Program Semantics, Specification, and Verification

Innopolis, October 19th–20th, 2024

A Report on Implementation of Jez's Algorithm on Solving Word Equations

Ilya Afanasyev
ilya.afanasyev@affeeal.ru

Antonina Nepeivoda
a_nevod@mail.ru

Word Equations in Computer Science

Everywhere in this talk “string” = “word”.

- Σ is the alphabet, $a, b, c, \ldots$ are letters in Σ;
- X, Y, Z are variables ranging over Σ^*;
- g_i, h_i are word parameters.
- ε is the empty word.

Definition

Given a variable set $\mathfrak{V}$, *a word equation* is an equation $\mathcal{U} \doteq \mathcal{V}$, where $\mathcal{U}, \mathcal{V} \in \{\Sigma \cup \mathfrak{V}\}^*$.

A solution to the word equation is a morphism $\sigma : (\mathfrak{V} \cup \Sigma) \to \Sigma^*$ being identity on Σ s.t. $\mathcal{U}\sigma$ textually coincides with $\mathcal{V}\sigma$.

Short remark: In case of finding *minimal* solutions, the letter alphabet Σ can be narrowed to the explicit letter alphabet of the equation, e.g. the alphabet of $\mathcal{U}\mathcal{V}$.

Word Equations Problem: Conceptual Approach

- Was stated by A. Markov (1960s).
- Algorithms for solving the quadratic 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).
- Simpler (but doubly exponential in the no solution case) algorithms (Plandowski, 2006, Jez, 2014)

Non-Determinism and State Space

Plandowski, 2018:

...the only implementable algorithm for the satisfiability problem of word equations is the Makanin algorithm, though its theoretical complexity is much worse the ones for other algorithms.
The reason is that the other algorithms are very nondeterministic.

Besides the theoretical concepts, the algorithms require careful case studying, in order to search for successful sequences of guesses.

Recompression: Change Generators Preserving Solvability

Given $\Sigma = \{a, b\}$, consider both equation $XabY \doteq YabX$ and its solution $\langle X \mapsto abab, Y \mapsto ab \rangle$.

$$\overbrace{a\,b\,a\,b}^{X}\ a\,b\ \underbrace{a\,b}_{Y} \doteq \underbrace{a\,b}_{Y}\ a\,b\ \overbrace{a\,b\,a\,b}^{X}$$

Compress: $ab \mapsto c$, now $\Sigma = \{c\}$ (no explicit occurrences of a, b), and solution is $\langle X \mapsto cc, Y \mapsto c \rangle$.

$$\overbrace{c\,c}^{X}\ c\ \underbrace{c}_{Y} \doteq \underbrace{c}_{Y}\ c\ \overbrace{c\,c}^{X}$$

The (successful) compression step makes the solutions lengths shorter with no impact on the equation solvability.

Recompression: Change Generators Preserving Solvability

The (successful) compression step makes the solutions lengths shorter with no impact on the equation solvability.

If all the guesses are assumed to be successful, then any minimal solution requires at most $\Phi(d)$ compressions, where $\Phi(d)$ is a theoretical bound on the minimal solution length.

Problem: in case of an unsuccessful guess, the solution length does not change.

Crossing Pairs and Pair Compression

Given $\mathcal{U} \doteq \mathcal{V}$ and a solution σ, one must replace all occurrences of a pair $ab \in \Sigma^2$ $(a \neq b)$ with a new letter $c \in \Sigma$. Occurrence of ab in a solution word is

- explicit if it comes from $\mathcal{U}$ or $\mathcal{V}$;
- implicit if it comes solely from $X\sigma$, $X \in \mathfrak{V}$;
- crossing otherwise.

Given $abXYbb \doteq YbbabX$, $X \mapsto abb$, $Y \mapsto aba$. Occurrences of ab:

$$ab\ \underbrace{abb}_{X}\ \underbrace{aba}_{Y}\ bb \doteq \underbrace{aba}_{Y}\ bbab\ \underbrace{abb}_{X}$$

Replacing ab with c directly won't affect crossing occurrences:

$$cXYbb \doteq YbbcX,\ X \mapsto cb,\ Y \mapsto ca$$

Crossing Pairs and Pair Compression

All crossing occurrences must be made explicit.

Given $\mathcal{U} \doteq \mathcal{V}$, a solution σ and a pair ab, let $\mathcal{U}$ or $\mathcal{V}$ contain

- Xb, and $X\sigma$ ends with $a \Rightarrow X \mapsto Xa$ creates ab explicitly;
- aY, and $Y\sigma$ starts with $b \Rightarrow Y \mapsto bY$ creates ab explicitly;
- XY, and $X\sigma$ ends with a, $Y\sigma$ starts with $b \Rightarrow \big(X \mapsto Xa \textit{ and } Y \mapsto bY\big)$ create ab explicitly.

Crossing occurrences become explicit with substitution $Y \mapsto Ya$:

$$abXYbb \doteq YbbabX, \qquad X \mapsto abb,\ Y \mapsto aba$$

$$abXYabb \doteq YabbabX, \qquad X \mapsto abb,\ Y \mapsto ab$$

$$cXYcb \doteq YcbcX, \qquad X \mapsto cb,\ Y \mapsto c$$

Block Compression

Given $\mathcal{U} \doteq \mathcal{V}$ and a letter a, one must replace all maximal a^l blocks with a new letter a_l, $l > 1$. Since there can be crossing blocks, one needs to pop whole a-prefix and a-suffix out of the equation's variables to uncross the occurrences (via $X \mapsto a^{l_1} X a^{l_2},\ l_1, l_2 \geq 0$).

Given $aabbbYX \doteq YbaXbba$, a solution $X \mapsto abba$, $Y \mapsto aabb$. Occurrences of b-blocks:

$$aabbb\underbrace{aabb}_{Y}\underbrace{abba} _{X}\doteq \underbrace{aabb}_{Y}ba\underbrace{abba}_{X}bba$$

Applying $Y \mapsto Yb^2$ and introducing $b_2 = b^2$, $b_3 = b^3$ ($b_2, b_3 \in \Sigma$):

$$aab^3Yb^2X \doteq Yb^2baXb^2a, \qquad X \mapsto ab^2a,\ Y \mapsto aa$$

$$aab_3Yb_2X \doteq Yb_3aXb_2a, \qquad X \mapsto ab_2a,\ Y \mapsto aa$$

An Outline of the Recompression Algorithm

```
while $\mathcal{U}(\forall X_i \mapsto \varepsilon) \neq \mathcal{V}(\forall X_i \mapsto \varepsilon)$ do
    $L \leftarrow$ letters from $\mathcal{U} \doteq \mathcal{V}$
    choose a pair of letters or a block from $L$
    if it is crossing then
        Uncross it
    Compress it
```

Recompression For Straight-Line Pattern Matching (Jez, 2011)

Straight-line programs: context-free grammars without recursion and non-determinism.

$$\begin{array}{rcl} S & \longrightarrow & A_1 A_2 \\ A_1 & \longrightarrow & B_1 B_2 \\ & \dots & \\ B_n & \longrightarrow & c_n \end{array}$$

Given two SLPs or a word and an SLP, the recompression technique:

- knows in advance all pairs and blocks occurring in the word;
- or can compute `First`-set by well-known recursive algorithm (and `Last`-set by its inverse) $\Rightarrow$ all the crossing pairs are known.

Recompression for One-Variable Word Equations (Jez, 2013)

Any one-variable word equation $\mathcal{U} \doteq \mathcal{V}$ either is trivial or can be normalised to

$$h_0 X h_1 X \dots h_k X \doteq X g_1 X \dots g_k X g_{k+1}$$

where $h_0 \neq \varepsilon$, $g_{k+1} \neq \varepsilon$.

- The first and last letters of any solution are known given h_0 and g_{k+1};
- The solutions defined by single blocks can be tested explicitly by substitution $X \mapsto a^l$.

Like in the SLP case, non-determinism is completely avoided. However, in most cases recompression generates a graph, not a mere sequence of guesses.

Possible Heuristics

Small-Step

- Choose relevant pair compressing cases;
- Resolve constraints;
- Prune immediate contradictions.

Big-Step

- Choose an optimal compression path;
- Propagate constraints;
- Prune excessive (repetitive) branches.

In this talk, we focus only on the small-step heuristics.

Negative Constraints in Compression

Natural Refinement of Recompression

Cases considered are strictly disjoint $\Leftrightarrow$ we enrich tree states with negative constraints:

- tracking consistency of compression paths;
- being able to propagate the constraints in order to reduce state space;
- being able to use the constraints in other heuristics.

inconsistent trace

$$\begin{tikzcd}
\boxed{XbY \doteq ZXcZ} \arrow[dr, "ab \mapsto d_0"] & \\
& \boxed{XbY \doteq ZXcZ} \arrow[dl, "ac \mapsto d_1"] \\
\boxed{XabY \doteq ZXd_1Z} &
\end{tikzcd}$$

Pair Compression: Empty Substitutions

An *essential* substitution creates new occurrences of the pair in the equation.

Proposition

If the composition $\eta_n \circ \cdots \circ \eta_i$ where $\eta_i : X_i \mapsto \varepsilon$ is essential, then any η_i is essential.

Given a pair ab to compress, let a part of the equation be presented as one of

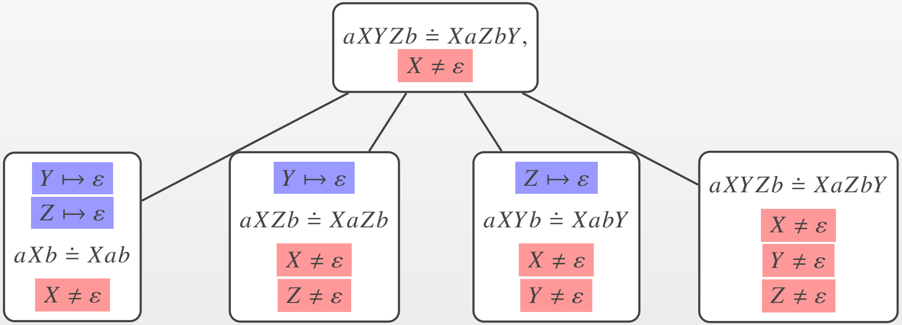
- $U_1 a V b U_2$, where $U_1, U_2 \in (\Sigma \cup \mathfrak{V})^*$, $V \in \mathfrak{V}^+$;
- $U_1 X V b U_2$, where $V \in (\mathfrak{V} \backslash X)^+$;
- $U_1 a V Y U_2$, where $V \in (\mathfrak{V} \backslash Y)^+$;
- $U_1 X V Y U_2$, where $V \in (\mathfrak{V} \backslash X \backslash Y)^+$.

Then for every $X \in V$ s.t. there is no constraint $X \neq \varepsilon$, substitution $X \mapsto \varepsilon$ is essential.

Pair Compression: Empty Substitutions

There are two options for an essential substitution $\eta : X \mapsto \varepsilon$:

1. η holds, negative constraints on X are removed.
2. η does not hold, the negative constraint $X \neq \varepsilon$ is added.



Pair Compression: Non-empty Substitutions

Given $\mathcal{U} \doteq \mathcal{V}$ and a pair ab to compress, let $\mathcal{U}$ or $\mathcal{V}$ contain

- $Xb \Rightarrow X \mapsto Xa$ can create ab explicitly;
- $aY \Rightarrow Y \mapsto bY$ can create ab explicitly;
- $XY \Rightarrow X \mapsto Xa \wedge Y \mapsto bY$ can create ab explicitly.

It is assumed there are no contradictory negative constraints.

There are two options for an essential elementary substitution $\sigma : X \mapsto Xa$:

- σ holds, redundant negative constraints on X are removed;
- σ does not hold, the negative constraint $\neg\,\mathrm{ends}(a, X)$ is added.

Pair Compression: Non-empty Substitutions

There are three option sets for the essential substitution composition $\sigma_2 \circ \sigma_1$, where $\sigma_1 : X \mapsto Xa$, $\sigma_2 : Y \mapsto bY$:

- both σ_1 and σ_2 are essential:
    - $X \mapsto Xa \wedge Y \mapsto bY$;
    - $X \mapsto Xa \wedge \neg\,\mathrm{starts}(b, Y)$;
    - $\neg\,\mathrm{ends}(a, X) \wedge Y \mapsto bY$;
    - $\neg\,\mathrm{ends}(a, X) \wedge \neg\,\mathrm{starts}(b, Y)$.
- σ_1 is essential, σ_2 is not essential:
    - $X \mapsto Xa \wedge Y \mapsto bY$;
    - $X \mapsto Xa \wedge \neg\,\mathrm{starts}(b, Y)$;
    - $\neg\,\mathrm{ends}(a, X)$.
- neither σ_1 nor σ_2 is essential:
    - $X \mapsto Xa \wedge Y \mapsto bY$;
    - $\neg\,\mathrm{ends}(a, X) \vee \neg\,\mathrm{starts}(b, Y)$.

Pair Compression: Non-empty Substitutions

$$\begin{tikzcd}[column sep=small]
 & \begin{array}{c} bXYaa \doteq YabaX, \\ \text{compressing } ab \mapsto c \end{array} \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] \arrow[drr, dash] & & \\
\begin{array}{c} X \mapsto Xa \land Y \mapsto bY \\ X \mapsto bX \\ bbXabYaa \doteq bYababXa \\ bbXcYaa \doteq bYccXa \end{array} & \begin{array}{c} X \mapsto Xa \land Y \mapsto bY \\ bXabYaa \doteq bYabaXa \\ bXcYaa \doteq bYcaXa \\ \neg\, \mathrm{starts}(b, X) \end{array} & \begin{array}{c} X \mapsto bX \\ bbXYaa \doteq YababX \\ bbXYaa \doteq YccX \\ \neg\, \mathrm{ends}(a, X) \\ \lor \neg\, \mathrm{starts}(b, Y) \end{array} & \begin{array}{c} bXYaa \doteq YabaX \\ bXYaa \doteq YcaX \\ \neg\, \mathrm{ends}(a, X) \\ \lor \neg\, \mathrm{starts}(b, Y) \\ \neg\, \mathrm{starts}(b, X) \end{array}
\end{tikzcd}$$

Block Compression

Given a letter a to compress, there are two options for a variable X of the equation:

- single block compression: $X \mapsto a^i$;
- prefix and suffix a-blocks extraction: $X \mapsto a^{i_1} X a^{i_2}$; negative constraints $X \neq \varepsilon$, $\neg\,\mathrm{starts}(a, X)$, $\neg\,\mathrm{ends}(a, X)$ are added.

Processing negative constraints of the equation:

- if $\neg\,\mathrm{starts}(a_0, X),\ a_0 \in \mathtt{First}(a)$, then
  - $X \mapsto a^i \Rightarrow X \mapsto \varepsilon$;
  - $X \mapsto a^{i_1} X a^{i_2} \Rightarrow X \mapsto X a^{i_2}$, the constraint $\neg\,\mathrm{starts}(a_0, X)$ remains.
- if $\neg\,\mathrm{starts}(b, X),\ b \notin \mathtt{First}(a)$, then
  - $X \mapsto a^i \Rightarrow$ the constraint $\neg\,\mathrm{starts}(b, X)$ is removed;
  - $X \mapsto a^{i_1} X a^{i_2} \Rightarrow \begin{cases} X \mapsto X a^{i_2} & \text{the constraint } \neg\,\mathrm{starts}(b, X) \text{ is saved} \\ X \mapsto a^{i_1} X a^{i_2} & \text{the constraint } \neg\,\mathrm{starts}(b, X) \text{ is removed } (i_1 \neq 0) \end{cases}.$
- etc.

$\texttt{First}$ and $\texttt{Last}$ Sets

Let $a, b \in \Sigma$.

$\texttt{First}$ Set

$b \in \texttt{First}(a)$ if there is a chain

$$\begin{aligned} a &= a_1^{i_1} w_1, \\ a_1 &= a_2^{i_2} w_2, \\ a_2 &= a_3^{i_3} w_3, \\ &\ldots, \\ a_n &= b^{i_n} w_n, \end{aligned}$$

$\texttt{Last}$ Set

$b \in \texttt{Last}(a)$ if there is a chain

$$\begin{aligned} a &= w_1 a_1^{i_1}, \\ a_1 &= w_2 a_2^{i_2}, \\ a_2 &= w_3 a_3^{i_3}, \\ &\ldots, \\ a_n &= w_n b^{i_n}, \end{aligned}$$

where $a_j \in \Sigma,\ w_j \in \Sigma^*,\ i_j \geq 1$ for $j = 1, \ldots, n$.

Let an equation have a constraint $\neg\,\text{starts}(a, X)$, and $d = bc$ and $b = a^i$ $(i > 0)$. We can't perform i.e. a substitution $X \mapsto dX$ since $a \in \texttt{First}(d)$.

What is Used in Practice?

- efficient algorithms for solving straight-line or linear word equations, generalized to chain-free word equations in 2023 (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 (e.g. in cvc5).

Levi's-Lemma-Based Heuristics

Variables at both sides

$$\begin{tikzcd}
 & \boxed{X\mathcal{U} \doteq Y\mathcal{V}} \arrow[dl] \arrow[dr] & \\
X \mapsto YX & & Y \mapsto XY
\end{tikzcd}$$

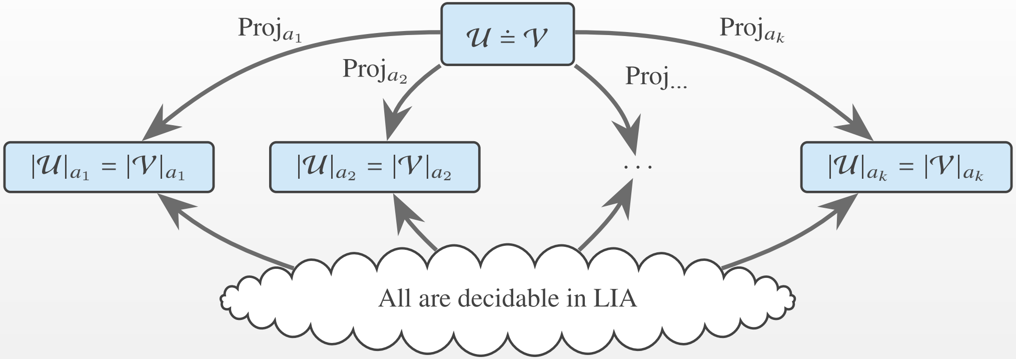
Variable at one side

$$\begin{tikzcd}
 & \boxed{X\mathcal{U} \doteq a\mathcal{V}} \arrow[dl] \arrow[dr] & \\
X \mapsto aX & & X \mapsto \varepsilon
\end{tikzcd}$$

- Can blow up number of variable occurrences when used directly;
- Safe to be used as a heuristic.

```
**while** $(E \text{ is } X_i\mathcal{U} \doteq a\mathcal{V} \text{ and } \neg\,\text{starts}(a, X_i))$ **do**
    **if** $(X_i \neq \varepsilon)$
        **return** $\perp$
    **else** $E := E(X_i \mapsto \varepsilon)$
```

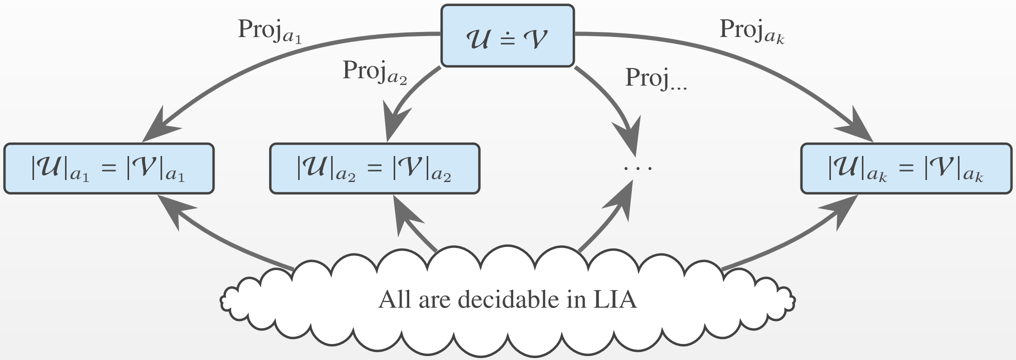
Solution Projection Working With Recompression



$\mathcal{U} \doteq \mathcal{V}$
Proj_{a_1}
Proj_{a_2}
$\mathrm{Proj}_{\dots}$
Proj_{a_k}
$|\mathcal{U}|_{a_1} = |\mathcal{V}|_{a_1}$
$|\mathcal{U}|_{a_2} = |\mathcal{V}|_{a_2}$
$\dots$
$|\mathcal{U}|_{a_k} = |\mathcal{V}|_{a_k}$
All are decidable in LIA

Given equation $XaY \doteq YbX$, it implies $|XaY|_a \neq |YbX|_a$, which is a contradiction.

Solution Projection Working With Recompression


$\mathcal{U} \doteq \mathcal{V}$
Proj_{a_1}
Proj_{a_2}
$\mathrm{Proj}_{\ldots}$
Proj_{a_k}
$|\mathcal{U}|_{a_1} = |\mathcal{V}|_{a_1}$
$|\mathcal{U}|_{a_2} = |\mathcal{V}|_{a_2}$
$\ldots$
$|\mathcal{U}|_{a_k} = |\mathcal{V}|_{a_k}$
All are decidable in LIA

Recompression changes generator set $\Rightarrow$ new projection-based checks become possible.

Splitting Heuristics

$$\boxed{\begin{array}{l}\mathcal{U}_1\mathcal{U}_2 \doteq \mathcal{V}_1\mathcal{V}_2,\ \text{where} \\ \quad \forall X_i\big(|\mathcal{U}_1|_{X_i} = |\mathcal{V}_1|_{X_i}\big) \text{ and } |\mathcal{U}_1| = |\mathcal{V}_1|\end{array} \Leftrightarrow \begin{cases}\mathcal{U}_1 \doteq \mathcal{V}_1 \\ \mathcal{U}_2 \doteq \mathcal{V}_2\end{cases}}$$

- Matches well with Levi's-Lemma-based method and basic projection heuristics;
- Still, the latter can lose contradictions after splitting, when used with recompression:

Given $XabcYZ \doteq aXZcbY$, where $\big(\neg\,\mathrm{ends}(a, X) \vee \neg\,\mathrm{starts}(b, Z)\big) \wedge (X \neq \varepsilon)$, the compression $ab \mapsto d_0$ results in $Xd_0cYZ \doteq aXZcbY$.
The projection $|Xd_0cYZ|_a = |aXZcbY|_a$ is contradictory, while the projections of the split equations $\begin{cases}Xa \doteq aX \\ bcYZ \doteq ZcbY\end{cases}$ are not.

Splitting Heuristics

$$\begin{array}{l} \mathcal{U}_1\mathcal{U}_2 \doteq \mathcal{V}_1\mathcal{V}_2, \text{ where} \\ \quad \forall X_i \big(|\mathcal{U}_1|_{X_i} = |\mathcal{V}_1|_{X_i}\big) \text{ and } |\mathcal{U}_1| = |\mathcal{V}_1| \end{array} \quad \Leftrightarrow \quad \begin{cases} \mathcal{U}_1 \doteq \mathcal{V}_1 \\ \mathcal{U}_2 \doteq \mathcal{V}_2 \end{cases}$$

- Matches well with Levi's-Lemma-based method and basic projection heuristics;
- Still, the latter can lose contradictions after splitting, when used with recompression:

Proposition

Splitting heuristics in composition with Levi's-Lemma heuristics is consistent with recompression-refined projection heuristics.

Experimental Results

- NC$\geq k$ — percent of the tests where number of non-contradictory cases after compression exceeds or equal to k.

	$< 3^N$	NC≥100	NC≥10	NC≥5	NC≥3
Asymmetric	98%	12%	61%	81%	90%
Symmetric	75%	1%	26%	46%	60%
Asymmetric 2-Var	100%	0	0	5%	52%
Symmetric 2-Var	100%	0	0	0	16%

- All case sets (including contradictory) are of cardinality less than 4^N, where N is cardinality of the variable set (*not* the number of their occurrences!)
- *Small bonus:* one-variable equations are always compressed deterministically with constraints and heuristics.

Conclusion

- Small-step heuristics together with constraints reduce the search tree in Jez's algorithm drastically.
  - An interesting case study: two-variable balanced equations.
- Recompression-based projection heuristic (and its compositions with other heuristics) can be useful in practical string solvers.

Thanks for attention!