to write a similar sketch.

Grading: roughly 2 tasks for 3, 3.5 for 4, and 5 for 5. Could be lowered, depending on the actual outcome.

You can send your solution to aje@cs.uni.wroc.pl till the end of December.

Task 1 The $\exists^*$-theory of word equations consists of all sentences of the form:

$$\exists X_1, X_2, \dots, X_k \; \varphi(X_1, X_2, \dots, X_k)$$

where φ is quantifier-free formula that uses $\wedge, \vee, \neg$ as connectives and atomic formulas are word equations that use constants from Σ^* and variables $X_1, X_2, \dots, X_k$.

Show that we can decide this theory in PSPACE.

Hint: The algorithm will heavily employ non-determinism to reduce this case to a system of word equations and inequalities. The inequalities are easy to handle: look for first differences. You might need to introduce new constants.

Task 2

Given a word w in one phase we first list all letters in the word and then perform the block compression for listed letters, one by one. Then list all pairs in the current word and perform the pair compression for all listed pairs, one by one.

Show that the word is shortened by a constant fraction in each phase.

Hint: Consider any two consecutive letters in w and show that at least one of them is compressed in a phase.

Algorithm 1 Algorithm for Task 2

```
1: while $|w| > 1$ do
2:     $L \leftarrow$ list of letters in $w$
3:     for $a \in L$ do
4:         compress blocks of $a$
5:     $P \leftarrow$ list pairs in $w$
6:     for $ab \in P$ do
7:         replace all occurrences of $ab$ in $w$ by a fresh letter $c$
```

Task 3 Consider a variant of the algorithm for word equations similar to Task 2: In one phase it (nondterministically) lists all letters that appear in $S(U)$ (and puts them into L). Then one by one chooses a non-crossing letter from L and compresses its blocks (and removes it from L). Then for each remaining letter in L compresses its blocks.

Then it lists (nondeterministically) all pairs in $S'(U')$ and puts them into P. It compresses one by one each noncrossing pair from P. Finally, one by one the compresses remaining pairs and letters from P (which may be crossing).

Give a quadratic bound on the size of the kept equation (for appropriate non-deterministic choice). You may use Task 2 even if you cannot show it.

Hint: How many crossing pairs can be in P, when you first consider a crossing pair? Similalry for L and blocks?

Algorithm 2 Algorithm for Task 3

```
 1: while $|U| > 1$ or $|V| > 1$ do
 2:     $L \leftarrow$ list of letters (in the equation)
 3:     while there is $a \in L$ without crossing blocks do
 4:         compress $a$-blocks
 5:         remove $a$ from $L$
 6:     for $a \in L$ do
 7:         uncross $a$-prefixes and suffixes
 8:         compress $a$-blocks
 9:     $P \leftarrow$ list pairs in the equation
10:     while there is non-crossing $ab \in P$ do
11:         take some non-crossing $ab \in P$
12:         compress $ab$
13:         remove $ab$ from $P$
14:     for $ab \in P$ do
15:         uncross $ab$
16:         compress $ab$
17: Solve the problem naively
```

Task 4 Using the bound on the size of the minimal solutions of integer programming show a doubly exponential bound on the size of the length-minimal solution of a word equation.
Hint: How much can the length-minimal solution change (decrease) after one compression? Look at the length-minimal solution of the new equation and at the corresponding solution of the old equation.

Task 5 Improve the lower bound on the (α, β)-balanced grammar, so that it holds for binary alphabet (the bound might be slightly weaker).
Hint: It is enough to replace b_i with appropriate gadget that uses letters a, b.

Task 6 Show that construction from Task 16 (Exercise Sheet 3) can be used for the SLP balancing with a centroid path decomposition instead of symmetrical centroid path decomposition (in centroid path decomposition we assign a node v the number $\lfloor \log \rho(v) \rfloor$, where $\rho(v)$ is the number of paths from v to the leaves).

You can use the result from Task 16 (Exercise Sheet 3) even if you cannot solve it.