arXiv:2603.05273v1 [cs.LO] 5 Mar 2026

通过幂和 Parikh 像求解字符串方程

Clemens Eisenhofer[1(✉)] Theodor Seiser[1], Nikolaj Bjørner[2], 和 Laura Kovács[1]

[1] 维也纳工业大学，维也纳，奥地利
clemens.eisenhofer@tuwien.ac.at[2] 微软研究院，雷德蒙德，美国

摘要。 我们提出了一种求解字符串方程的新方法，将其作为 Nielsen 变换的扩展。我们的方法的关键在于结合三种技术：字符串的幂算子、Parikh 像的推广以及等式分解。借助这些方法，我们能够求解复杂的字符串方程，包括较少遇到的字符串可满足性模理论输入。

关键词： 字符串方程求解 · 可满足性模理论 · Nielsen 变换 · 字符串幂 · Parikh 像 · 等式分解

1 引言

字符串求解被广泛应用于多种场景，包括形式化验证 [1,9],安全分析 [14,15,33,34], 以及自动推理 [2,17]。现代可满足性模理论（可满足性模理论）求解器 [5], ，如 Z3 [24], cvc5 [3],或 Princess [29]„ 通过专门的求解技术支持字符串约束，以处理长度约束 [26], 正则表达式 [7,30], 包含谓词 [27], 以及许多其他约束。然而，求解涉及长重复子序列或包含相互依赖的字符串变量的字符串（字）方程，对最先进的求解器而言仍然极具挑战 [1,9,10,12,20,23,24,33]。

例 *1*（动机示例）。考虑以下两个字符串方程的合取：

$$x_3x_3x_4bx_5b \simeq x_5x_5x_5x_5x_4bb, \quad (E_1)$$
$$x_1x_1acx_2x_4x_2x_5x_3bax_5x_3x_4x_3 \simeq x_2x_2abcx_1x_1x_3x_3x_3x_4x_4ax_4 \quad (E_2)$$

其中 $\simeq$ 表示（一阶）相等， x_1， x_2， x_3， x_4， x_5 是字符串变量，而a， b， c 是常数字符。求解此类方程需要对依赖于自身或彼此相互依赖的字符串变量进行推理，而这大多超出当前技术的范围。

我们的工作应对了与求解方程 (E_1)–(E_2) 类似的挑战。我们的方法将字符串方程上的 重写和生成规则 （表 1）作为 Nielsen 图 [17,22,26] 上的 Nielsen 变换（第 3 节）。我们通过以下方式扩展 Nielsen 变换规则：

等式分解 – 在推理过程中将字符串方程分解为子方程。使用等式分解可大幅提高字符串幂推理和 Parikh像的适用性。

显式幂表示——用于求解字符串变量依赖于自身的字符串方程。此外，它还使我们能够紧凑地推理较长但重复的字符串（第5节）。

广义 Parikh 图像——当 Nielsen 规则和字符串幂推理均无法奏效时，用于检测不可满足性（第6节）。我们调整了 Parikh 像的广义形式 [25] ，以用于字符串求解。

我们使用 SMT 求解器 Z3 [24] 的用户传播框架 [8]，将该框架实现为名为 ZIPT 的原型。我们的实验表明，该方法在 `SMT-LIB2` 仅包含方程的基准测试 [4] 上具有实际应用潜力（第7节）。

2 预备知识

字符串。为满足我们的需要，我们将采用字符串项的扩展定义。

定义 1（Token 与字符串项）。我们固定一个由 $\mathcal{K}$ 组成的集合，并用 $\mathcal{T}$ 表示字符串项的集合。字符串项 $u \in \mathcal{T}$ 是由有限个Token$t \in \mathcal{K}$ 组成的序列，我们区分以下几类：

– 具体字符***Token***，记作 a、b、c、d，其集合记作 Σ；– 符号字符***Token***，记作 o，其集合为 $\mathcal{V}_C$。–（字符串）变量***Token***，记作 x、y、z，其集合记作 $\mathcal{V}_S$；– 幂***Token***，形式为 u^m，其中 $u \in \mathcal{T}$ 不包含字符串变量，而 m 是任意整数项，可能包含整数变量。我们称 u 为幂标记的底，m 为幂标记的指数。

我们的词元记法可能会使用下标，我们用 $@ \in \Sigma \cup \mathcal{V}_C$ 表示（具体或符号）字符词元，用 $t \in \mathcal{T}$ 表示任意类型的词元。集合 Σ、$\mathcal{V}_C$ 和 $\mathcal{V}_S$ 两两不相交，并且 Σ 还满足有限且非空。字符串项集合 $\mathcal{T}$ 是所有有限词元序列的集合。使用正则表达式记法，我们有 $\mathcal{T} := \mathcal{K}^*$，并用 ε 表示空词元序列。ε 表示连接的单位元，我们通过将词元并置来表示连接。幂标记 u^m 表示词元 u 重复 m 次；也就是说，$u^0 = \varepsilon$，并且当 $m > 0$ 且其中 m 是某个整数项时，$u^m = uu^{m-1}$。鉴于幂项的引入方式，指数不可能为负数。我们用 $|u|$ 表示长度，其中 $u \in \mathcal{T}$；显然，$|u|$ 表示自然数。我们用 u^R 表示 u 的反向字符串项。例如，$(ax(ab)^m)^R = (ba)^m x^R a$。最后，我们注意到，符号字符 o 等价于满足 $|x| = 1$ 的字符串变量 x。尽管如此，我们仍使用符号字符，以便之后的定义更简洁。

字符串方程与代换。字符串方程是字符串项 u_1、 $u_2 \in \mathcal{T}$ 上的 $u_1 \simeq u_2$；这里，我们将 u_1 称为方程的左侧（LHS），而 u_2 则是方程的右侧（RHS）。仅包含字符串变量和具体字符的字符串方程称为纯字符串方程；也可能包含符号字符和幂项的字符串方程称为已扩展字符串方程。我们用 2^u 表示 u 中连续词元的集合。$||u||$ 表示 u 中词元的数量，其中 u 中的幂标记视为原子；我们称其为 u 的符号长度： $||\varepsilon|| := 0$ 且 $||tv|| := 1 + ||v||$。例如，在 $u = (abc)^m xb$ 中，我们有 $2^u = \{ \varepsilon, a, b, c, (abc)^m、 x、 ab、 bc、 (abc)^m x、 xb、 abc、 (abc)^m xb \}$ 和 $||u|| = 3$。

不包含字符串变量的字符串项 $u \in \mathcal{T}$ 称为<苹style id='2'>基项。因此，对于基项 u，有 $2^u \cap \mathcal{V}_S = \emptyset$。不过，基项可能包含幂和符号字符标记。

一个代换 σ 将 (i) 字符串变量 $x \in \mathcal{V}_S$ 映射为字符串项 $u \in \mathcal{T}$；并将 (ii) 符号字符 $o \in \mathcal{V}_C$ 映射为字符 @ $\in \Sigma \cup \mathcal{V}_C$。变量集合 $\{x \in \mathcal{V}_S \mid \sigma(x) \neq x\}$ 是有限的。符号字符的情况也类似。我们用 $(x/u) \in \sigma$ 表示 $\sigma(x) = u$。如果代换 σ 使字符串变量 x 满足 $(x/u) \in \sigma$，且 u 是基项，则称该代换消去了该字符串变量。表达式 C 在代换 σ下的应用 记作 $C[\sigma]$，其中 C 是字符串项，或字符串/整数（不）等式（的集合）。

代换 σ 被假定为无环且完全扩展：对于任意非恒等$(x/u) \in \sigma$ 与 $y \in 2^u$，我们有$(y/y) \in \sigma$。我们用 $m_1 \simeq m_2$ 和$m_1 \leq m_2$ 表示整数项 m_1, m_2上的整数（不）方程。

解释与模型。 给定字符串方程 $u \simeq v$，该方程的一个解释$\mathcal{M}$ 是一个代换，连同将整数变量赋到整数值的赋值。对于每个 $x \in \mathcal{V}_S \cap (2^u \cup 2^v)$，我们有$(x/w) \in \mathcal{M}$，其中 $w \in \Sigma^*$；此外，对于每个 $o \in \mathcal{V}_C \cap (2^u \cup 2^v)$，我们有$(o/a) \in \sigma$，且 $a \in \Sigma$。若把 $\mathcal{M}$ 应用到 S 会得到形如$u \simeq u$ 且满足 $u \in \Sigma^*$的简化方程集合，则解释 $\mathcal{M}$ 称为普通字符串方程集合 S 的纯模型 。若 S 具有模型 $\mathcal{M}$，则 S 可满足；否则 S 不可满足。类似地，一组扩展字符串方程与整数不等式的扩展模型 会把每个扩展字符串方程简化为 $u \simeq u$且 $u \in \mathcal{T}^*$，并满足所有整数约束。u 中的所有字符串变量和符号字符都被隐式假定为全称量化。普通字符串方程集合的任一扩展模型，都可通过将所有符号字符代换为 Σ 的某个字符、并将字符串变量代换为 Σ^*的某个元素，而变成纯模型。幂标记可以利用其已求值幂项的整数值被完全展开。

3 字符串求解工作流

给定一个非空的集合 S （由普通字符串方程组成），我们的任务是判定其可满足性。为此，我们求解与 S对应的扩展字符串方程，这些方程可能包含符号字符和幂，以及整数不等式。

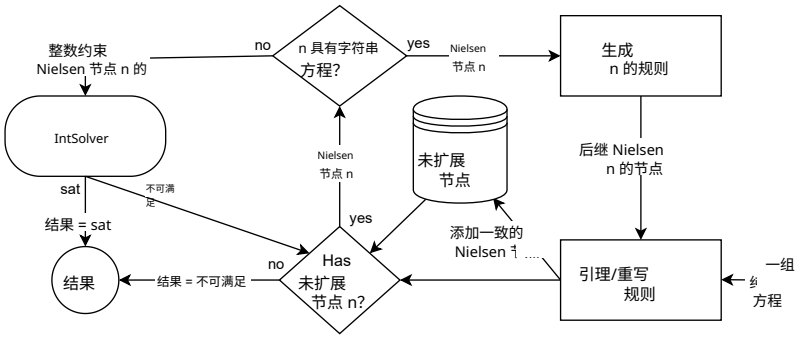

图 1. 字符串求解工作流。

为此，我们通过反复应用一组已扩展的Nielsen变换规则（第3.2节），来扩展字符串方程的Nielsen图（第3.1节）。我们将Nielsen变换推广至任意字符串项 $u \in \mathcal{T}$，而不仅仅是纯字符串项，从而化简扩展字符串方程和整数（不）等式。我们的方法总结于图1，并在下文中详述。

3.1 Nielsen 图的扩展

我们借助 Nielsen 图[26], 来检测字符串方程集合 S 的可满足性，并使用第 3.2 节中的 Nielsen 变换规则对其进行扩展和简化。

Nielsen 图。 简单来说，我们将一个***(Nielsen)*** 节点 n 视为元组$\langle \mathcal{E}(n)\ \mathcal{I}(n)\rangle$, 其中 $\mathcal{E}(n)$是扩展字符串方程的有限集， $\mathcal{I}(n)$是其整数（不）等式的有限集。特别地，与普通字符串方程输入集 S 对应的 Nielsen 节点为 $\langle S,\ \emptyset\rangle$；该节点是 S 的 Nielsen 图的根节点。

我们假设， $\mathcal{E}(n)$和 $\mathcal{I}(n)$中显然满足的约束（例如 $\varepsilon \simeq \varepsilon$或 $0 \leq 1$）会从 n中隐式移除。给定某个整数（不）等式 C，我们用 $n \models C$ 表示可以从 $\mathcal{I}(n)$推导出 C 。

Nielsen 节点 n 有有限个后继节点，若满足以下任一条件，则称其为不一致: (i) $\bot \in \mathcal{E}(n)$，或 (ii) $\mathcal{I}(n) \models \bot$，或 (iii) n 的所有后继节点均不一致；其中， $\bot$ 表示恒假公式。相反，若满足以下条件，则满足 n ： (i) $\mathcal{E}(n) = \emptyset$，且 (ii) $\mathcal{I}(n) \not\models \bot$。为检查 $n \models C$，我们假设可以访问一个可靠的整数推理器 IntSolver。

如果节点 n 的后继节点已添加到 Nielsen 图中，且 n 既未满足也不一致，那么该节点就是已扩展的。相应地，尚未扩展的节点称为未扩展。Nielsen 图记录了 Nielsen 节点之间的依赖关系，因此可通过子目标归约见证约束的（不）可满足性：如果节点 n 的约束 $\mathcal{E}(n) \cup \mathcal{I}(n)$ 有模型，那么 n 的至少一个后继节点也有模型。因此，Nielsen 图可视为证明对象或表 tableau 推导的一种变体。

通过扩展 *Nielsen* 图求解字符串。 我们使用 Nielsen 图来表示和处理（初始的）纯字符串方程集合 S 。为此，我们应用已扩展的 Nielsen变换规则（第 3.2 节及后续章节中的扩展），以简化 Nielsen 节点。下文中，令 simpl(n) 表示 Nielsen 节点 n 的简化版本。

输入集 S 中纯字符串方程的可满足性，通过递归扩展 S 的 Nielsen 图来判定，如图 1 所示。首先，(i) 将纯方程集 S 添加到 Nielsen 图中，形式为一个未扩展且经过简化的 Nielsen 节点 simpl($\langle S,\ \emptyset\rangle$)，并将其作为 Nielsen 图的根节点；此处使用第 3.2 节中的重写规则和引理规则。接下来，(ii) 从 Nielsen 图中任选一个未扩展的 Nielsen 节点 n ，并通过第 3.2 节中的某个生成规则生成其后继节点 n_1，.....， n_k。然后，(iii) 考虑 n 的后继节点 n_i 中那些不一致节点的简化版本 simpl(n_i)。这些 n 的简化后继节点会作为新的未扩展节点添加到 Nielsen 图中，并通过相应的边将 n_i 与 n 连接起来——通常我们会返回步骤 (ii)。一旦 (iv) 找到一个从根节点可达的满足节点 n，我们就报告初始方程集 S 可满足。根据已扩展 Nielsen 图的构造可知， S 的可满足性成立：如果一个节点 n 满足，则其所有字符串方程 $\mathcal{E}$（n）均已被移除，且其剩余整数约束 $\mathcal{I}$（n）可满足。另一方面，(v) 如果再也没有从根节点可达的未扩展 Nielsen 节点，且未找到满足节点，我们便判定输入集 S 不可满足。

备注 *1*（扩展 *Nielsen* 图的终止性）。如果一组普通字符串方程 S 可满足，则至少存在一个满足的 Nielsen 节点 n ，它可以从根节点 simpl($\langle S, \emptyset\rangle$) 到达。然而，一个完全扩展的 Nielsen 图可能是无限大的，并且可能包含无限多个满足的节点。对于不可满足实例，无法保证终止。由于我们的规则（第3.2节）通常不可逆—— n 可满足仅意味着其某一个后继节点也可满足——因此在实践中，我们需要采用公平的树遍历方法，例如广度优先或迭代加深，以便使我们的方法在可满足实例上更可靠地终止 S。

3.2 已扩展 Nielsen 变换规则

我们的字符串求解过程通过重写并在相应的 Nielsen 图中生成 Nielsen 节点 n 来扩展字符串方程的 Nielsen 图，如第3.1节所述。具体而言，我们通过 simpl(n)，利用以下规则简化某个节点 n ：（i）引理规则；（ii）项重写规则；以及（iii）方程改写规则，从而得到更易处理的节点 simpl(n)。我们使用（iv）生成规则引入 n 的后继节点 n' ，其中 n' 的约束由将代换 $\sigma_{n'}$ 应用于 n 的约束并添加更多约束$C_{n'}$ 而得到。也就是说， $n' =$simpl($\langle \mathcal{E}(n)[\sigma_{n'}],\ \mathcal{I}(n)[\sigma_{n'}] \cup C_{n'}\rangle$)。

下文中，我们固定任意一个 Nielsen 节点 n ，并针对 n 引入以下规则。

引理规则。 如果 $\mathcal{E}(n) \cup \mathcal{I}(n)$ 包含幂标记 u^m、字符串方程 $u \simeq v$ 或字符串变量 x，我们将相应的整数约束 $m \geq 0$、$|u| \simeq |v|$ 和 $|x| \geq 0$ 添加到 $\mathcal{I}(n)$。

项重写规则。 在下文中，我们用 $e_1 \rightsquigarrow e_2$ 表示在所考虑的 Nielsen 节点中，将 e_1 替换为 e_2。项重写规则用于简化长度约束并重写幂项，如下所示：

$$
\begin{array}{lll}
|uv| \rightsquigarrow |u| + |v| & |u^m| \rightsquigarrow m|u| & \varepsilon^m \rightsquigarrow \varepsilon \\
|\varepsilon| \rightsquigarrow 0 & |@| \rightsquigarrow 1 & v^m \rightsquigarrow \varepsilon, \text{ if } n \models m \simeq 0 \\
(v^{m_1})^{m_2} \rightsquigarrow v^{m_1 m_2} & v^{m_1} v^{m_2} \rightsquigarrow v^{m_1 + m_2} & v^m \rightsquigarrow v, \text{ if } n \models m \simeq 1
\end{array}
$$

若某项无法再应用任何重写规则，则称其为**完全重写的**。若一个（不）等式约束中的所有项均已完全重写，则称该约束**完全重写的**。

例 *2*（计算 simpl(n)，见例 *1*）。考虑例 1 中的两个输入方程，并令 n 为它们在相应 Nielsen 图中的对应节点 n。我们在 (E_1) 中删去公共后缀 b，并使用引理规则为每个字符串方程添加一个长度约束；这些约束可通过相应的项重写规则重写为完全重写的约束 $2|x_3| \simeq 3|x_5|$ 和 $2|x_5| \simeq |x_4|$。最后，我们为所有 $1 \leq i \leq 5$ 添加侧约束 $|x_i| \geq 0$。由此得到的约束集定义了 simpl(n)。

方程重写与生成规则。方程重写规则会移除字符串方程 $\mathcal{E}(n)$ 的某个前缀，有时还需满足额外的侧条件。如果不再有任何重写规则可应用于 n，则会触发针对 n 的生成规则，以添加额外信息；通常通过考虑多个情形来实现，并形成后继节点。

针对扩展字符串方程 $u \simeq v \in \mathcal{E}(n)$ 的方程重写/生成规则，是根据 u 和 v 的首个词元给出的。我们的重写规则和生成规则汇总于表 1，按照左侧和右侧首个词元（或 ε）的各种可能组合列出（表 1 的第 1–2 列）。表中的带撇号变量 x' 表示新字符串变量。若方程符合给定形式，且在当前 Nielsen 节点中满足侧条件，则可以应用相应规则。请注意，我们也可以将所有规则应用于 $v \simeq u$、$u^R \simeq v^R$ 和 $v^R \simeq u^R$。最后，任何经重写后变为 $\varepsilon \simeq \varepsilon$ 的方程都会从 $\mathcal{E}(n)$ 中移除；平凡满足的整数（不）等式也同样如此。

方程改写规则会将某个方程 $u \simeq v$ 替换为一个新方程，或检测到冲突 $\perp$（第 4 列），并且可能要求满足某些侧条件（第 3 列）。另一方面，生成规则会对 n 应用替换（第 5 列），以生成 n 的后继节点 n'，或向 n' 添加新的（整数）约束（第 6 列），从而使更多方程改写规则得以应用。

示例*3*（例 1 中*simpl()* 的生成规则）。考虑在例 1 中通过例 2 计算得到的简化节点 simpl(n)。适用的

LHS	RHS	重写规则		生成规则	
		侧条件	新方程	代换	新约束
tu	tv	–	$u \simeq v$	–	–
ε	av	–	$\bot$	–	–
ε	ov	–	$\bot$	–	–
ε	xv	–	–	x/ε	–
ε	$w^m v$	–	–	– –	$\{w \simeq \varepsilon\}$ $\{m \simeq 0\}$
au	bv	–	$\bot$	–	–
ou	$@v$	–	–	$o/@$	–
$^\dagger @u$	xv	–	–	x/ε $x/@x'$	– –
$^\ddagger @u$	$w^m v$	$n \models m \simeq 0$ $n \models m > 0$	$@u \simeq v$ $@u \simeq w w^{m-1} v$	– –	$\{m \simeq 0\}$ $\{m > 0\}$
xu	yv	–	–	x/ε y/ε y/xy' x/yx'	– $\{\|x\| > 0\}$ $\{\|x\| > 0\}$ $\{\|y\| > 0, \|x'\| > 0\}$
$^\dagger xu$	$w^m v$	–	–	$x/w^m x'$ $x/w^{m'} p_1$ $\cdots$, $x/w^{m'} p_k$	– $s_1 \cup \{0 \le m' < m\}$ $\cdots$ $s_k \cup \{0 \le m' < m\}$
$^\ddagger w^{m_1} u$	$w^{m_2} v$	$n \models m_1 \ge m_2$ $n \models m_1 < m_2$	$w^{m_1 - m_2} u \simeq v$ $u \simeq w^{m_2 - m_1} v$	– –	$\{m_1 \ge m_2\}$ $\{m_1 < m_2\}$
$^\ddagger w_1^{m_1} u$	$w_2^{m_2} v$	$n \models m_2 \simeq 0$ $n \models m_2 > 0$	$w_1^{m_1} u \simeq v$ $w_1^{m_1} u \simeq w_2 w_2^{m_2 - 1} v$	– –	$\{m_2 \simeq 0\}$ $\{m_2 > 0\}$

表 1. 扩展方程的方程改写规则和生成规则 $u \simeq v$。

生成规则由 x_3 和 x_5； x_5^R和 b； x_1 和 x_2；以及 x_3^R和 x_4^R 的左侧/右侧标记组合给出。

我们强调，使用表 1 中的规则构造的已扩展 Nielsen 图对应于一棵证明“树”，如下所示。

例*4*（ $xx \simeq yb$的完全展开的 *Nielsen* 图）。考虑普通字符串方程 $xx \simeq yb$。我们按照第3.1节所述并使用表 1 中的规则，构造其已扩展的 Nielsen 图，以此证明该方程可满足。 $xx \simeq yb$ 的完全展开的 Nielsen 图如图2所示。通过重用变量名称，可以使用一个节点，而不必使用多个同构节点。因此，
$x' \simeq \varepsilon \wedge |x'| \simeq 0$ 和 $x'' \simeq \varepsilon \wedge |x''| \simeq 0$ 可以收缩到同一个节点。

我们最后指出，表 1 中标注的规则
† 和 ‡ 需要特别谨慎处理，详见第5节。具体而言，当左侧出现 xu 、右侧出现 $w^m v$ 时，需要考虑由有限集 pre(w) $= \{\langle p_1, s_1 \rangle, \ldots, \langle p_k, s_k \rangle\}$ 给出的 $k+1$ 个分支；该集合表示 w 的所有语法前缀 p_i ，以及可能的侧条件 s_i。集合 pre(w)

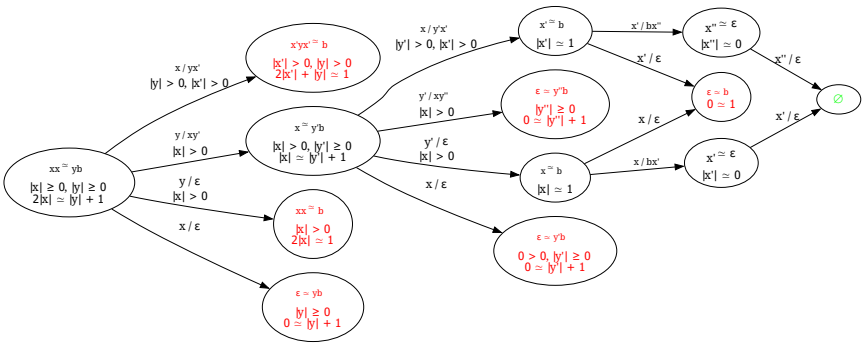

图 2. 普通字符串方程 $xx \simeq yb$ 的完全展开的 Nielsen 图，表示一种表 tableau 推导的变体。

详见第5节。我们通过使用更多已扩展规则（第4–6节）来优化这些规则的应用，下一节将对此进行讨论。

4 等式分解

本节针对某些无法使用第 3.2 节的 Nielsen变换规则求解字符串方程的情形，提出一种补救方法。直观而言，我们将字符串方程分解为子方程，再使用我们的变换规则进一步求解。请注意，表 1 的规则只能应用于字符串方程左侧/右侧的第一个（最后一个）词元。通过等式分解，我们将方程拆分成更小的方程。这样，等式分解便使我们能够将表 1 应用于左侧/右侧中位于首个词元以外位置的词元。

简而言之，我们按如下方式进行。考虑一个扩展字符串方程$u_1u_2 \simeq v_1v_2 \in \mathcal{E}(n)$，并设 $d = |u_1| - |v_1|$是一个已知整数常量。通过将方程拆分为两个方程来应用等式分解，并可能在其中一侧填充 d 个字符。若 $d = 0$，则方程 $u_1u_2 \simeq v_1v_2$ 可分解为由两个更小方程组成的集合 $\{u_1 \simeq v_1, u_2 \simeq v_2\}$ 。如果所有字符串项 u_1，u_2，v_1，v_2 都至少包含一个字符串变量或幂标记，我们就将 $\mathcal{E}(n)$中的 $u_1u_2 \simeq v_1v_2$ 替换为 $\{u_1 \simeq v_1, u_2 \simeq v_2\}$。例如，我们将$xayw \simeq ybxz$ 分解为 $xay \simeq ybx$ 和 $w \simeq z$，因为 $|xay| - |ybx| = 0$。

另一方面，如果 $d > 0$，那么分解 $u_1u_2 \simeq v_1v_2$ 还需要满足一个额外条件： u_1 的最后 d 个字符与 v_2的前 d 个字符相同。为此，我们引入 $|d|$ 个新的符号字符标记 $\bar{o_d}' = o'_1$，……，o'_d来表示 u_1的这些 d个字符。再次注意，这等价于引入一个新的字符串变量 x ，且满足 $|x| = d$。然后，$u_1u_2 \simeq v_1v_2$ 被分解为以下方程：

$$\{\ u_1 \simeq v_1\bar{o}'_d,\ \bar{o}'_d u_2 \simeq v_2\ \}.$$

因此，我们的等式分解方法使用长度约束 $|u_1| - |v_1|$，这些约束可以借助 n的整数约束 $\mathcal{I}(n)$来实现。我们的方法可以视为一种强化版的常规分解规则，不使用填充，主要用作其他求解器中的预处理规则：我们使用 d个符号字符标记对字符串方程进行填充，以便对其进行分解。

例 *5*（例 *1* 中的等式分解）。利用引理和例 2 中的项重写规则，我们推导出：

$$2|x_3| \simeq 3|x_5| \wedge 2|x_5| \simeq |x_4|.$$

因此，我们推断出 $d = |x_1x_1acx_2x_4x_2x_5| - |x_2x_2abcx_1x_1x_3x_3| = 1$。我们使用单个符号字符 o 对 (E_2) 进行分解，并将 (E_2) 替换为由此得到的两个新方程 (E_3) 和 (E_4)：

$$\begin{aligned} x_1x_1acx_2x_4x_2x_5o &\simeq x_2x_2abcx_1x_1x_3x_3, && (E_3)\\ x_3bax_5x_3x_4x_3 &\simeq ox_3x_4x_4ax_4 && (E_4) \end{aligned}$$

换言之，在 $\mathcal{E}(n)$ 中，(E_2) 被替换为 (E_3) 和 (E_4)；随后我们进一步利用引理和项重写规则化简 n 。

5 基项幂引言

我们引入幂项，以消除字符串方程中冗长且重复的子串，并将一系列重写和生成规则的应用压缩为幂项。幂项使我们能够在应用形式为 $x/@x'$ 的代换时，消除数量可能无界的方程分解；此处， x' 又是一个新字符串变量。例如，当 $xu = axv$ 可满足时，对于某个 $m \geq 0$，其所有模型都要求 x 为 a^m。若结合表 1 中的规则使用等式分解来寻找此类模型，那么对于某些不可满足方程，就需要分析无限多种 Token 组合。即使对于可满足的问题实例，所需的分析也可能非常漫长。为避免这种情况，我们使用基于以下性质的规则：若 $xu = vx$，且 $v \neq \varepsilon$，则对于某个自然数 m 以及字 w_1 和 w_2 [21]，有 $x = (w_1w_2)^m w_1$、$v = w_1w_2$ 和 $u = w_2w_1$ 。

幂引入和规则 † 见表 1。 </style}表 1 中由 † 标注的规则仅在幂引入不适用时才应用。在这种情况下，给定一个基字符串项 $w \neq \varepsilon$ 和约束 $xu \simeq wxv$，我们有：

$$\exists m.\ \bigvee_{\langle p,s\rangle \in \mathrm{pre}(w)} \left(x \simeq w^m p \wedge \bigwedge s \right) \vee w \simeq \varepsilon \tag{1}$$

其中 pre(w) 表示 w 的可能真前缀 p 的配对集合 $\langle p, s\rangle$ ，使得仅存在一个有限集 s 的侧条件来约束 p。形式上，pre(w) 对基字符串项 w 定义为：

$-\,\mathsf{pre}(uv) := \mathsf{pre}(u) \cup \{\langle up, s\rangle \mid \langle p, s\rangle \in \mathsf{pre}(v)\}$,$-\,\mathsf{pre}(\varepsilon) := \emptyset$,$-\,\mathsf{pre}(@) := \{\langle \varepsilon, \emptyset\rangle\}$, $-\,\mathsf{pre}(u^m) := \{\langle u^{m'}p, \{0 \le m' < m\} \cup s\rangle \mid \langle p, s\rangle \in \mathsf{pre}(u)\}$

其中 m' 是一个新的整数变量。

让我们观察一下幂引入与前缀 $\mathsf{pre}(w)$ 分析之间的相互作用。当考虑方程 $xu \simeq wxv$ 时，如果在某个潜在模型中 w 表示 ε，可能会出现不可靠性问题，因为 w 会被约掉。当 w 由幂标记组成，且它们的指数均为零，因此都变成 ε 时，就可能出现这种情况。根据 $\mathsf{pre}(w)$ 的定义，我们会引入一个新的 m' 整数变量，使其满足 $0 \le m' < m$；当 $m = 0$ 时，这会导致冲突，从而错误地排除一个模型。作为补救措施，我们还会显式地对 $w \simeq \varepsilon$ 进行分情况处理，以涵盖这种情况。另一方面，如果 w 不是 ε，幂引入便可以借助整数约束替换字符串变量 x，从而将其消去，如下所示。

例 *6*（$xbxa \simeq axbx$）。如果不引入幂，应用表 1 中的规则将无法求解不可满足的方程 $xbxa \simeq axbx$。应用表 1 中的规则会产生一个无限的代换链，其形式为 x/ax'，并导致字符串项越来越长。通过引入幂项，我们可以通过一次代换 x/a^m 消去 x：将此代换应用于 $xbxa \simeq axbx$ 后得到 $a^mba^ma \simeq aa^mba^m$，而它可以很容易地化简为 $\bot$。

幂引入及规则 ‡ 表 *1* 的内容。接下来，我们将说明如何结合使用幂引入与表 1 中标注的 Nielsen变换规则 ‡。当某个方程一侧的首个 Token 是幂标记时，我们首先检查能否重写方程另一侧，使其以底相同的幂标记开头，从而将二者抵消。为此，我们应用以下额外的项重写规则：

$$\begin{array}{rcrcrcl} ww^m & \rightsquigarrow & w^{m+1} & & w_2\,(w_1w_2)^m & \rightsquigarrow & (w_2w_1)^m\,w_2 \\ w^mw & \rightsquigarrow & w^{m+1} & & (w_1w_2)^m\,w_1 & \rightsquigarrow & w_1\,(w_2w_1)^m \\ ww & \rightsquigarrow & w^2 & & & & \end{array}$$

这些额外规则大大减少了引入幂时所需的方程分解次数。更重要的是，它们是处理形如 $a(ba)^mu \simeq (ab)^mv$ 的方程所必需的。如果求解器没有发现这些前缀可以重写为相同的幂，那么表 1 的规则就会建议针对 m 的每个可能取值进行显式展开。

增强幂归约。我们将规则 (1) 从 $xu \simeq wxv$ 推广到适用于由下式给出的方程集合 $EQ \subseteq \mathcal{E}(n)$：

$$EQ := \{\; w_1x_ku_1 \simeq x_1v_1,\; w_2x_1u_2 \simeq x_2v_2,\; \ldots,\; w_kx_{k-1}u_k \simeq x_kv_k \;\},$$

其中 $k \geq 1$，且所有项 $w_1, \ldots, w_k$ 均为基项。与 (1) 类似，集合 EQ强制采用更一般形式的幂引入

$$\exists m. \bigvee_{1 \leq i \leq k} \bigvee_{\langle p,s \rangle \in \mathrm{pre}(\bar{w}_i)} \left(x_i \simeq \bar{w}_i^m p \wedge \bigwedge s \right) \vee \bar{w}_1 \simeq \varepsilon, \tag{2}$$

其中 $\bar{w}_i := w_i w_{i-1} \ldots w_1 w_k \ldots w_{i+1}$。

例 7（例 5 中的幂引入）。在将 (E_2) 分解为 (E_3) 和 (E_4) 之后，我们引入一个由 (E_4) 的前缀所证明的幂标记：(E_4) 的左侧以 x_3 开头，右侧以 ox_3 开头。由于仅由符号字符 o 构成的字符串项是基项且不能为 ε，我们应用幂引入，并只考虑一个后继节点作为 $\mathrm{pre}(o) = \{\langle \varepsilon, \emptyset \rangle\}$：该后继由 x_3/o^{m_1} 针对某个新常量 m_1 生成。根据一条引理规则，我们得到 $m_1 \geq 0$，并将长度约束 $|x_3[x_3/o^{m_1}]|$ 改写为 m_1。字符串方程 (E_1)、(E_3) 和 (E_4) 分别变为：

$$\begin{aligned}
o^{2m_1} x_4 b x_5 &\simeq x_5 x_5 x_5 x_5 x_4 b, && (E_1') \\
x_1 x_1 a c x_2 x_4 x_2 x_5 o &\simeq x_2 x_2 a b c x_1 x_1 o^{2m_1}, && (E_3') \\
o^{m_1} b a x_5 o^{m_1} x_4 o^{m_1} &\simeq o o^{m_1} x_4 x_4 a x_4. && (E_4')
\end{aligned}$$

得益于本节前面引入的附加重写规则，可以将（E_4'）右侧的 Tokenoo^{m_1} 重写为 o^{m_1+1}。因此，（E_3'）中的公共前缀被移除，接着应用 o/b。得到（E_1'）、（E_3'）和（E_4'）对应的方程如下：

$$\begin{aligned}
b^{2m_1} x_4 b x_5 &\simeq x_5 x_5 x_5 x_5 x_4 b, && (E_1'') \\
x_1 x_1 a c x_2 x_4 x_2 x_5 b &\simeq x_2 x_2 a b c x_1 x_1 b^{2m_1}, && (E_3'') \\
a x_5 b^{m_1} x_4 b^{m_1} &\simeq x_4 x_4 a x_4. && (E_4'')
\end{aligned}$$

可以进一步引入幂：既可以因左侧的后缀$x_4 b^{m_1}$ 和右侧的 x_4 而使用（E_4''），也可以利用方程（E_1''）和（E_4''）通过幂引入的广义形式（2）来引入。在后一种情况下，我们会得到以下四个后继节点：

$$\left\{ x_4/(ab^{2m_1})^{m_2},\ x_4/(ab^{2m_1})^{m_2} a b^{m_3},\ x_5/(b^{2m_1} a)^{m_2} b^{m_3},\ x_5/(b^{2m_1} a)^{m_2} b^{2m_1} \right\}$$

以及 $m_3 < 2m_1$。在前一种情况下，有两种处理方式：应用 $x_4/b^{m_3}(b^{m_1})^{m_2}$并令 $m_3 < m_1$ ——这将简化为 $b^{m_1 m_2 + m_3}$ ——或者添加 $b^{m_1} \simeq \varepsilon$。我们选择前一种方式，并首先考察第一个后继节点：应用 $x_4/b^{m_3}(b^{m_1})^{m_2}$。

$$\begin{aligned}
b^{2m_1 + m_1 m_2 + m_3} b x_5 &\simeq x_5 x_5 x_5 x_5 b^{m_1 m_2 + m_3} b, && (E_1''') \\
x_1 x_1 a c x_2 b^{m_1 m_2 + m_3} x_2 x_5 b &\simeq x_2 x_2 a b c x_1 x_1 b^{2m_1}, && (E_3''') \\
a x_5 b^{2m_1} &\simeq b^{2m_1 m_2 + 2m_3} a. && (E_4''')
\end{aligned}$$

由于 $0 \leq m_3 < m_1$，我们推导出 $m_1 > 0$，因此在 (E_4''') 中，使用表 1 中相应的规则，将 b^{2m_1} 展开为 $b^{2m_1 - 1} b$，从而导致冲突。因此，我们

考虑通过添加约束 $b^{m_1} \simeq \varepsilon$ 所生成的后继节点。这意味着 $m_1 = 0$，消除了所有幂标记，使我们能够推导出 $|x_4| = |x_5| = 0$。经过进一步简化后，我们推导出 x_4/ε 和 x_5/ε。因此，最终得到包含以下单个普通字符串方程的 Nielsen 节点：

$$x_1x_1acx_2x_2b \simeq x_2x_2abcx_1x_1.$$

6 Parikh像

Parikh像 [25] 可以是一种有效方法，通过提取关于终结字符 [31]出现次数的约束来证明字符串约束集合不可满足。它表示字符串方程的一种抽象，略去字符位置信息，仅保留出现次数。接下来，我们建立一种保留字符相对位置信息的方法，从而得到更紧的抽象。我们的方法计算一个我们称为（误差有界）多序列*Parikh*像的值。

为了形式化Parikh像，令 $\pi_a(u)$为一个函数，表示存在 $a \in \Sigma$ 于 u中的位置集合。集合 $\{ (a, \pi_a(u)) \mid a \in \Sigma \}$是 u的完整表示。令 $\alpha_a(u) := |\pi_a(u)|$，则方程 $u \simeq v$可由$\bigwedge_{a\in\Sigma} \alpha_a(u) \simeq \alpha_a(v)$过近似。例如，$\alpha_a(xay) = \alpha_a(x) + 1 + \alpha_a(y)$，且 $\alpha_a(xby) = \alpha_a(x) + \alpha_a(y)$，因此方程 $xay \simeq ybx$ 可立即被识别为不可满足。我们注意到，无论是等式分解，还是与表 1的规则相结合的基项幂引入，都无法将该方程检测为不可满足。尽管如此，基于单个具体字符的Parikh约束仍无法捕捉某些方程的不可行性，例如我们运行示例中唯一Nielsen 节点内的方程：$x_1x_1acx_2x_2b \simeq x_2x_2abcx_1x_1$。我们的方法将完成这一点。

6.1 无边框模式的 Parikh像

接下来，我们考虑长度大于 1 的字符串 $w \in \Sigma^+$ 的 Parikh像——我们称之为 w 模式 ——并假设给定的是纯字符串方程。就本文的讨论范围而言，在计算 Parikh像时，我们将符号字符和幂标记视为字符串变量。例如，在纯字符串方程 $xabcy \simeq ybacx$ 中，模式 bc 在方程左侧出现 $\alpha_{bc}(xabcy) = \alpha_{bc}(x) + 1 + \alpha_{bc}(y)$ 次，在右侧出现 $\alpha_{bc}(ybacx) = \alpha_{bc}(y) + \alpha_{bc}(x)$ 次。因此，该方程不可满足。我们的贯穿示例也提供了一个使用此类模式的机会：

例*8*（例 *7* 中的 *Parikh*像）。考虑我们贯穿全文的例 7 中的方程。我们可以考虑模式 $w = abc$，从而得到 $\alpha_w(x_1x_1acx_2x_2b) \simeq \alpha_w(x_2x_2abcx_1x_1)$。直观地，我们可以将这个方程重写为 $\alpha_w(x_1x_1) + \alpha_w(x_2x_2) \simeq \alpha_w(x_2x_2) + 1 + \alpha_w(x_1x_1)$，进而得到 $0 = 1$，这证明了该方程本身以及贯穿全文的整体示例均不可满足，因为根节点的所有后继 Nielsen 节点都不一致。接下来，我们将正式定义这种“直观重写”。

与通常针对单字符的 Parikh像不同，对于 $\alpha_w(u)$，不存在一种通用的定义，能够将其分解为 u 中字符串常量和字符串变量上的部分。例如，我们无法将 $\alpha_{ab}(ybacx)$ 分解为 $ybacx$ 各个组成部分之和，因为其取值取决于 x 是否以 a 结尾。

我们通过以下方式解决这一局限：

1. 将 Parikh 像限制为字符序列模式 w，使得 w 的任何真后缀都不是 w 的前缀。这类字称为无边框的。2. 定义上近似 $\alpha^{\uparrow}_w(u)$ 和下近似 $\alpha^{\downarrow}_w(u)$，使得当 $\alpha^{\uparrow}_w(u) - \alpha^{\downarrow}_w(v)$ 或对称地，当 $\alpha^{\uparrow}_w(v) - \alpha^{\downarrow}_w(u)$ 重写为负常数时，即可判定字符串方程 $u \simeq v$ 无解。

我们针对 $\alpha^{\uparrow}_w(u)$ 和 $\alpha^{\downarrow}_w(u)$ 的重写规则满足以下性质：

引理 1（无边框模式的 Parikh 像正确性）。假设 w 是一个无边框模式。

– 如果 $u \in \Sigma^*$，那么 $\alpha^{\uparrow}_w(u) = \alpha^{\downarrow}_w(u) =$ 在 u 中出现的次数。– 假设 $\alpha^{\uparrow}_w(u)$ 针对自然数 c_x 和 k 重写为 $k + \sum_x c_x\alpha_w(x)$。那么，对于从字符串变量到 Σ^* 的每个代换 σ，$\alpha^{\uparrow}_w(u[\sigma]) \leq k + \sum_x c_x\alpha^{\uparrow}_w(x[\sigma])$。对 $\alpha^{\downarrow}_w(u)$ 也成立对称的性质。

接下来需要定义上近似与下近似。正如示例所示，对无边界模式的限制意味着存在许多上近似与下近似相同的情况。

定义 2（d-间隙和 $w \bowtie u$）。

d-间隙 我们称字符串项 u 为一个 d-间隙（带有 $d > 1$），当且仅当它属于以下形式之一：$u = xvy$（带有 $v \in \Sigma^*$）、$u = xv$，或 $u = vy$（后两种情况下为 $v \in \Sigma^+$），且 $0 < ||u|| \leq d$。

$w \bowtie u$ 如果 w 是一个模式，且 u 是一个 $|w|$-间隙，我们记作 $w \bowtie u$，表示 w 在 $|w|$-间隙 u 内可能存在交叉出现。形式上，若 $u = xw_2y$ 且我们可以将 $w = w_1w_2w_3$（$\{w_1, w_3\} \neq \varepsilon$）分解，则 $w \bowtie u$ 为真；否则为假。类似地，$u = w_1y$ 的情况要求 $w = w_1w_2$（$w_2 \neq \varepsilon$），而在 $u = xw_2$ 的情况下要求 $w = w_1w_2$（$w_1 \neq \varepsilon$）

利用间隙的定义，应用以下项重写规则，逐步将 $\alpha^{\downarrow}_w(u)/\alpha^{\uparrow}_w(u)$ 重写为无边界模式 w 所需的分组形式：

$$
\begin{aligned}
\alpha^{\updownarrow}_w(x) &\rightsquigarrow \alpha_w(x) && \\
\alpha^{\updownarrow}_w(w_2) &\rightsquigarrow 0 && \text{if } w_2 \in \Sigma^* \wedge |w_2| < |w| \\
\alpha^{\updownarrow}_w(uwv) &\rightsquigarrow 1 + \alpha^{\updownarrow}_w(u) + \alpha^{\updownarrow}_w(v) && \\
\alpha^{\updownarrow}_w(ua_1w_2a_2v) &\rightsquigarrow \alpha^{\updownarrow}_w(ua_1w_2) + \alpha^{\updownarrow}_w(w_2a_2v) && \text{if } a_1w_2a_2 \in \Sigma^+ \wedge \\
& && \quad |a_1w_2a_2| = |w| \wedge a_1w_2a_2 \neq w \\
\alpha^{\updownarrow}_w(uxw_2yv) &\rightsquigarrow \alpha^{\updownarrow}_w(uxw_2) + \alpha^{\updownarrow}_w(w_2yv) && \text{if } xw_2y \text{ is } |w|\text{-gap and } w \not\bowtie xw_2y \\
\alpha^{\updownarrow}_w(aw_2yv) &\rightsquigarrow \alpha^{\updownarrow}_w(w_2yv) && \text{if } aw_2y \text{ is } |w|\text{-gap and } w \not\bowtie aw_2y
\end{aligned}
$$

$$\alpha^{\updownarrow}_w(uxw_2a) \rightsquigarrow \alpha^{\updownarrow}_w(uxw_2) \qquad \text{如果 } xw_2a \text{ 是 } |w|\text{-间隙且 } w \not\bowtie xw_2a$$

现在，我们为先前精确重写规则未涵盖的情况定义下近似和上近似。它们是：

$$\begin{array}{lll}
\alpha^{\downarrow}_w(x) \rightsquigarrow \alpha_w(x) & \alpha^{\uparrow}_w(x) \rightsquigarrow \alpha_w(x) & \\
\alpha^{\downarrow}_w(w_2u) \rightsquigarrow \alpha^{\downarrow}_w(u) & \alpha^{\uparrow}_w(w_2xu) \rightsquigarrow 1 + \alpha^{\uparrow}_w(xu) & \text{if } w_2 \in \Sigma^+ \\
\alpha^{\downarrow}_w(xw_2) \rightsquigarrow \alpha_w(x) & \alpha^{\uparrow}_w(xw_2) \rightsquigarrow 1 + \alpha_w(x) & \text{if } w_2 \in \Sigma^+ \\
 & \alpha^{\uparrow}_w(xw_2yv) \rightsquigarrow 1 + \alpha_w(x) + \alpha^{\uparrow}_w(yv) & \text{if } w_2 \in \Sigma^*
\end{array}$$

重写规则的正确性依赖于先穷尽应用非近似规则，然后再考虑近似所呈现的情形。通俗地说，我们要求剩余的 $\alpha_w(u)$ 中所有 u 都是“$|w|$-间隙的串接”，且其中可以包含交叉出现。

我们用于筛选不可满足方程 $u \simeq v$ 的过程如下：**(i)** 枚举在 u 和 v 中出现的极大无边界模式 $w \in \Sigma^*$。这些模式仅相对于所考虑一侧 u 或 v 中的模式而言是极大的。**(ii)** 对于其中每个 w，将重写 $\alpha^{\uparrow}_w(u) - \alpha^{\downarrow}_w(v)$ 为总和 $k + \sum_x c_x\alpha_w(x)$。如果每个 c_x 都是 0 且 $k < 0$，我们便可以断定方程 $u \simeq v$ 不可满足。

例 *9*（$xaxaabbby \simeq xyabababx$）。考虑无边界模式 $w = ab$，它在右侧中是极大的。于是，$\alpha^{\uparrow}_w(xaxaabbby) \rightsquigarrow 2\alpha_w(x) + 2 + \alpha_w(y)$ 和 $\alpha^{\downarrow}_w(xyabababx)$ 可改写为 $\alpha_w(y) + 3 + 2\alpha_w(x)$。因此，$\alpha^{\updownarrow}_w(xaxaabbby) - \alpha^{\downarrow}_w(xyabababx)$ 是 -1；这证明该方程不可满足。

7 实现与实验

实现。我们在一个新的原型 ZIPT 中实现了我们的方法。我们的实现[3] 使用 Z 3 [24] SMT 求解器，既用于辅助整数求解器，也用于通用的 CDCL($\mathcal{T}$) 框架，并通过用户传播 [8] 调用我们的字符串求解过程，处理普通字符串方程集合。

我们的实现遵循图 1 所示的工作流程，结合使用表 1 中的 Nielsen 变换规则、等式分解 4、幂引入 5 和 Parikh 图像 6。如果表 1 中有多条生成规则适用，我们会选择预计能更快终止的规则。例如，优先选择消除字符串变量的规则，而非不会消除字符串变量的规则。

除了所介绍的生成规则外，我们还使用前瞻启发式方法，优先选择这样的生成规则：除了一个生成的后继节点外，其余所有节点都会导致冲突。此类情况的示例包括：

– 消去某个字符串变量 x，使用 x/ε，其中 $xu_1v_1 \simeq u_2v_2$，且$u_1 \in (\{x\} \cup \mathcal{V}_C \cup \Sigma)^+$ 和 $u_2 \in (\mathcal{V}_C \cup \Sigma)^+$。除非可以应用幂引入规则，否则我们可以计算 u_2 的最长前缀 w，使得

[3] 可通过 `https://github.com/CEisenhofer/ZIPT` 获取

	ZIPT	Z3	cvc5	OSTRICH	Z3-Noodler	Z3str3	总计	
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	

表 2. 使用 SMT-LIB 的 woorpje 基准测试集中的字符串求解基准测试，在四个赛道上求解的问题数量。第 8 列列出了每个赛道的问题总数。我们将求解器 ZIPT 与相关方法 Z3 [24], cvc5 [3], Ostrich、 [9], Z3-Noodler [10], 和 Z3str3 [6]进行了比较。

我们可以安全地应用 x/wx'。例如， $xxbau \simeq abv$ 会直接得到 x/abx' ，无需分支，因为 x/ε 和 x/a 都会立即导致冲突。如果 x 是幂标记，也适用类似情况；– 展开幂标记 $w_1^m u \simeq w_2 v$ ，其中 $w_2 \in (\mathcal{V}_C \cup \Sigma)^+$。每当 w_2不可能是至少展开一次 w_1^m后得到的任何单词的前缀或后缀时，我们就得出结论 $m = 0$。例如，$(ab^{m_2}c)^{m_1}u = aav$ 要求$m_1 = 0$ ，因为对于任意 m_2 ， aa 都不是 $ab^{m_2}c$ 的前缀，也不是其后缀；– 长度约束：在当前 Nielsen 节点 n 中，我们可以推导出 $n \models |x| > |y|$ 。如果考虑 $xu \simeq yv$，就必须应用 x/yx'。类似地，如果 $n \models\!\not| x| = |y|$，我们选择了 x/y。

遍历和扩展 Nielsen 图时，我们采用了一种迭代加深的变体：对于节点 n 的后继，如果它们严格减少了n 的约束，就会以更深的层级进行扩展。

实验设置。 因此，我们在 SMT-LIB [4]的 woorpje 基准测试集 [12] 的四个 QF_S 赛道上运行求解器 ZIPT；该基准测试集包含 409 个仅含字符串方程的基准测试文件。我们将本方法与最近一届 SMT-COMP 竞赛[4]中参赛的最先进的求解器进行了比较。我们采用 10秒超时、8 GB内存和默认的求解器配置，并使用 Intel i7-13850HX 处理器的专用核心。
实验分析。 我们的结果汇总于表 2，展示了 ZIPT 优于当前最先进的方法。在求解 track 02 中的问题时，我们的方法得益于幂引入；这是因为这些基准所描述的模型，其规模随字符串变量的数量呈指数增长。使用（嵌套）幂标记，我们无需耗费指数级的步数就能求解此类示例，这是我们与其他工作的一个关键区别。

8 相关工作

将字符串方程分解为子序列 通常主要用作字符串求解器中的预处理步骤。尽管这种方法主要只应用于不需要填充的情况，但此前关于字符串求解的研究已详细介绍了我们填充方法的一种变体 [28]。

[4] https://smt-comp.github.io/2024/

识别字符串变量中的自依赖，尤其是通过显式幂进行识别，已在 [16,19]中得到研究。虽然这类方程所能定义的语言通常属于 EDT0L [11], ，但我们所考虑的显式幂属于更易处理的子类。关于表达特定幂项的字方程，理论研究还要追溯到更早以前 [21]。就求解器支持而言， [20] 的方法能够使用正则表达式处理某些自依赖。相比之下，我们识别出一个范围更广的自依赖类别，并引入显式幂，以支持嵌套幂，从而提高效率（见第 7 节的 `track 02` ）。

*Parikh*像通过正则表达式成员关系改进字符串推理 [9,31]。大多数技术使用单字符Parikh像来检测平凡的不可满足性。我们通过同时考虑多个字符序列上的Parikh信息，对此进行了推广。Parikh 矩阵 [32] 等相关概念及其在可判定性中的作用已在 [13]中得到研究。我们对潜在交叉出现的分析方法与重压缩 [18], 密切相关，在重压缩中，这类出现通过情形区分逐步消除。

9 结论

我们提出了一种字符串求解方法，该方法基于Nielsen变换规则，并结合了等式分解、幂引入和Parikh像。初步结果表明，我们的工作具有实际应用潜力。作为后续工作，我们计划通过使用更紧的误差近似，并关联多个方程中的Parikh信息，而不是孤立地考虑每个方程，来改进基于Parikh像的分析。其他工作还包括：基于依赖性分析，利用辅助变量对方程进行启发式拆分；以及引入非基幂项，以表达当前无法在我们的演算中表示的依赖关系。此外，支持其他与字符串相关的SMT-LIB函数（包括正则表达式）也是下一步工作的重点，这些函数由SMT-LIB标准 [4], 定义。

致谢。 本研究全部或部分由ERC整合者基金ARTIST 101002685、ERC概念验证基金LEARN 101213411、维也纳工业大学博士学院SecInt、FWF SpyCoDe基金10.55776/F85、WWTF ForSmart基金10.47379/ICT22007以及亚马逊研究奖2023 QuAT资助。

利益披露。 作者声明不存在与本文内容相关的利益冲突。

参考文献

1. Abdulla, P.A., Atig, M.F., Chen, Y., Holík, L., Rezine, A., Rümmer, P., Stenman, J.：用于验证的字符串约束。载于：CAV。LNCS，第8559卷，第150–166页。施普林格（2014）。 https://doi.org/10.1007/978-3-319-08867-9_10

2. Amadini, R.：字符串约束求解综述。《ACM计算调查》**55**(2), 16 1 16 38 2023 https://doi.org/10.1145/3484198 : – : ( )。Barbosa, H., Barrett, C. 3 W., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mohamed, A., Mohamed, M., Niemetz, A., Nötzli, A., Ozdemir, A., Preiner, M., Reynolds, A., Sheng, Y., Tinelli, C., Zohar, Y.：cvc5：一款多功能且适用于工业级应用的可满足性模理论求解器。载于：TACAS。LNCS。第13243卷，第415–442页。施普林格（2022）。4 https://doi.org/10.1007/978-3-030-99524-9_24 。

Barrett, C., Fontaine, P., Tinelli, C.：《SMT-LIB标准：2.7版》。技术报告，爱荷华大学计算机科学系（2025） 5. Barrett, C.W., Sebastiani, R., Seshia, S.A., Tinelli, C.：可满足性模理论。载于：《可满足性手册》第二版。《人工智能与应用前沿》。第336卷，第1267–1329页。IOS出版社（2021）。 https 6 //doi.org/10.3233/FAIA201017 。Berzish, M., Ganesh, V., Zheng, Y.：

Z3str3：采用理论感知启发式的字符串求解器。载于：FMCAD。第55–59页。IEEE（2017）。 https://doi.org/10.23919/FMCAD.2017.8102247. Berzish, M., Kulczynski, M., Mora, F., Manea, F., Day, J.D., Nowotka, D., Ganesh, V.：一种用于正则表达式以及字符串长度上的线性算术的可满足性模理论求解器。载于：CAV。LNCS。第12760卷，第289–312页。施普林格（2021）。 https://doi.org/10 .8 1007/978-3-030-81688-9_14 。Bjørner, N.S., Eisenhofer, C.,

Kovács, L.：Z3中自定义理论的可满足性模理论。载于：VMCAI。LNCS。第13881卷，第91–105页。施普林格（2023）。 https://doi.org/9 10.1007/978-3-031-24950-1_5 。Chen, T.,

Hague, M., Lin, A.W., Rümmer, P., Wu, Z.：用于判定具有复杂操作的字符串操作程序路径可行性的过程。《ACM 程序设计语言》 **3** 49 1 49 30 2019 https://doi.org/10.1145/3290362 (POPL), : – 10 : ( )。Chen, Y., Chocholatý, D., Havlena, V., Holík, L., Lengál, O., Síc, J.：Z-noodler：一种基于 3 自动机的字符串求解器。载于：TACAS。LNCS，第14570卷，第24–33页。施普林格（ 2024 https://doi.org/10.1007/978-3-031-57246-3_2 ）。Ciobanu, L., Diekert, V., 11 Elder, M.：自由群上方程的解集是EDT0L语言。《国际代数与计算杂志》**26**(5), 843–886 (2016)。 https 12 //doi.org/10.1142/S0218196716500363 。Day, J.D., Ehlers, T., Kulczynski, M.,

Manea, F., Nowotka, D., Poulsen, D.B.：使用SAT求解字方程。载于：RP。LNCS，第11674卷，第93–106页。施普林格（ 2019 https://doi.org/10.1007/978-3-030-30806-3_8 ）。 13 Day, J.D., Ganesh, V., He, P., Manea, F., Nowotka, D.：字方程的可满足性：可判定与不可判定的理论。载于：RP。LNC，第11123卷，第15–29页。施普林格（ 2018 https://doi.org/10.1007/978-3-030-00250-3_2 ）。Emmi, M., 14 Majumdar, R., Sen, K.：数据库应用的动态测试输入生成。载于：ISSTA。第151–162页。ACM（2007）。 https://doi.org/10.1145/1273463.1273484 15. Eriksson, B., Stjerna, A., Masellis, R.D., Rümmer, P., Sabelfeld, A.：Black Ostrich：使用字符串求解器扫描Web应用。载于：SIGSAC。第549–563页。ACM（ 2023 https://doi.org/10.1145/3576915.3616582 ）。 16 Ganesh, V., Minnes, M., Solar-Lezama, A., Rinard, M.C.：带长度约束的字方程：哪些是可判定的？载于：HVC。LNCS，第7857卷，第209–226页。施普林格（ 2012 https://doi.org/10.1007/978-3-642-39611-3_21 ）。Hague, M.： 17 MOSCA中的字符串。ACM SIGLOG News ( ), – (2019)。 https **6** 4 4 22 //doi.org/10.1145/3373394.3373396 。Jez, A.：重压缩：一种简单而强大的字方程技术。 18

《ACM杂志》**63**(1), 4:1–4:51 (2016)。 https://doi.org/10.1145/2743014

19. Karhumäki, J., Mignosi, F., Plandowski, W.：字方程对语言和关系的表达能力。《ACM 学报》**47**(3), 483–505 (2000) 20. Liang, T., Reynolds, A., Tsiskaridze, N., Tinelli, C., Barrett, C.W., Deters, M.：一种高效的字符串约束可满足性模理论求解器。Formal Methods Syst. Des. **48**(3), 206 234 2016 https://doi.org/10.1007/S10703-016-0247-6 – ( ). . Lyndon, R.C., 21 Schützenberger, M.P., 等：自由群中的方程 $a^M = b^N c^P$。Michigan Math. J **9**(4), 289–298 (1962) 22. Makanin, G.S.：自由半群中方程可解性问题。Matem- aticheskii Sbornik **145**(2), 147–236 (1977) 23. Mora, F., Berzish, M., Kulczynski, M., Nowotka, D., Ganesh, V.：Z3str4：一种多臂字符串求解器。载于：FM。LNCS，第 13047 卷，第 389–406 页。施普林格 (2021)。 https://doi.org/10.1007/978-3-030-90870-6_21 24 3 . de Moura, L.M., Bjørner, N.S.：Z：一种高效的可满足性模理论求解器。载于：TACAS。LNCS，第 4963 卷，第 337–340 页。施普林格 (2008)。 https://doi.org/10.1007/978-3-540-78800-3 24_25. Parikh, R.：论上下文无关语言。J. ACM**13**(4), 570–581 (1966)。 https: //doi.org/10.1145/321356.321364 https://doi.org/10.1145/321356.321364 , . 26 Przybocki, B., Barrett, C.W.：应用于带长度约束字方程的 Nielsen 变换的终止性。CoRR (2025)。 https://doi.org/10.48550/ARXIV.2501.11789 27. Reynolds, A., Nötzli, A., Barrett, C.W., Tinelli, C.：用于简化扩展字符串约束的高层抽象（可满足性模理论）。载于：CAV。LNCS，第 11562 卷，第 23 42 2019 https://doi.org/10.1007/978-3-030-25543-5_2 – 28 页。施普林格 ( )。 . Reynolds, A., Nötzli, A., Barrett, C.W., Tinelli, C.：重新审视字符串和正则表达式的归约，载于：FMCAD，第 225–235 页。IEEE (2020)。 https:// doi.org/10.34727/2020/isbn.978-3-85448-042-6_30 29 . Rümmer, P.：带线性整数算术的一阶逻辑的约束序列演算。载于：LPAR。LNCS，第 5330 卷，第 274–289 页。施普林格 (2008) 30. Stanford, C., Veanes, M., Bjørner, N.S.：用于高效求解扩展正则表达式约束的符号布尔导数。载于：PLDI。第 620–635 页。ACM (2021 https://doi.org/10.1145/3453483.3454066)。 . Stjerna, A., Rümmer, P.： 31 一种求解正则语言 Parikh 像的约束方法。《ACM 程序设计语言》**8**(OOPSLA1), 1235–1263 (2024)。 https ://doi.org/10.1145/3649855 32 . Subramanian, K.G., Huey, A.M., Nagar, A.K.：论 Parikh 矩阵。Int. J. Found. Comput. Sci. **20**(2), 211–219 (2009)。 https://doi.org/10.1142/S0129054109006528 33. Trinh, M., Chu, D., Jaffar, J.：S3：一种用于检测 Web 应用程序漏洞的符号字符串求解器。载于：SIGSAC。第 1232–1243 页。ACM (2014)。 https ://doi.org/10.1145/2660267.2660372 34 . Wassermann, G., Su, Z.：对 Web 应用程序注入漏洞进行准确且可靠的分析。载于：SIGPLAN。第 32–41 页。ACM (2007)。 https://doi.org/10.1145/1250734.1250739