Jez Recompression Algorithm for Solving Word Equations:

from Theory to Implementation

Antonina Nepeivoda
Program Systems Institute of RAS

$\mathcal{STEP}$, *September 7th*

Plan

- Review of state-of-art on existential theory of strings;
- Basic heuristics used before and their possible extensions;
- Natural idea of recompression;
- Recompression basics;
- Simple heuristics on recompression.

Word equations

Definition

Given a constant alphabet $\mathfrak{A}$ and a variable set $\mathfrak{V}$, *a word equation* is an equation $\Phi = \Psi$, where $\Phi, \Psi \in \{\mathfrak{A} \cup \mathfrak{V}\}^*$. A solution to the word equation is a substitution $\sigma : \mathfrak{V} \mapsto \mathfrak{A}^*$ s.t. $\Phi\sigma$ textually coincides with $\Psi\sigma$.

Let E be $\mathrm{x}\ \mathbf{A}\ \mathbf{B} = \mathbf{B}\ \mathbf{A}\ \mathrm{x}$, where $\mathbf{A}, \mathbf{B} \in \mathfrak{A}$, $\mathrm{x} \in \mathfrak{V}$. Consider the sequence $\sigma_1 : \mathrm{x} \mapsto \mathbf{B}\mathrm{x}$, $\sigma_2 : \mathrm{x} \mapsto \varepsilon$. Then $\sigma_2 \circ \sigma_1 : \mathrm{x} \mapsto \mathbf{B}$ is a solution to E: $(\mathrm{x}\ \mathbf{A}\ \mathbf{B})\sigma_1\sigma_2 = \mathbf{B}\ \mathbf{A}\ \mathbf{B} = (\mathbf{B}\ \mathbf{A}\ \mathrm{x})\sigma_1\sigma_2$.

The history of the word equations

In theory:

- Algorithms for solving the quadratic (e.g. $\mathtt{x}\ \mathbf{A}\ \mathtt{y} = \mathtt{y}\ \mathbf{A}\ \mathtt{x}$) and one-variable word equations (Matiyasevich, 1965)
- An algorithm for solving the three-variable word equations (Hmelevskij, 1971)
- An algorithm for solving the word equations in the general case (Makanin, 1977) — triply exponential in the no solution case!
- More efficient (but still doubly exponential in the no solution case) algorithms (Plandowski, 2006, Jez, 2016)

The history of the word equations

In practice:

- efficient algorithms for solving the straight-line (e.g. $\mathtt{x}\ \mathtt{x}\ \mathtt{x} = \mathtt{y}\ \mathbf{A}\ \mathtt{z}$) word equations (Rümmer et al., 2014–...)
- algorithms for solving quadratic word equations together with constraints in LIA and finite transducers (Le et al., 2018, Lin et al., 2016–...)
- algorithms for solving the word equations in the case when the solution lengths are bounded (Bjørner, 2009–..., Day, 2019)
- general-case algorithms implemented in SMT-solvers using Levi's Lemma + heuristics.

Inconsistency in String Models

Simple random string models:

- 3–5 string parameters;
- 3–15 axioms;
- the second argument in predicate $\preceq$ is constant;
- no trivial inconsistencies.

Even in this simple case at least 20% of the random inconsistent models are not proved to be so by cvc5 and z3.

Hardness Results

$\mathcal{EST}$ — Existential String Theory.

Theory	letter counting	length counting	REGEX	Hardness
$\mathcal{EST}$	✗	✗	✓	PSPACE
$\mathcal{EST}$+len	✗	✓	✗	???
$\mathcal{EST}$+count	✓	✗	✗	Undec.

Note: letter and length counting can be used as additional datum in the pure existential string theory.

Adding Counting Heuristics

- Length counting ($\approx$25% successes).
- Letter counting ($\approx$50% successes).

Why are they working well? How can they be extended?

Base General Heuristics: Levi's Lemma

Levi's Lemma

Given equation $x\,\Phi_1 = y\,\Phi_2$, the following condition holds for all its solutions $x = y\,x' \vee y = x\,y'$.

Important case: if the equation is $x\,\Phi_1 = \xi\,\Phi_2$ ($\xi \in \Sigma$), then either $x = \xi\,x' \vee x = \varepsilon$.

- Asymmertic;
- Explodes variables multiplicity;
- Explodes regular restrictions (later).

Equation Classes Solvable by LL

(also «terminating wrt LL»)

- Quadratic equations (NP-hard).
- Straight-line equations (linear with heuristics).
- One-variable equations (linear with heuristics; require splitting).
- Equation systems containing an equation of classes 1–3 and an arbitrary set of equations $x_i\,\Phi_i = \Psi_i\,x_i$, where Φ_i, Ψ_i are constant strings (require splitting).

Splitting Heuristics

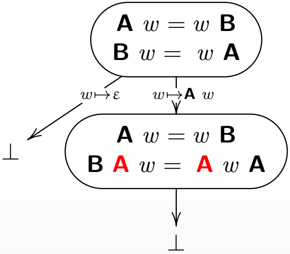
LL directly applied to the equation $\mathbf{A}\ w\ \mathbf{B}\ w = w\ \mathbf{B}\ w\ \mathbf{A}$ results in an infinite tree.

$$
\begin{tikzcd}
& \mathbf{A}\ w\ \mathbf{B}\ w = w\ \mathbf{B}\ w\ \mathbf{A} \arrow[dl, "w \mapsto \varepsilon"'] \arrow[d, "w \mapsto \mathbf{A}\ w"] \\
\perp & \mathbf{A}\ w\ \mathbf{B}\ {\color{red}\mathbf{A}}\ w = w\ \mathbf{B}\ {\color{red}\mathbf{A}}\ w\ \mathbf{A} \arrow[dl, "w \mapsto \varepsilon"'] \arrow[d, "w \mapsto \mathbf{A}\ w"] \\
\perp & \mathbf{A}\ w\ \mathbf{B}\ {\color{red}\mathbf{A}^2}\ w = w\ \mathbf{B}\ {\color{red}\mathbf{A}^2}\ w\ \mathbf{A} \arrow[dl, "w \mapsto \varepsilon"'] \arrow[d, "w \mapsto \mathbf{A}\ w"] \\
\perp & \cdots
\end{tikzcd}
$$

Still, we can split the equation wrt the length-equal prefixes (or suffixes): $\mathbf{A}\ w\ \ \mathbf{B}\ w = w\ \mathbf{B}\ \ w\ \mathbf{A}$

Splitting Heuristics

LL directly applied to the equation $\mathbf{A}\ w\ \mathbf{B}\ w = w\ \mathbf{B}\ w\ \mathbf{A}$ results in an infinite tree.
Still, we can split the equation wrt the length-equal prefixes (or suffixes): $\boxed{\mathbf{A}\ w}\ \boxed{\mathbf{B}\ w} = \boxed{w\ \mathbf{B}}\ \boxed{w\ \mathbf{A}}$
The resulting equation system is trivially solvable.



Safety of Splitting wrt LL

Transition from $\Phi_1\Phi_2 = \Psi_1\Psi_2$ to the system

$$\begin{cases} \Phi_1 = \Psi_1 \\ \Phi_2 = \Psi_2 \end{cases} \tag{1}$$

cannot transform a terminating equation into a non-terminating system.

If unfolding wrt LL of $\Phi_1\Phi_2 = \Psi_1\Psi_2$ terminates, then unfolding of the system (1) also terminates.

Counting Heuristics

- Equation $w\,u\,u = u\,\mathbf{A}\,u\,w\,v$ cannot be splitted. However, its image in LIA gives an equation with no solution:

$$|w| + 2 \cdot |u| < |w| + 2 \cdot |u| + |v| + 1$$

- Equation $u\,u\,\mathbf{A}\,w = w\,u\,\mathbf{B}\,u$ is not contradictory in LIA, but counting of **B**'s leads to a trivial contradiction.

$$|w|_{\mathbf{A}} + 2 \cdot |u|_{\mathbf{A}} + 1 > |w|_{\mathbf{A}} + 2 \cdot |u|_{\mathbf{A}}$$

- A similar heuristics is successfully applied to $\mathbf{ABC}w\mathbf{C}u\mathbf{C}u = w\mathbf{C}u\mathbf{C}u\mathbf{CBA}$, if the *subwords* **AB** are counted:

$$|w|_{\mathbf{AB}} + 2 \cdot |u|_{\mathbf{AB}} + 1 > |w|_{\mathbf{AB}} + 2 \cdot |u|_{\mathbf{AB}}$$

This equation holds, since all the **AB** occurrences are either inside values of w and u, or explicitly appear in the equation sides, separated with **C** from variables.

Counting Heuristics

Let us consider **C**: $\mathbf{AB}wuu = wuu\mathbf{BA}$. Mapping $x \mapsto |x|_{\mathbf{AB}}$, $\mathbf{AB} \mapsto 1$, $\mathbf{BA} \mapsto 0$ leads to a contradiction. However, the equation has solutions, ie $w \mapsto \mathbf{A}$, $u \mapsto \varepsilon$.
Let us consider the subwords **AB** after this substitution:

$$\mathbf{AB}\ \mathbf{A} = \mathbf{A}\ \mathbf{BA}$$

The straightforward subword counting of **AB** does not takes the *crossing pair* into account.

Counting Heuristics

Let us say that the pair $\gamma_1\gamma_2$ ($\gamma_1 \neq \gamma_2$) can have a crossing occurrence in a solution of $\Phi_1 = \Phi_2$, if:

- Φ_1 or Φ_2 contains two neighborous variables;
- Φ_i contains a variable occurrence left to γ_2 occurrence;
- Φ_i contains a variable occurrence right to γ_1 occurrence.

If $\gamma_1 = \gamma_2$ is non-trivial: we must forbid overlapping of $\gamma_1\gamma_1$ with itself when counting.

Natural Idea of Recompression

- If the pair $\gamma_1\gamma_2$ ($\gamma_1 \neq \gamma_2$) has no crossing occurrences in any solution, then the pair can be considered as a new «letter», forcing the minimal solution of the resulting equation to shorten.

Given $w\mathbf{CAB} = \mathbf{B}w\mathbf{C}w$, pair $\mathbf{AB}$ is non-crossing, so we can replace it with a new constant $\mathbf{A}_1$. The resulting equation is $w\mathbf{CA}_1 = \mathbf{B}w\mathbf{C}w$.

- *Maximal* non-trivial blocks of the same letter γ can be also treated as letters.

Given $\mathbf{A}^2\mathbf{B}w = w\mathbf{BAB}w\mathbf{BA}^3$, letter $\mathbf{A}$ is organised in subwords $\mathbf{A}^1$, $\mathbf{A}^2$, $\mathbf{A}^3$, splitted by $\mathbf{B}$. We can compress $\mathbf{A}^i$ into $\mathbf{A}_i$ and receive the equation $\mathbf{A}_2\mathbf{B}w = w\mathbf{BA}_1\mathbf{B}w\mathbf{BA}_3$.

Crossing pairs

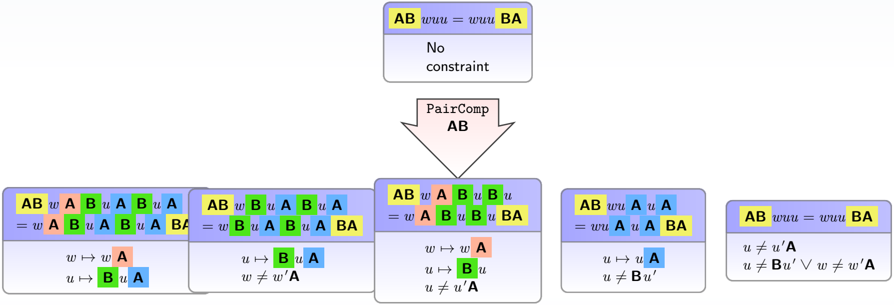
How to get rid of the crossing pairs $\gamma_1\gamma_2$ in $\Phi_1 = \Phi_2$? Let us make all of them explicit.

- $\Phi_1 = \Phi_2$ contains substring $w_i\gamma_2 \Rightarrow w_i = w_i'\gamma_1$;
- $\Phi_1 = \Phi_2$ contains substring $\gamma_1 w_i \Rightarrow w_i = \gamma_2 w_i'$;
- $\Phi_1 = \Phi_2$ contains substring $w_i w_j \Rightarrow w_i = w_i'\gamma_1$ *and* $w_j = \gamma_2 w_j'$.

We can consider all the given substitutions together with the constraints on w_i values forbidding them, and construct all crossing pairs options.

Crossing pairs

Consider the equation $\mathbf{AB}wuu = wuu\mathbf{BA}$. The obvious cases of crossing pairs occurrences are given below.

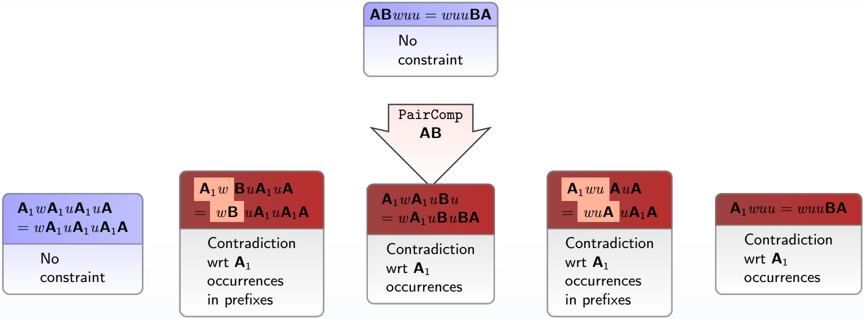


Negative constraints were simplified in the nodes.

A B	substitution $u \mapsto u\,\mathbf{A}$
A B	substitutions $u \mapsto \mathbf{B}\,u$ and $w \mapsto w\,\mathbf{A}$
A B	substitutions $u \mapsto \mathbf{B}\,u$ and $u \mapsto u\,\mathbf{A}$

Crossing pairs

Now we can compress $\mathbf{AB} \mapsto \mathbf{A}_1$, since every equation considered contains no crossing $\mathbf{AB}$. Almost all resulting equations are contradictory.



Compression cannot lose solutions, thus contradictory branches can be pruned.

Empty Substitutions

If the composition $\eta_n \circ \cdots \circ \eta_1$ where $\eta_i : w_i \mapsto \varepsilon$ creates a new crossing pair, then application of any η_i creates a crossing pair.

The non-empty substitutions do not satisfy this property.

Block Compression

- Compression to single $X \mapsto \alpha^i$;
- Taking block prefixes $X \mapsto \alpha^{i_1} X \alpha^{i_2}$, with restriction $X \neq \varepsilon$, $X \neq \alpha X$, $X \neq X\alpha$.

The new letters are not equal by default, but the indexes can be substituted to get equal letters.

Block Compression

$A_0B_0A_0B_0\mathbf{ww} = \mathbf{ww}B_0B_0A_0A_0$

No constraints
No conditions

$\xrightarrow{\texttt{BlockComp}\ A_0}$

$A_0B_0A_0B_0A_1\mathbf{w}B_1\mathbf{w}C_1 = A_1\mathbf{w}B_1\mathbf{w}C_1B_0B_0D_1$

$\mathbf{w} \neq \varepsilon$
$\mathbf{w} \neq \mathbf{w_P}A_0$
$\mathbf{w} \neq A_0\mathbf{w_S}$
$A_1 := A_0^{i_1}$ $B_1 := A_0^{i_1+i_2}$ $C_1 := A_0^{i_2}$
$D_1 := A_0^2$

$A_0B_0A_0B_0A_1 = A_1B_0B_0B_1$

No constraints
$A_1 := A_0^{2\cdot i_1}$
$B_1 := A_0^2$

$$\begin{cases} \text{Prefixes} : A_1 = A_0 \\ \text{Suffixes} : A_1 = B_1 \end{cases}$$

Thus, $A_0 = A_0^2$, which is contradictory.

$$\begin{cases} \text{Prefixes} : A_1 = A_0 \\ \text{Suffixes} : C_1 = D_1 \end{cases} \Rightarrow \begin{cases} i_1 = 1 \\ i_2 = 2 \end{cases}$$

Thus, $A_0 \neq B_1 \neq C_1 \neq D_1$ is verified, and we can count letters A_0 as usual.

Linearity & Counting

- Indexes of suffix/prefix letters are linear functions;
- Thus no other from linear diophantine equation may appear in a configuration.

(But non-linearity can appear with counting heuristics)

Case explosion & Heuristics

(tested on balanced equations, i.e. having same sets of variables in left- and right-hand sides)

- Straightforward case study $\Rightarrow O(4^n)$ options (n — distinct variables);
- Constraint simplification $\Rightarrow O(2^m)$ options (m — neighboring occurrences);
- Letter counting $\Rightarrow$ dramatic decrease of cases;
- Levi's Lemma Heuristics + negative constraints $\Rightarrow$ realistic options sets for balanced equations.

Looping Heuristics

- No new variable introduction $\Rightarrow$ possible loops w.r.t. letter renaming.
- Must consider negative restrictions:
    - remove insignificant dependencies;
    - substitute all the given dependencies & find an inclusion (similar to MSCP-A constraint treating).

Regular Restrictions

Consider $\mathbf{z}\,\mathbf{y}\,\mathbf{x} = \mathbf{x}\,\mathbf{x}\,\mathbf{z}$, where $\mathbf{x} \in \mathbf{A}^*$, $\mathbf{y} \in \mathbf{A}^+\mathbf{B}^+$, $\mathbf{z} \in \mathbf{B}^*$.

- LL: Case study is problematic, since the substitutions are of the form $x_i \mapsto x_j x_i$. In general: regex intersection explodes regex size.
- Jez: To prove the equation is inconsistent, it is enough to compress **A** blocks. In general, constant compression simplifies regex structure (but explodes the alphabet size).