On Solving String Equations via

Powers and Parikh Images

Clemens Eisenhofer
Theodor Seiser
Laura Kovács

TU Wien, Austria

Nikolaj Bjørner

MS Research, USA

28^{th} of September, 2025

Super Compact Summary

Super Compact Summary

We have a new String Solver!

What this Talk is About

- Background
  - What is string equation solving?
  - Nielsen transformation

What this Talk is About

- Background
  - What is string equation solving?
  - Nielsen transformation
- New string equation solving calculus using interplay of
  - ground power introduction
  - generalised Parikh images
    - multi-sequence (MS) Parikh
    - multi-sequence error bounded (MSEB) Parikh
  - lazy equality decomposition
  - optimised Nielsen rules
  - integer reasoning

What this Talk is About

- Background
  - What is string equation solving?
  - Nielsen transformation
- New string equation solving calculus using interplay of
  - ground power introduction
  - generalised Parikh images
    - multi-sequence (MS) Parikh
    - multi-sequence error bounded (MSEB) Parikh
  - lazy equality decomposition
  - optimised Nielsen rules
  - integer reasoning

What is String Equation Solving?

What is String Equation Solving?

For mathematicians

Solving (existentially quantified word) equations over the free monoid $\langle \Sigma, +, \varepsilon \rangle$

What is String Equation Solving?

For mathematicians

Solving (existentially quantified word) equations over the free monoid $\langle \Sigma, +, \varepsilon \rangle$

For computer scientists

Finding values for string variables s.t. equations evaluate true

What is String Equation Solving?

For mathematicians

Solving (existentially quantified word) equations over the free monoid $\langle \Sigma, +, \varepsilon \rangle$

For computer scientists

Finding values for string variables s.t. equations evaluate true

- String term: finite concatenation of string variables and characters
- String variable: represent finite character sequences
- Empty string: neutral element denoted by ε (aka "").
- Important reasoning problem in software/protocol verification

Example String Equation

Example

Problem:

Formally: $axaz = yby$

Java: `"a" + x + "a" + z == y + "b" + y`

Example String Equation

Example

Problem:

Formally: $axaz = yby$

Java: `"a" + x + "a" + z == y + "b" + y`

Solution:

Formally: $\mathscr{I} = \{\, x/b,\ \ y/a,\ \ z/\varepsilon \,\}$

Java: `x = "b", y = "a", z = ""`

Example String Equation

Example

Problem:

Formally: $axaz = yby$

Java: `"a" + x + "a" + z == y + "b" + y`

Solution:

Formally: $\mathscr{I} = \{\, x/b,\ \ y/a,\ \ z/\varepsilon \,\}$

Java: `x = "b", y = "a", z = ""`

Check:

Formally: $aba = aba$

Java: `"a" + "b" + "a" + "" == "a" + "b" + "a"`

String Term/Token

String Term: Sequence of tokens (u, v, w)

String Tokens:

- string variables: x, y, z
- (distinct) characters: a, b, c, d

String Term/Token

String Term: Sequence of tokens (u, v, w)
String Tokens:

- string variables: x, y, z
- (distinct) characters: a, b, c, d

Interpretation/Substitution

Maps every string variable x to string term: x/u

String Term/Token

String Term: Sequence of tokens (u, v, w)
String Tokens:

- string variables: x, y, z
- (distinct) characters: a, b, c, d
- powers: u^n

Interpretation/Substitution

Maps every string variable x to string term: x/u

Word Equations

- Important for software/protocol verification

Word Equations

- Important for software/protocol verification
- Makanin (1977): equational fragment (e.g., $axaz = yby$) is decidable

Word Equations

- Important for software/protocol verification
- Makanin (1977): equational fragment (e.g., $axaz = yby$) is decidable
- Plandowski (1999), Jez (2012): decidable in PSPACE

Word Equations

- Important for software/protocol verification
- Makanin (1977): equational fragment (e.g., $axaz = yby$) is decidable
- Plandowski (1999), Jez (2012): decidable in PSPACE
- Still: SMT solver incomplete approaches (theory interplay)
  - Automatons (z3-noodler, Ostrich, Sloth, TRAU, ...)
  - SAT (Woorpje, nfa2sat)
  - Unwinding/model repair (z3, cvc5, S3, z3Str3, ...)
  - Nielsen transformation (more or less in all solvers)
  - ...

Word Equations

- Important for software/protocol verification
- Makanin (1977): equational fragment (e.g., $axaz = yby$) is decidable
- Plandowski (1999), Jez (2012): decidable in PSPACE
- Still: SMT solver incomplete approaches (theory interplay)
  - Automatons (z3-noodler, Ostrich, Sloth, TRAU, ...)
  - SAT (Woorpje, nfa2sat)
  - Unwinding/model repair (z3, cvc5, S3, z3Str3, ...)
  - Nielsen transformation (more or less in all solvers)
  - ...
- Termination only on "common-fragments"

  quadratic, 2-variable, straight-line, chain-free, ...

What this Talk is About

- Background
  - What is string equation solving? ✓
  - Nielsen transformation
- New string equation solving calculus using interplay of
  - ground power introduction
  - generalised Parikh images
    - multi-sequence (MS) Parikh
    - multi-sequence error bounded (MSEB) Parikh

Nielsen Transformation

Nielsen Transformation

Originally used for free-*groups* (1921)

High-level idea

For $tu = sv$ with token t and s

- Align t and s by splitting
  - "t is a prefix of s or s is a prefix of t"

Nielsen Transformation

Originally used for free-*groups* (1921)

High-level idea

For $tu = sv$ with token t and s

- Align t and s by splitting
  - "t is a prefix of s or s is a prefix of t"
- Prune equal prefixes

Nielsen Transformation

Originally used for free-*groups* (1921)

High-level idea

For $tu = sv$ with token t and s

- Align t and s by splitting
  - "t is a prefix of s or s is a prefix of t"
- Prune equal prefixes
- Repeat until $\varepsilon = \varepsilon$

Nielsen Transformation

Originally used for free-*groups* (1921)

High-level idea

For $tu = sv$ with token t and s

- Align t and s by splitting
  - "t is a prefix of s or s is a prefix of t"
- Prune equal prefixes
- Repeat until $\varepsilon = \varepsilon$
- Backtrack on obvious conflicts

Nielsen Transformation

Originally used for free-*groups* (1921)

High-level idea

For $tu = sv$ with token t and s

- Align t and s by splitting
  - "t is a prefix of s or s is a prefix of t"
- Prune equal prefixes
- Repeat until $\varepsilon = \varepsilon$
- Backtrack on obvious conflicts

Example ($axaz = yby$)

Align a and y – *nothing else matters* (for now...)!

Nielsen Transformation

Originally used for free-*groups* (1921)

High-level idea

For $tu = sv$ with token t and s

- Align t and s by splitting
  - "t is a prefix of s or s is a prefix of t"
- Prune equal prefixes
- Repeat until $\varepsilon = \varepsilon$
- Backtrack on obvious conflicts

Example ($axaz = yby$)

Align a and y – *nothing else matters* (for now...)!

Represented as "(destructive) unsigned tableau calculus"

Nielsen Rules

$$\frac{au = bv}{\bot} \qquad \frac{au = \varepsilon}{\bot} \qquad \frac{wu = wv}{\mathrm{u} = \mathrm{v}} \qquad \frac{xu = \varepsilon}{(xu = \varepsilon)[x/\varepsilon]}$$

Nielsen Rules

$$\frac{au = bv}{\bot} \qquad \frac{au = \varepsilon}{\bot} \qquad \frac{wu = wv}{\mathrm{u} = \mathrm{v}} \qquad \frac{xu = \varepsilon}{(xu = \varepsilon)[x/\varepsilon]}$$

$$\frac{xu = av}{(xu = av)[x/\varepsilon] \qquad (xu = av)[x/ax']}$$

$$\frac{xu = yv}{(xu = yv)[x/y] \qquad (xu = yv)[x/yx'] \qquad (xu = yv)[y/xy']}$$

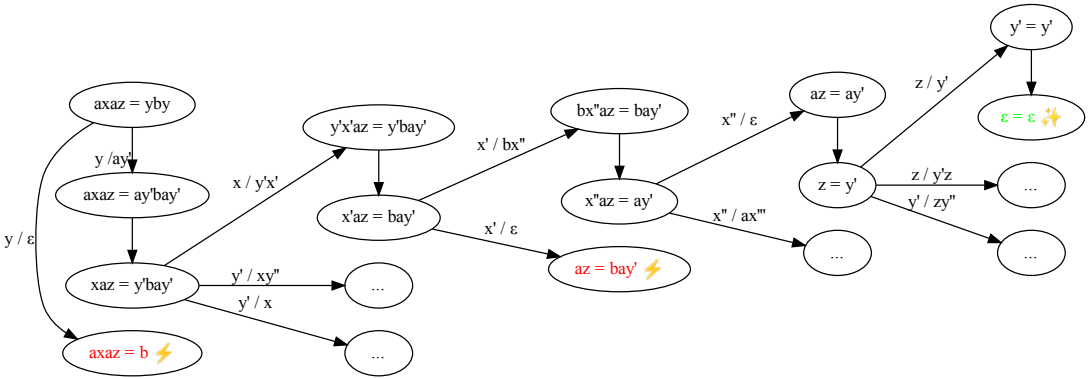
Nielsen Rules

$$\frac{au = bv}{\bot} \qquad \frac{au = \varepsilon}{\bot} \qquad \frac{wu = wv}{\mathrm{u} = \mathrm{v}} \qquad \frac{xu = \varepsilon}{(xu = \varepsilon)[x/\varepsilon]}$$

$$\frac{xu = av}{(xu = av)[x/\varepsilon] \qquad (xu = av)[x/ax']}$$

$$\frac{xu = yv}{(xu = yv)[x/y] \qquad (xu = yv)[x/yx'] \qquad (xu = yv)[y/xy']}$$

Nielsen Transformation as Graph



Thank you for attention?

Nielsen Transformation – Properties

Nielsen transformation is...

- sound

Nielsen Transformation – Properties

Nielsen transformation is...

- sound
- semi-complete
  - If there is a model, we will find it

Nielsen Transformation – Properties

Nielsen transformation is...

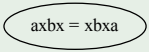
- sound
- semi-complete
  - If there is a model, we will find it
- not terminating in general

What this Talk is About

- Background
  - What is string equation solving? ✓
  - Nielsen transformation ✓
- New string equation solving calculus using interplay of
  - ground power introduction
  - generalised Parikh images
    - multi-sequence (MS) Parikh
    - multi-sequence error bounded (MSEB) Parikh

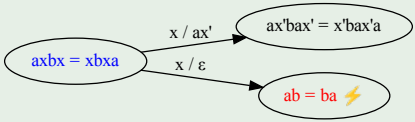
Nielsen Transformation can Diverge

Example ($axbx = xbxa$)



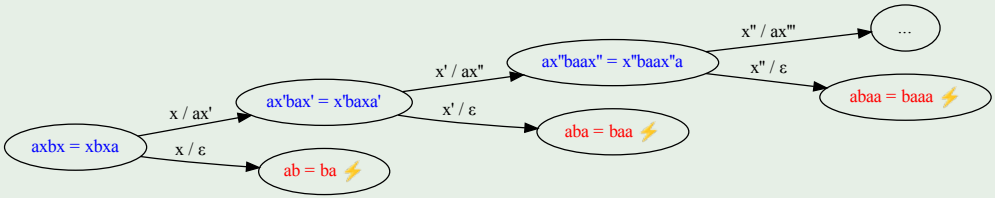
Nielsen Transformation can Diverge

Example ($axbx = xbxa$)



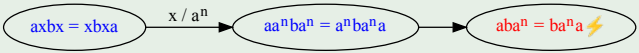
Nielsen Transformation can Diverge

Example ($axbx = xbxa$)



. . . but some Cases can be Fixed

Example ($axbx = xbxa$)



Ground Power Introduction

(Ground) Powers in Detail

We define u^n as

$$u^0 := \varepsilon \qquad u^{n+1} := uu^n$$

Example $(((ab)^2c)^3)$

Represents: *ababc ababc ababc*

(Ground) Powers in Detail

We define u^n as

$$u^0 := \varepsilon \qquad u^{n+1} := uu^n$$

Example $(((ab)^2c)^3)$

Represents: *ababc ababc ababc*

Ground Power Introduction

For u variable free

$$wxu = xv \models \exists n \in \mathbb{N} : \exists w' \in prefix(w) : x = w^n w'$$

(Ground) Powers in Detail

We define u^n as

$$u^0 := \varepsilon \qquad u^{n+1} := uu^n$$

Example $(((ab)^2c)^3)$

Represents: *ababc ababc ababc*

Ground Power Introduction

For u variable free

$$wxu = xv \models \exists n \in \mathbb{N} : \exists w' \in prefix(w) : x = w^n w'$$

What are n and w'?

(Ground) Powers in Detail

We define u^n as

$$u^0 := \varepsilon \qquad u^{n+1} := uu^n$$

Example $(((ab)^2c)^3)$

Represents: *ababc ababc ababc*

Ground Power Introduction

For u variable free

$$wxu = xv \models \exists n \in \mathbb{N} : \exists w' \in prefix(w) : x = w^n w'$$

What are n and w'?

- Keep n symbolic (integer Skolem constant)

(Ground) Powers in Detail

We define u^n as

$$u^0 := \varepsilon \qquad u^{n+1} := uu^n$$

Example $(((ab)^2c)^3)$

Represents: *ababc ababc ababc*

Ground Power Introduction

For u variable free

$$wxu = xv \models \exists n \in \mathbb{N} : \exists w' \in prefix(w) : x = w^n w'$$

What are n and w'?

- Keep n symbolic (integer Skolem constant)
- w var-free $\rightsquigarrow$ $prefix(w)$ finite, var-free, and computable $\rightsquigarrow$ finite splits

Examples Power Introduction (1/3)

Example ($abx = xba$)

Examples Power Introduction (1/3)

Example ($abx = xba$)

Two cases

1. $x = (ab)^n$

Examples Power Introduction (1/3)

Example ($abx = xba$)

Two cases

1. $x = (ab)^n$

   $\hookrightarrow\ ab(ab)^n = (ab)^n ba \rightsquigarrow ab = ba$ ↯

Examples Power Introduction (1/3)

Example ($abx = xba$)

Two cases

1. $x = (ab)^n$
   $\hookrightarrow\ ab(ab)^n = (ab)^n ba \rightsquigarrow ab = ba$ ↯
2. $x = (ab)^n a$.

Examples Power Introduction (1/3)

Example ($abx = xba$)

Two cases

1. $x = (ab)^n$
   $\hookrightarrow\ ab(ab)^n = (ab)^n ba \rightsquigarrow ab = ba$ ↯
2. $x = (ab)^n a$.
   $\hookrightarrow\ ab(ab)^n a = (ab)^n aba \rightsquigarrow \varepsilon = \varepsilon$ ☺

Examples Power Introduction (2/3)

Example ($xaby = ybax$)

Examples Power Introduction (2/3)

Example ($xaby = ybax$)

Ground power introduction not applicable ¯_(ツ)_/¯

Examples Power Introduction (3/3)

Example $((ab)^n cx = xbac)$

Examples Power Introduction (3/3)

Example $((ab)^n cx = xbac)$

Three cases

- $x = ((ab)^n c)^m (ab)^l \wedge 0 \leq l < m$
- $x = ((ab)^n c)^m (ab)^l a \wedge 0 \leq l < m$
- $x = ((ab)^n c)^m (ab)^n$

Some Power Rules

Some Power Rules

$$\frac{xu = wxv,\ w \text{ var-free}}{(xu = wxv)[x/w^n \text{prefix}(w)]} \qquad \frac{xu = w^n v}{(xu = w^n v)[x/w^n x'] \quad (xu = w^n v)[x/\text{prefix}(w^n)]}$$

Some Power Rules

$$\frac{xu = wxv,\ w \text{ var-free}}{(xu = wxv)[x/w^n \text{prefix}(w)]} \qquad \frac{xu = w^n v}{(xu = w^n v)[x/w^n x'] \qquad (xu = w^n v)[x/\text{prefix}(w^n)]}$$

$$\frac{w^m u = w^n v}{m \leq n \qquad n < m}$$

Some Power Rules

$$\frac{xu = wxv,\ w \text{ var-free}}{(xu = wxv)[x/w^n \text{prefix}(w)]} \qquad \frac{xu = w^n v}{(xu = w^n v)[x/w^n x'] \qquad (xu = w^n v)[x/\text{prefix}(w^n)]}$$

$$\frac{w^m u = w^n v}{m \leq n \qquad n < m} \qquad \frac{au = w^n v}{n = 0 \qquad n > 0}$$

Some Power Rules

$$\frac{xu = wxv,\ w \text{ var-free}}{(xu = wxv)[x/w^n \text{prefix}(w)]} \qquad \frac{xu = w^n v}{(xu = w^n v)[x/w^n x'] \quad (xu = w^n v)[x/\text{prefix}(w^n)]}$$

$$\frac{w^m u = w^n v}{m \leq n \quad n < m} \qquad \frac{au = w^n v}{n = 0 \quad n > 0}$$

Combination with integer reasoning

One Second to Recap

- String equation solving: Aligning tokens
- Algorithmically/Calculus: Nielsen transformation
  - Incomplete for UNSAT
  - Extended by now with explicit powers
    - Performance improvement
    - More termination

What this Talk is About

- Background
  - What is string equation solving? ✓
  - Nielsen transformation ✓
- New string equation solving calculus using interplay of
  - ground power introduction ✓
  - generalised Parikh images
    - multi-sequence (MS) Parikh
    - multi-sequence error bounded (MSEB) Parikh

Length and Parikh Abstraction

Length Abstraction

Clearly,

$$u = v \models |u| = |v|$$

Length Abstraction

Clearly,

$$u = v \models |u| = |v|$$

where length is a homomorphism w.r.t. concatenation:

$$|uv| \rightsquigarrow |u| + |v|$$

Clearly,

$$u = v \models |u| = |v|$$

where length is a homomorphism w.r.t. concatenation:

$$|uv| \rightsquigarrow |u| + |v|$$

Example ($xayy = ybxx$)

$$xayy = ybxx \rightsquigarrow |xayy| = |yaxx| \rightsquigarrow |x| = |y|$$

Clearly,

$$u = v \;\models\; |u| = |v|$$

where length is a homomorphism w.r.t. concatenation:

$$|uv| \;\rightsquigarrow\; |u| + |v|$$

Example ($xayy = ybxx$)

$$xayy = ybxx \;\rightsquigarrow\; |xayy| = |yaxx| \;\rightsquigarrow\; |x| = |y|$$

Thus, $x = y \;\rightsquigarrow\; xaxx = xbxx \;\rightsquigarrow\; a = b$ ↯

Parikh Image

Length abstraction often too coarse:

Example ($xay = ybx$)

Length abstraction gives $0 = 0$ ¯_(ツ)_/¯

Length abstraction often too coarse:

Example ($xay = ybx$)

Length abstraction gives $0 = 0$ ¯_(ツ)_/¯

Parikh Image $\alpha_a(u)$

Number of a in u.

$$u = v \models \alpha_a(u) = \alpha_a(v)$$

$$\alpha_a(uv) \rightsquigarrow \alpha_a(u) + \alpha_a(v)$$

Parikh Image

Length abstraction often too coarse:

Example ($xay = ybx$)

Length abstraction gives $0 = 0$ ¯_(ツ)_/¯

Parikh Image $\alpha_a(u)$

Number of a in u.

$$u = v \models \alpha_a(u) = \alpha_a(v)$$

$$\alpha_a(uv) \rightsquigarrow \alpha_a(u) + \alpha_a(v)$$

Example ($xay = ybx$ – ctd.)

$$\alpha_a(xay) = \alpha_a(ybx)$$

$$\alpha_a(x) + 1 + \alpha_a(y) = \alpha_a(y) + \alpha_a(x) \lightning$$

Generalizing Parikh Image

Example ($xabcy = ybacx$)

unsatisfiable . . . but presented approaches will diverge

Parikh abstraction: powerful, but often still too weak.

Generalizing Parikh Image

Example ($xabcy = ybacx$)

unsatisfiable . . . but presented approaches will diverge

Parikh abstraction: powerful, but often still too weak.e.g.,

- $xabcy = ybacx \rightsquigarrow$ multi-sequence (MS) Parikh

Generalizing Parikh Image

Example ($xabcy = ybacx$)

unsatisfiable . . . but presented approaches will diverge

Parikh abstraction: powerful, but often still too weak.e.g.,

- $xabcy = ybacx \rightsquigarrow$ multi-sequence (MS) Parikh
- $xaxaaabbby = xxababababy \rightsquigarrow$ multi-sequence error bounded (MSEB) Parikh

What this Talk is About

- Background
  - What is string equation solving? ✓
  - Nielsen transformation ✓
- New string equation solving calculus using interplay of
  - ground power introduction ✓
  - generalised Parikh images
    - multi-sequence (MS) Parikh
    - multi-sequence error bounded (MSEB) Parikh

Multi-Sequence (MS) Parikh

MS Parikh – $xabcy = ybacx$

We realize that the number of bc are different on both sides, so

$$\alpha_{bc}(xabcy) = \alpha_{bc}(x) + 1 + \alpha_{bc}(y)$$

$$\alpha_{bc}(ybacx) = \alpha_{bc}(y) + \alpha_{bc}(x)$$

We realize that the number of *bc* are different on both sides, so

$$\alpha_{bc}(xabcy) = \alpha_{bc}(x) + 1 + \alpha_{bc}(y)$$
$$\alpha_{bc}(ybacx) = \alpha_{bc}(y) + \alpha_{bc}(x)$$

Question

Why not search for *ab*?

MS Parikh – $xabcy = ybacx$

We realize that the number of bc are different on both sides, so

$$\alpha_{bc}(xabcy) = \alpha_{bc}(x) + 1 + \alpha_{bc}(y)$$
$$\alpha_{bc}(ybacx) = \alpha_{bc}(y) + \alpha_{bc}(x)$$

Question

Why not search for ab?

$$\alpha_{ab}(xabcy) = \alpha_{ab}(x) + 1 + \alpha_{ab}(y)$$
$$\alpha_{ab}(ybacx) = \alpha_{ab}(yb) + \alpha_{ab}(x)$$

$\alpha_{ab}(yb)$ cannot be further decomposed!

We realize that the number of bc are different on both sides, so

$$\alpha_{bc}(xabcy) = \alpha_{bc}(x) + 1 + \alpha_{bc}(y)$$
$$\alpha_{bc}(ybacx) = \alpha_{bc}(y) + \alpha_{bc}(x)$$

Question

Why not search for ab?

$$\alpha_{ab}(xabcy) = \alpha_{ab}(x) + 1 + \alpha_{ab}(y)$$
$$\alpha_{ab}(ybacx) = \alpha_{ab}(yb) + \alpha_{ab}(x)$$

$\alpha_{ab}(yb)$ cannot be further decomposed!

- ab might be crossing in yb!
- bc was definitely not crossing

So Restriction to Definitely Non-Crossing?

No – we lose interesting cases!

Example ($xxabcyy = yybacxx$)

Any u with $|u| > 1$ can crossing in xx or yy!

So Restriction to Definitely Non-Crossing?

No – we lose interesting cases!

Example ($xxabcyy = yybacxx$)

Any u with $|u| > 1$ can crossing in xx or yy!

$\rightsquigarrow$ bc is crossing if $x/cx'b$

$\hookrightarrow$ error bounded Parikh

Parikh Grouping

1. Given some equation $u = v$ with $\mathrm{Vars}(u) = \mathrm{Vars}(v)$
   - ↪ e.g., $xxabcyy = yybacxx$

Parikh Grouping

1. Given some equation $u = v$ with $\mathrm{Vars}(u) = \mathrm{Vars}(v)$
   - ↪ e.g., $xxabcyy = yybacxx$
2. Chose some "pattern" $w \in \Sigma^+$ such that
   1. w does not end with any of its prefixes ("unbordered")
      - ↪ e.g., $w := abcab$ is not permitted

Parikh Grouping

1. Given some equation $u = v$ with $\mathrm{Vars}(u) = \mathrm{Vars}(v)$
   - ↪ e.g., $xxabcyy = yybacxx$
2. Chose some "pattern" $w \in \Sigma^+$ such that
   1. w does not end with any of its prefixes ("unbordered")
   2. w is a syntactic subsequence of u or v
      - ↪ e.g., $w := cba$ is not permitted

Parikh Grouping

1. Given some equation $u = v$ with $\mathrm{Vars}(u) = \mathrm{Vars}(v)$
   - ↪ e.g., $xxabcyy = yybacxx$
2. Chose some "pattern" $w \in \Sigma^+$ such that
   1. w does not end with any of its prefixes ("unbordered")
   2. w is a syntactic subsequence of u or v
   3. w is maximal w.r.t. at least one side and these constraints.
      - ↪ e.g., $w := abc$ is permitted; $w := ab$ is not

Parikh Grouping

1. Given some equation $u = v$ with $\mathrm{Vars}(u) = \mathrm{Vars}(v)$
   - ↪ e.g., $xxabcyy = yybacxx$
2. Chose some "pattern" $w \in \Sigma^+$ such that
   1. w does not end with any of its prefixes ("unbordered")
   2. w is a syntactic subsequence of u or v
   3. w is maximal w.r.t. at least one side and these constraints.
3. Group u and v such there cannot be crossings with pattern w
   - ↪ e.g., using $w := abc$ gives

$$\{\ (xx),\ (abc),\ (yy)\ \}^{abc} = \{\ (yy),\ (bac),\ (xx)\ \}^{abc}$$

Parikh Grouping

1. Given some equation $u = v$ with $\mathrm{Vars}(u) = \mathrm{Vars}(v)$
   - ↪ e.g., $xxabcyy = yybacxx$
2. Chose some "pattern" $w \in \Sigma^+$ such that
   1. w does not end with any of its prefixes ("unbordered")
   2. w is a syntactic subsequence of u or v
   3. w is maximal w.r.t. at least one side and these constraints.
3. Group u and v such there cannot be crossings with pattern w
4. Cancel out common subsequences
   - ↪ e.g.,

$$\{\ (xx),\ (abc),\ (yy)\ \}^{abc} = \{\ (yy),\ (bac),\ (xx)\ \}^{abc}$$

$$\rightsquigarrow\ \{\ (abc)\ \}^{abc} = \{\ (bac)\ \}^{abc}$$

Parikh Grouping

1. Given some equation $u = v$ with $\mathrm{Vars}(u) = \mathrm{Vars}(v)$
   - ↪ e.g., $xxabcyy = yybacxx$
2. Chose some "pattern" $w \in \Sigma^+$ such that
   1. w does not end with any of its prefixes ("unbordered")
   2. w is a syntactic subsequence of u or v
   3. w is maximal w.r.t. at least one side and these constraints.
3. Group u and v such there cannot be crossings with pattern w
4. Cancel out common subsequences
5. Replace variable-free groups (w) by 1 and others by 0
   - ↪ e.g., $|\{\ (abc)\ \}^{abc}| = |\{\ (bac)\ \}^{abc}| \rightsquigarrow 1 = 0 \rightsquigarrow \bot$

Another Example

For equation $xaxaaabbby = yxababab x$ and pattern $w := ab$ we get

$$\{\ (x),\ (ax),\ (aa),\ (ab),\ (bb),\ (y)\ \}^{ab} = \{\ (yx),\ (ab),\ (ab),\ (ab),\ (x)\ \}^{ab}$$

Another Example

For equation $xaxaaabbby = yxababababx$ and pattern $w := ab$ we get

$$\{\ (x),\ (ax),\ (aa),\ (ab),\ (bb),\ (y)\ \}^{ab} = \{\ (yx),\ (ab),\ (ab),\ (ab),\ (x)\ \}^{ab}$$

$$\{\ (ax),\ (y)\ \}^{ab} = 2 + \{\ (yx)\ \}^{ab}$$

No conflict! Are we done?

Another Example – Error-Bounded Parikh Image

For equation $xaxaaabbby = yxababababx$ and pattern $w := ab$ we get

$$\{\ (x),\ (ax),\ (aa),\ (ab),\ (bb),\ (y)\ \}^{ab} = \{\ (yx),\ (ab),\ (ab),\ (ab),\ (x)\ \}^{ab}$$

$$\{\ (ax),\ (y)\ \}^{ab} = 2 + \{\ (yx)\ \}^{ab}$$

We over-/under-approximate!

- Overapprox. LHS: $\{\ (ax),\ (y)\ \}^{ab} \leq 1 + \{\ (x),\ (y)\ \}^{ab}$
- Underapprox. RHS: $2 + \{\ (yx)\ \}^{ab} \geq 2 + \{\ (y),\ (x)\ \}^{ab}$

Another Example – Error-Bounded Parikh Image

For equation $xaxaaabbby = yxababababx$ and pattern $w := ab$ we get

$$\{\ (x),\ (ax),\ (aa),\ (ab),\ (bb),\ (y)\ \}^{ab} = \{\ (yx),\ (ab),\ (ab),\ (ab),\ (x)\ \}^{ab}$$

$$\{\ (ax),\ (y)\ \}^{ab} = 2 + \{\ (yx)\ \}^{ab}$$

We over-/under-approximate!

- Overapprox. LHS: $\{\ (ax),\ (y)\ \}^{ab} \leq 1 + \{\ (x),\ (y)\ \}^{ab}$
- Underapprox. RHS: $2 + \{\ (yx)\ \}^{ab} \geq 2 + \{\ (y),\ (x)\ \}^{ab}$

Thus, we have

$$\{\ (ax),\ (y)\ \}^{ab} \leq\ 1 + \{\ (x),\ (y)\ \}^{ab} \lneq 2 + \{\ (y),\ (x)\ \}^{ab} \leq\ 2 + \{\ (yx)\ \}^{ab}$$

Experiments

- Prototype implementation: ZIPT
- Evaluation on "word equations-only" SMT-LIB2 benchmarks

	ZIPT	Z3	cvc5	OSTRICH	Z3-Noodler	Z3str3	Total
`track 01`	**200**	198	191	198	**200**	**200**	*200*
`track 02`	**9**	4	1	4	6	6	*9*
`track 03`	**195**	176	164	127	190	190	*200*
`track 04`	**200**	198	**200**	198	199	**200**	*200*

Recap

- String Equation Solving: Aligning tokens
- Algorithmically: Nielsen transformation
  - Extended by explicit powers
  - Extended by generalised Parikh image


thank you for
x = y +
your attention
Questions?

Appendix

Future Work

- Improve implementation
  - ↪ currently often very inefficient

Future Work

- Improve implementation
  - ↪ currently often very inefficient
- Detect Parikh cases efficiently
  - ↪ currently we try out all cases

Future Work

- Improve implementation
  - ↪ currently often very inefficient
- Detect Parikh cases efficiently
  - ↪ currently we try out all cases
- Major problem: regular expression membership
  - ↪ call Z3-noodler?

Nielsen Transformation can be Slow

Example

$xaxbybz = axyybzzbaa$

- has the unique model

$$x_1/aaaaaaaa, \quad x_2/aaaa, \quad x_3/aa$$

- double exponential time

Why Power Normalization?

$$\frac{u[w'(w''w')^n] = v}{u[(w'w'')^n w'] = v}$$

Why Power Normalization?

$$\frac{u[w'(w''w')^n] = v}{u[(w'w'')^n w'] = v}$$

Example ($(ab)^n cx = a(ba)^m d$ – Without normalisation)

$n = 0$: $(ab)^0 cx = a(ba)^m d \rightsquigarrow cx = a(ba)^m d$ ↯

$n > 0$: $ab(ab)^{n-1} cx = a(ba)^m d \rightsquigarrow b(ab)^{n-1} cx = (ba)^m d$

Why Power Normalization?

$$\frac{u[w'(w''w')^n] = v}{u[(w'w'')^n w'] = v}$$

Example ($(ab)^n cx = a(ba)^m d$ – Without normalisation)

$n = 0$: $(ab)^0 cx = a(ba)^m d \rightsquigarrow cx = a(ba)^m d$ ↯

$n > 0$: $ab(ab)^{n-1} cx = a(ba)^m d \rightsquigarrow b(ab)^{n-1} cx = (ba)^m d$

2 other case:

$m = 0$: $b(ab)^{n-1} cx = (ba)^0 d \rightsquigarrow b(ab)^{n-1} cx = d$ ↯

Why Power Normalization?

$$\frac{u[w'(w''w')^n] = v}{u[(w'w'')^n w'] = v}$$

Example ($(ab)^n cx = a(ba)^m d$ – Without normalisation)

$n = 0$: $(ab)^0 cx = a(ba)^m d \rightsquigarrow cx = a(ba)^m d$ ↯

$n > 0$: $ab(ab)^{n-1} cx = a(ba)^m d \rightsquigarrow b(ab)^{n-1} cx = (ba)^m d$

2 other case:

$m = 0$: $b(ab)^{n-1} cx = (ba)^0 d \rightsquigarrow b(ab)^{n-1} cx = d$ ↯

$m > 0$: $b(ab)^{n-1} cx = ba(ba)^{m-1} d \rightsquigarrow (ab)^{n-1} cx = a(ba)^{m-1} d$

2 more cases ...

Why Power Normalization?

$$\frac{u[w'(w''w')^n] = v}{u[(w'w'')^n w'] = v}$$

Example ($(ab)^n cx = a(ba)^m d$ – Without normalisation)

$n = 0$: $(ab)^0 cx = a(ba)^m d \rightsquigarrow cx = a(ba)^m d$ ↯

$n > 0$: $ab(ab)^{n-1} cx = a(ba)^m d \rightsquigarrow b(ab)^{n-1} cx = (ba)^m d$

2 other case:

$m = 0$: $b(ab)^{n-1} cx = (ba)^0 d \rightsquigarrow b(ab)^{n-1} cx = d$ ↯

$m > 0$: $b(ab)^{n-1} cx = ba(ba)^{m-1} d \rightsquigarrow (ab)^{n-1} cx = a(ba)^{m-1} d$

2 more cases ...

Why Power Normalization?

$$\frac{u[w'(w''w')^n] = v}{u[(w'w'')^n w'] = v}$$

Example $((ab)^n cx = a(ba)^m d$ – With normalisation)

$$(ab)^n cx = a(ba)^m d \rightsquigarrow (ab)^n cx = (ab)^m ad$$

$n \leq m$: $cx = (ab)^{m-n} ad \rightsquigarrow \ldots$ ↯

$m < n$: $(ab)^{n-m} cx = ad \rightsquigarrow \ldots$ ↯

Equality Decomposition

Two big hammers: Power Introduction and Parikh Image

- Restricted applicability
  - Classical Nielsen: beginning/ending of string term relevant
  - Powers: $ux \ldots = x$
  - Parikh: $\mathrm{Vars}(u) = \mathrm{Vars}(v)$

Equality Decomposition

Two big hammers: Power Introduction and Parikh Image

- Restricted applicability
  - Classical Nielsen: beginning/ending of string term relevant
  - Powers: $ux\ldots = x$
  - Parikh: $\mathrm{Vars}(u) = \mathrm{Vars}(v)$
- Cases that will never transform into required form
  - e.g., $xuy = yvx$ with $\mathrm{Vars}(u) \neq \mathrm{Vars}(v)$

Equality Decomposition

Two big hammers: Power Introduction and Parikh Image

- Restricted applicability
  - Classical Nielsen: beginning/ending of string term relevant
  - Powers: $ux\ldots = x$
  - Parikh: $\mathrm{Vars}(u) = \mathrm{Vars}(v)$
- Cases that will never transform into required form
  - e.g., $xuy = yvx$ with $\mathrm{Vars}(u) \neq \mathrm{Vars}(v)$
  - Cry?

Equality Decomposition

Two big hammers: Power Introduction and Parikh Image

- Restricted applicability
  - Classical Nielsen: beginning/ending of string term relevant
  - Powers: $ux\ldots = x$
  - Parikh: $\mathrm{Vars}(u) = \mathrm{Vars}(v)$
- Cases that will never transform into required form
  - e.g., $xuy = yvx$ with $\mathrm{Vars}(u) \neq \mathrm{Vars}(v)$
  - ~~Cry?~~

  ↪ Equality Decomposition
  two equations = two beginnings/endings ☺

Equality Decomposition

Example ($xayybx = ybxxay$)

Looks hard, but is trivial, as $|xay| = |ybx|$. We split on

$$xay^{\dagger}\, ybx = ybx^{\dagger}\, xay$$

and thus

$$xay = ybx \wedge ybx = xay$$

Lazy Equality Decomposition

Example ($xayybx = yabxxy$)

Lazy Equality Decomposition

Example ($xayybx = yabxxy$)

There is no position to put †.

	y	a	b	x	x	y
x	x / y	x / y + 1	x / y + 2	0 / y + 2	0 / x + y + 2	0 / x + 2y + 2
a	x + 1 / y	x / y	x / y + 1	0 / y + 1	0 / x + y + 1	0 / x + 2y + 1
y	x + 1 / 0	x / 0	x / 1	0 / 1	0 / x + 1	0 / x + y + 1
y	x + y + 1 / 0	x + y / 0	x + y / 1	y / 1	y / x + 1	0 / x + 1
b	x + y + 2 / 0	x + y + 1 / 0	x + y / 0	y / 0	y / x	0 / x
y	x + 2y + 2 / 0	x + 2y + 1 / 0	x + 2y / 0	2y / 0	2y / x	y / x

Lazy Equality Decomposition

Example ($xayybx = yabxxy$)

There is no position to put †.

	y	a	b	x	x	y
x	x / y	x / y + 1	x / y + 2	0 / y + 2	0 / x + y + 2	0 / x + 2y + 2
a	x + 1 / y	x / y	x / y + 1	0 / y + 1	0 / x + y + 1	0 / x + 2y + 1
y	x + 1 / 0	x / 0	x / 1	0 / 1	0 / x + 1	0 / x + y + 1
y	x + y + 1 / 0	x + y / 0	x + y / 1	y / 1	y / x + 1	0 / x + 1
b	x + y + 2 / 0	x + y + 1 / 0	x + y / 0	y / 0	y / x	0 / x
y	x + 2y + 2 / 0	x + 2y + 1 / 0	x + 2y / 0	2y / 0	2y / x	y / x

They only differ by 1 at $xay^{\dagger}ybx = yabx^{\dagger}xy$.

$$\hookrightarrow y/\varepsilon \ \vee \ y/oy'$$

with o being *some unknown* character

Lazy Equality Decomposition

Example ($xayybx = yabxxy$)

$$\hookrightarrow y/\varepsilon \ \vee \ y/oy'$$

with o being *some unknown* character
If y/oy':

$$xaoy'o \ ^{\dagger} \ y'bx = oy'abx \ ^{\dagger} \ xoy'$$

As $|xaoy'o| = |oy'abx|$ we split into

$$\rightsquigarrow xaoy'o = oy'abx \wedge y'bx = xoy'$$

Power Introduction Rules

Power Introduction Rules

$$\frac{xu = wxv,\ w \text{ var-free}}{x/w^n \,\mathrm{prefix}(w)}$$

Power Introduction Rules

$$\frac{xu = wxv,\ w \text{ var-free}}{x/w^n \,\text{prefix}(w)} \qquad \frac{xu = w^n v}{x/w^n x' \qquad x/\,\text{prefix}(w^n)}$$

Power Introduction Rules

$$\frac{xu = wxv,\ w \text{ var-free}}{x/w^n \,\mathrm{prefix}(w)} \qquad \frac{xu = w^n v}{x/w^n x' \qquad x/\mathrm{prefix}(w^n)}$$

$$\frac{\begin{array}{c} x_1 u_1 = w_1 x_m v_1 \\ \dots \\ x_m u_m = w_m x_{m-1} v_m \\ \text{all } w_i \text{ var-free} \end{array}}{x_i/(w_i w_{i-1} \dots w_1 w_m \dots w_{i+1})^n \,\mathrm{prefix}(w_i w_{i-1} \dots w_1 w_m \dots w_{i+1})}$$

Power Elimination Rules

Power Elimination Rules

$$\frac{w^m u = w^n v, \quad \vdash_{\mathbb{Z}} \; m < n}{\cancel{w^m u = w^n v} \qquad u = w^{n-m} v}$$

$$\frac{w^m u = w^n v}{m \leq n \qquad n < m}$$

Power Elimination Rules

$$\frac{w^m u = w^n v, \quad \vdash_{\mathbb{Z}} \; m < n}{\cancel{w^m u = w^n v} \qquad u = w^{n-m} v} \qquad \frac{au = w^n v, \quad \vdash_{\mathbb{Z}} \; n > 0}{\cancel{au = w^n v} \qquad au = ww^{n-1} v}$$

$$\frac{w^m u = w^n v}{m \leq n \qquad n < m} \qquad \frac{au = w^n v}{n = 0 \qquad n > 0}$$

Power Elimination Rules

$$\frac{w^m u = w^n v, \quad \vdash_{\mathbb{Z}} \; m < n}{\cancel{w^m u = w^n v} \qquad u = w^{n-m} v} \qquad \frac{au = w^n v, \quad \vdash_{\mathbb{Z}} \; n > 0}{\cancel{au = w^n v} \qquad au = ww^{n-1} v}$$

$$\frac{w^m u = w^n v}{m \leq n \qquad n < m} \qquad \frac{au = w^n v}{n = 0 \qquad n > 0}$$

$$\frac{w^n u = \varepsilon}{n = 0}$$

Power Normalisation Rules

$$\frac{u[(w^n)^m] = v}{u[w^{mn}] = v} \qquad \frac{u[w^n] = v, \ \vdash_{\mathbb{Z}} \ n = 0}{u[\varepsilon] = v} \qquad \frac{u[w^n] = v, \ \vdash_{\mathbb{Z}} \ n = 1}{u[w] = v}$$

Power Normalisation Rules

$$\frac{u[(w^n)^m] = v}{u[w^{mn}] = v} \qquad \frac{u[w^n] = v, \ \vdash_{\mathbb{Z}} \ n = 0}{u[\varepsilon] = v} \qquad \frac{u[w^n] = v, \ \vdash_{\mathbb{Z}} \ n = 1}{u[w] = v}$$

$$\frac{u[ww] = v}{u[w^2] = v} \qquad \frac{u[w^n w] = v}{u[w^{n+1}] = v} \qquad \frac{u[ww^n] = v}{u[w^{n+1}] = v} \qquad \frac{u[w'(w''w')^n] = v}{u[(w'w'')^n w'] = v}$$

Required for termination in certain cases!

Power Introduction $\stackrel{?}{=}$ Saviour

Power Introduction $\neq$ Saviour!

Power Introduction $\neq$ Saviour!

Example ($xaby = yabx \wedge xaby = byax$)

$$xaby = yabx \wedge xaby = byax$$

Power Introduction ≠ Saviour!

Example ($xaby = yabx \wedge xaby = byax$)

$$xaby = yabx \wedge xaby = byax \quad \rightsquigarrow^{x/bx} \quad bxaby = yabbx \wedge xaby = yabx$$

Power Introduction ≠ Saviour!

Example ($xaby = yabx \wedge xaby = byax$)

$xaby = yabx \wedge xaby = byax \quad \rightsquigarrow^{x/bx} \quad bxaby = yabbx \wedge xaby = yabx$

$bxaby = yabbx \wedge xaby = yabx$

Power Introduction ≠ Saviour!

Example ($xaby = yabx \wedge xaby = byax$)

$$xaby = yabx \wedge xaby = byax \quad \rightsquigarrow^{x/bx} \quad bxaby = yabbx \wedge xaby = yabx$$

$$bxaby = yabbx \wedge xaby = yabx \quad \rightsquigarrow^{y/by} \quad xabby = yabbx \wedge xabby = byabx$$

Power Introduction ≠ Saviour!

Example ($xaby = yabx \wedge xaby = byax$)

$$xaby = yabx \wedge xaby = byax \rightsquigarrow^{x/bx} bxaby = yabbx \wedge xaby = yabx$$

$$bxaby = yabbx \wedge xaby = yabx \rightsquigarrow^{y/by} xabby = yabbx \wedge xabby = byabx$$

$\ldots$

Actually, $x = b^n$ **or** $y = b^n$