arXiv:2303.01142v1 [cs.LO] 2 Mar 2023

带扩展的词约束求解中分情况规则的符号算法（技术报告）

Yu-Fang Chen[a]、Vojtˇech Havlena[b]、Ondˇrej Leng´al[b,*]、Andrea Turrini[c,d]

[a]*Institute of Information Science, Academia Sinica, 128 Academia Road, Section 2, Nangang, 11500, Taipei, Taiwan*[b]*Faculty of Information Technology, Brno University of Technology, Bozetechova 2, 61200, Brno, Czech Republic*
[c]*State Key Laboratory of Computer Science, Institute of Software, Chinese Academy ofSciences, Haidian District, Zhongguancun 4# South Fourth Street, 100190, Beijing, China*
[d]*Institute of Intelligent Software, 221, Nansha Street West, 511458, Guangzhou, China*

摘要

分情况是当前字符串约束理论判定过程中的一项核心证明规则。它的使用是字符串约束求解中状态空间爆炸的主要原因，因为它是唯一会在证明树中创建分支的规则。此外，显式处理分情况规则可能导致在证明树的多个分支中重复计算相同的任务。本文提出了一种符号算法，可显著减少此类冗余。具体而言，我们将字符串约束编码为正则语言，并将证明规则编码为有理变换器。这使我们能够仅执行一次证明树中的相似步骤，从而缓解状态空间爆炸。我们还扩展了该编码，以处理字符串约束、长度约束和正则约束的任意布尔组合。实验结果验证了，我们的技术在许多实际情形中均有效，而其他最先进的求解器无法给出答案；我们的 Python 原型实现求解了超过 50 % 的其他工具无法求解的字符串约束。

关键词：字符串约束、理论模可满足性、正则模型检测、尼尔森变换、有限自动机、字符串上的单子二阶逻辑

1. 引言

约束求解是形式化验证和分析许多领域中作为使能技术使用的一种技术，例如符号执行（Cadar 等人（2006）；Godefroid 等人（2005）；King（1976）；Sen 等人（2013））、静态分析（Wang 等人（2017）；Gulwani 等人（2008））或程序综合（Gulwani 等人（2011）；Osera（2019）；Knoth 等人（2019））。例如，在符号执行中，通过创建一个编码给定路径上程序中变量值演化的约束，并检查该约束是否可满足，来测试一条路径的可行性。

*通讯作者

由于被分析程序所使用的特性，检查该约束的可满足性可能是一项复杂的任务。例如，求解器必须处理不同的数据类型，如布尔型、整数型、实数型或字符串。前三种数据类型的理论已广为人知、得到广泛发展，并已在工具中实现；而字符串数据类型的理论直到最近才开始在 Abdulla 等人（2014）；Berzish 等人（2017）；Bjørner 等人（2009）；Chen 等人（2019a、2018）；Hol´ık 等人（2018）；Lin 和 Majumdar（2018）；Liang 等人（2014）；Wang 等人（2016）；Yu 等人（2014）；Abdulla 等人（2017、2015）；Kiezun 等人（2012）；Lin 和 Barcel´o（2016）；Berzish 等人（2021）；Reynolds 等人（2019）；Blotsky 等人（2018）；Stanford 等人（2021）；Loring 等人（2019）；Trinh 等人（2020）；Chen 等人（2019b）中得到研究，尽管 A. A. Markov 早在 20 世纪 60 年代末就已结合希尔伯特第十问题对其进行研究（参见，例如 Matiyasevich（1968）；Durnev 和 Zetkina（2009）；Kosovskii（1976））。

目前大多数字符串约束判定过程都涉及所谓的分情况 规则。该规则根据变量可能的对齐方式进行分情况。几乎所有（半）判定字符串约束的过程都会用到分情况规则，包括 (Makanin (1977)) 中的 Makanin 算法、Nielsen 变换 (Nielsen (1917))（也称为 Levi 引理 (Levi (1944))），以及大多数先进求解器中实现的过程，例如 Z3 (Bjørner 等人 (2009))、CVC4 (Liang 等人 (2014))、Z3Str3 (Berzish 等人 (2017))、Norn (Abdulla 等人 (2014)) 等。本文将借助最简单的方法——Nielsen 变换，阐释我们符号方法的一般思路；尽管如此，我们认为该方法也适用于其他过程。

考虑字方程 $xz = yw$，这是本文所讨论的主要原子字符串约束类型，其中 x、 z、 y 和 w 是字变量。在确定该字方程的可满足性时，尼尔森变换（Nielsen (1917) 中提出）首先根据变量 x 和 y（分别为该方程左侧和右侧的第一个符号）的可能对齐方式进行分情况。更准确地说，它将 $xz = yw$ 的可满足性问题归约为以下四种情况中至少一种的可满足性问题：(1) y 是 x 的前缀；(2) x 是 y 的前缀；(3) x 是空字符串；(4) y 是空字符串。注意，这些情况并不互斥：例如，空字符串是每个变量的前缀。对于这些情况，尼尔森变换生成以下方程。

对于情况 (1)，即 y 是 x 的前缀，将 $xz = yw$ 中所有出现的 x 替换为 yx'，其中 x' 是一个新词变量（我们将此情况记为 $x \hookrightarrow yx'$），即得到方程 $yx'z = yw$，该方程可简化为 $x'z = w$。事实上，由于变换 $x \hookrightarrow yx'$ 会移除变量 x 的所有出现次数，因此我们可以直接重新使用变量 x，并改为执行变换 $x \hookrightarrow yx$（随后在构造模型时将此点纳入考虑）。

尼尔森变换的情形（2）只是上述情形（1）的对称对应。对于情形（3）和（4）， x 和 y分别被替换为空字符串。考虑方程 $xz = yw$ 的所有四种可能变换后，我们得到以下方程：

$$(1)\ xz = w \qquad (2)\ z = yw \qquad (3)\ z = yw \qquad (4)\ xz = w$$

（注意，(1) 和 (4) 的结果相同，(2)

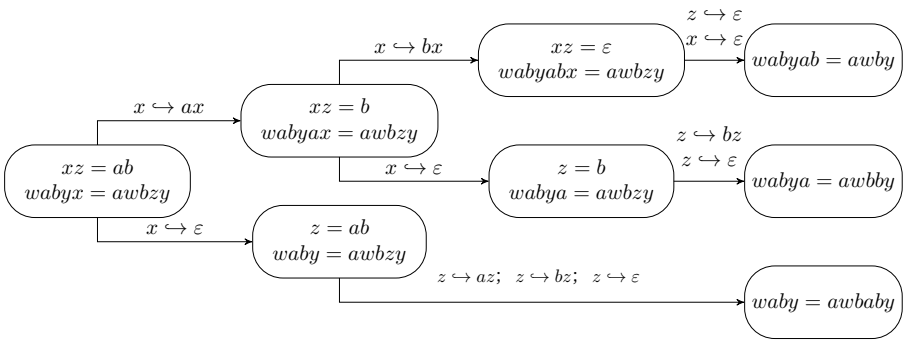


图 1：对字符串约束 $xz = ab \wedge wabyx = awbzy$ 应用尼尔森变换得到的部分证明树。叶节点是完全处理第一个字方程 $xz = ab$ 后的结果。省略了通向矛盾的分支。

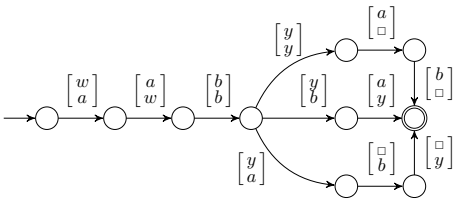


图 2：对三个方程 $wabyab = awby$、$wabya = awbby$ 和 $waby = awbaby$ 进行编码的有限自动机。

和 (3) 的结果也相同。）如果 $xz = yw$ 有解，那么上述方程中至少有一个也有解。尼尔森变换持续对所得方程应用变换规则，构建证明树，并搜索形如 $\varepsilon = \varepsilon$ 的重言式。

分别处理每个得到的方程可能会造成一些冗余（正如我们刚才已经看到的）。让我们来看图 1 中的例子：我们对字符串约束$xz = ab \wedge wabyx = awbzy$应用尼尔森变换来求解，其中 x、z、w 和 y 是字变量，a和 b 是常量符号。处理完第一个字方程 $xz = ab$后，我们得到一棵证明树，其三个叶节点 $wabyab = awby$、$wabya = awbby$ 和 $waby = awbaby$ 非常相似，它们分别共享方程左右两侧的前缀 $waby$ 和 awb。如果继续对这三个叶节点应用尼尔森变换，就会生成三棵非常相似的子树，其中的操作几乎完全相同。尤其是这些子树中靠近根节点、用于变换 $waby\ldots = awb\ldots$的节点，实质上将完全相同。因此，生成的证明树只有在处理完这段共同部分后才会开始出现差异。由此，分别处理这些方程会导致某些操作被重复执行。如果每个字方程的证明树都有 n 个叶节点，且该字符串约束是 k 个字方程的合取，那么我们可能需要生成 n^k 棵相似的子树。

如果使用符号编码，将上述叶节点的公共部分一并处理，便能更高效地进行分情况。在本文中，我们将一组方程编码为正则语言，并用有限自动机表示。图 2 给出了一个示例，其中展示了一个有限

自动机，其运行于双轨字母表之上，其中两条轨道分别表示方程的一侧。例如，方程 $wabyab = awby$ 表示为自动机所接受的词 $\left[\begin{smallmatrix} w \\ a \end{smallmatrix}\right]\left[\begin{smallmatrix} a \\ w \end{smallmatrix}\right]\left[\begin{smallmatrix} b \\ b \end{smallmatrix}\right]\left[\begin{smallmatrix} y \\ y \end{smallmatrix}\right]\left[\begin{smallmatrix} a \\ \square \end{smallmatrix}\right]\left[\begin{smallmatrix} b \\ \square \end{smallmatrix}\right]$，其中 $\square$ 符号是一种填充符，用于确保两条轨道的长度相同。

基于我们基于正则语言的符号编码，我们需要一种机制，对编码为正则语言的方程集合执行尼尔森变换步骤。我们将说明，这些变换可以编码为有理关系，并用有限状态转换器表示；整个可满足性检查问题也可以纳入正则模型检测框架。第 3 至第 6 节将逐步详细介绍具体做法。第 3 节介绍一个较简单情形下的方法：输入为二次词方程，即每个变量最多出现两次的字方程。在这种情况下，尼尔森变换是可靠且完备的，也就是说，它返回的解是正确的，并且只要存在解，就一定会返回一个解。第 4 节将该技术扩展到支持非二次字方程的合取。第 5 节将我们的方法扩展到支持字符串约束的任意布尔组合。第 6 节则为该框架加入两种额外的原子字符串约束类型——长度和正则约束，分别用于约束赋给字变量的值的长度，以及这些值在正则语言中的成员关系。

我们已将该方法实现为一个名为 Retro 的 Python 原型工具，并在两组基准测试上评估了其性能：一组是取自 Le 和 He（2018）的 Kepler_{22}，另一组是通过在 Reynolds 等人（2017）的 Python 程序上运行 PyEx 符号执行引擎，并收集 CVC4 或 Z3 无法处理的示例而得到的 PyEx-Hard。Retro解决了 Kepler_{22} 中的大多数问题（CVC4 和 Z3 在这些问题上的表现不佳）。此外，它还解决了 PyEx-Hard 中超过 50% 的基准测试，而这些基准测试既无法由 CVC4 解决，也无法由 Z3 解决。

本文是发表于 APLAS'20 会议论文集的论文（Chen 等人（2020））的扩展版本，包含文中所述引理和定理的完整证明，并进一步扩展了所提出的技术，以处理 (i) 字符串约束的任意布尔组合（第 5 节）、(ii) 长度约束（第 6.1 节）以及 (iii) 正则约束（第 6.2 节）。

2. 预备知识

字母表 Σ 是一个有限字符集合，而 Σ 上的词是由 Σ中的字符组成的序列 $w = a_1 \dots a_n$，其中 ε表示空词。我们用$w_1.w_2$（通常简写为 w_1w_2）表示词 w_1 和 w_2的连接。Σ^* 是 Σ上的所有词的集合，$\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$，且 $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$。$\Sigma$ 上的语言是 Σ^*的子集 L。给定一个词 $w = a_1 \dots a_n$，我们用 $|w|$ 表示 w 的长度 n，用 $|w|_a$ 表示字符$a \in \Sigma$ 在 w中的出现次数。此外，我们用 $w[i]$ 表示 a_i，即 w 的第 i 个字符，并用 $w[i{:}]$ 表示词 $a_i \dots a_n$。当 $i > n$ 时，$w[i]$ 和 $w[i{:}]$ 的值均为 $\perp$，这是一种特殊的未定义值，不同于所有其他值，也不同于其自身（即 $\perp \neq \perp$）。我们用 Σ^k 表示 $k \geq 2$，以表示由 Σ中的符号组成的 k元组所构成的堆叠字母表，例如，a、$b \in \Sigma$对应的 $\left[\begin{smallmatrix} a \\ b \end{smallmatrix}\right] \in \Sigma^2$。

2.1. 自动机与转换器

一个（有限） k带转换器是一个五元组 $\mathcal{T} = (Q, \Sigma, \Delta, Q_i, Q_f)$，其中$Q$ 是有限状态集合， Σ是字母表， $\Delta \subseteq Q \times \Sigma_\varepsilon^k \times Q$是由形如 $q \xrightarrow{\{a^1,\ldots,a^k\}} s$的转移组成的集合，其中 $a^1, \ldots, a^k \in \Sigma_\varepsilon$； $Q_i \subseteq Q$是初始状态集合， $Q_f \subseteq Q$是终止状态集合。 $\mathcal{T}$在 k个词组成的元组 $(w_1, \ldots, w_k)$ 上的一次运行 π是转移序列 $q_0 \xrightarrow{\{a_1^1,\ldots,a_1^k\}} q_1, q_1 \xrightarrow{\{a_2^1,\ldots,a_2^k\}} q_2, \ldots, q_{n-1} \xrightarrow{\{a_n^1,\ldots,a_n^k\}} q_n \in \Delta$，满足对每个 $i \in [1, k]$ 都有 $w_i = a_1^i a_2^i \ldots a_n^i$（注意， a_m^i可以是 ε，因此 w_i和 w_j的长度可能不同，适用于 $i \neq j$）。若 $q_0 \in Q_i$ 且 $q_n \in Q_f$，则运行 π是接受运行；若存在 $\mathcal{T}$在 $(w_1, \ldots, w_k)$ 上的一次接受运行，则元组 $(w_1, \ldots, w_k)$ k 被 $\mathcal{T}$ 接受。我们将被转换器接受的关系类称为有理关系。若 Δ 中的转移均不包含 ε，则 $\mathcal{T}$是长度保持的；长度保持转换器所接受的关系类称为正则关系。对于一个双带转换器 $\mathcal{T}$和 $(w_1, w_2) \in \mathcal{L}(\mathcal{T})$，我们将 w_1 记作输入，将 w_2 记作 $\mathcal{T}$的输出。有限自动机（FA）是一带有限转换器；有限自动机所接受的语言称为正则语言。有关自动机和转换器的更多详情，参见例如 Pin（2021）。

给定两个 k 元关系 R_1, R_2，我们定义它们的连接 $R_1.R_2 = \{(u_1v_1, \ldots, u_kv_k) \in (\Sigma^*)^k \mid (u_1, \ldots, u_k) \in R_1 \wedge (v_1, \ldots, v_k) \in R_2\}$；给定两个二元关系 R_1, R_2，我们定义它们的复合 $R_1 \circ R_2 = \{(x, z) \in (\Sigma^*)^2 \mid \exists y \in \Sigma^*: (x, y) \in R_2 \wedge (y, z) \in R_1\}$。给定一个 k 元关系 R，我们定义 $R^0 = \{\varepsilon\}^k$, $R^{i+1} = R.R^i$ ，其中 $i \geq 0$。于是， R 的迭代定义为 $R^* = \bigcup_{i \geq 0} R^i$。给定一个语言 $\mathcal{L} \subseteq \Sigma^*$和一个二元关系 R，我们用 $R(\mathcal{L})$ 表示语言 $\{y \in \Sigma^* \mid \exists x \in \mathcal{L}: (x, y) \in R\}$，称为 $\mathcal{L}$ 的 R像。我们还用 $R^{-1}(w)$ 表示语言 $\{u \mid (w, u) \in R\}$，称为词 w 的原像。

命题 1 (Berstel（1979）)。 以下命题成立：

（*i*）二元有理关系类在（有限）并、复合、连接和迭代运算下封闭；但在交和补运算下不封闭。（*ii*）对于二元有理关系 R、正则语言 $\mathcal{L}$ 和词 w，语言 $R(\mathcal{L})$ 和 $R^{-1}(w)$ 也都是有效正则的（即可以计算得到）。（*iii*）正则关系类在布尔运算下封闭。

2.2. 字符串约束

设 Σ 为一个字母表， $\mathbb{X}$ 为一组取值范围为 Σ^* 的字变量，其中 $\mathbb{X} \cap \Sigma = \emptyset$。我们使用 $\Sigma_{\mathbb{X}}$ 表示扩展字母表 $\Sigma \cup \mathbb{X}$。 $\mathbb{X}$ 的一个赋值是映射 I： $\mathbb{X} \to \Sigma^*$。字项是字母表 $\Sigma_{\mathbb{X}}$上的字符串。通过定义 $I(\varepsilon) = \varepsilon$、 $I(a) = a$以及 $I(x.w) = I(x).I(w)$，我们将赋值 I 提升到字项，其中 $a \in \Sigma$、 $x \in \Sigma_{\mathbb{X}}$和 $w \in \Sigma_{\mathbb{X}}^*$。字方程 φ_e 的形式为 $t_1 = t_2$ ，其中 t_1 和 t_2 是字项。若 $I(t_1) = I(t_2)$，则 I 是 φ_e 的一个模型。我们将字方程称为原子字符串约束。字符串约束通过使用布尔连接词（$\wedge$、$\vee$、$\neg$）由原子字符串约束构造，其语义按标准方式定义。字符串约束

$\sigma \models P \subseteq R$ 当且仅当 $\sigma(P)$ 是 $\sigma(R)$ 的子集；$\sigma \models P = R + 1$ 当且仅当 $\sigma(P) = \{r + 1 \mid r \in \sigma(R)\}$，并且 $r + 1 \leq |\sigma|\}\sigma \models w[P] = a$ 当且仅当对于所有 $p \in P$，都有 $\sigma(w)[p]$ 是 a；$\sigma \models \varphi_1 \wedge \varphi_2$ 当且仅当 $\sigma \models \varphi_1$，并且 $\sigma \models \varphi_2\sigma \models \neg\varphi$ 当且仅当非 $\sigma \models \varphi$；$\sigma \models \forall^{\mathbb{P}} P(\varphi)$ 当且仅当对于所有 $v \subseteq \{1, \ldots, |\sigma|\}$，都有 $\sigma[P \mapsto v] \models \varphi\sigma \models \forall^{\mathbb{W}} w(\varphi)$；当且仅当对于所有 $v \in \Gamma^{|\sigma|}$，都有 $\sigma[w \mapsto v] \models \varphi$

图 3： MSO(Str) 的语义

当且仅当它有一个模型时，它是 可满足的。给定一个字项 $t \in \Sigma_{\mathbb{X}}^*$、一个变量 $x \in \mathbb{X}$ 和一个字项 $u \in \Sigma_{\mathbb{X}}^*$，我们用 $t[x \mapsto u]$ 表示将 t 中所有出现的 x 替换为 u 后得到的字项，例如 $(abxcxy)[x \mapsto cy] = abcyccyy$。如果每个变量在 ψ 中最多出现两次，我们称字符串约束 ψ 为 二次的；如果每个变量在 ψ 中最多出现三次，则称其为 三次的。

我们使用以下术语。设 Φ 为一个字符串约束。用于求解字符串约束的（半）算法 **A**

1. 若**A** 返回一个对 Φ 中变量的赋值 I，则 I 是 Φ 的模型；此时称其为 可靠的；2. 若 Φ 可满足，则**A** 在有限步内返回 Φ 的一个模型；此时称其为完备的；3. 若**A** 总能在有限步内返回一个赋值或 false，则称其为终止的。

2.3. 字符串上的单子二阶逻辑（MSO(Str)）

我们在字母表 Γ 上定义字符串上的单子二阶逻辑（MSO(Str)）（B¨uchi（1960））， 如下所示。设 $\mathbb{W}$ 是一个可数集，其元素为字符串变量，取值范围为 Γ^*；设 $\mathbb{P}$ 是一个可数集，其元素为集合（二阶）位置变量，取值范围为 $\mathbb{N}_1 = \mathbb{N} \setminus \{0\}$ 的有限子集，使得$\mathbb{W} \cap \mathbb{P} = \emptyset$。 MSO(Str) 的公式 φ 定义如下：

$$\varphi ::= P \subseteq R \mid P = R + 1 \mid w[P] = a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \forall^{\mathbb{P}} P(\varphi) \mid \forall^{\mathbb{W}} w(\varphi)$$

其中 P， $R \in \mathbb{P}$、 $w \in \mathbb{W}$ 和 $a \in \Gamma$。我们用 $\varphi(w_1, \ldots\ldots, w_k)$ 表示 φ 的自由变量包含于 $\{w_1, \ldots, w_k\}$。

MSO(Str) 的语义定义见图 3。 MSO(Str) 变量赋值是一个赋值 σ: $\mathbb{W} \cup \mathbb{P} \to (\Gamma^* \cup 2^{\mathbb{N}_1})$，它遵循变量的类型，并附加以下要求：对于每个 u, $v \in \mathbb{W}$，都有$|\sigma(u)| = |\sigma(v)|$。（通常在 σ 中省略未使用的变量。）我们用 $|\sigma|$ 表示任意 $w \in \mathbb{W}$ 的值 $|\sigma(w)|$。注意， $|\sigma|$ 定义良好，因为我们假设所有 $w \in \mathbb{W}$ 都映射到长度相同的字符串。记号 $\sigma[x \mapsto v]$ 表示 σ 的一种变体，其中变量 x 的赋值更改为值 v。

如果一个 MSO(Str) 公式不含自由位置变量，我们就称其为字符串公式。这样的公式（含有 k 个自由字符串变量）表示 Γ^*上的 k元关系。特别地，给定一个含有 k 个自由字符串变量 $w_1, \ldots, w_k$的 MSO(Str) 字符串公式 $\varphi(w_1, \ldots, w_k)$，我们用 $\mathcal{L}(\varphi)$表示该关系

$$
\begin{array}{rcl@{\qquad}rcl}
\varphi \vee \psi & \triangleq & \neg(\varphi \wedge \psi) & \exists^{\mathbb{P}} P(\varphi) & \triangleq & \neg\forall^{\mathbb{P}} P(\neg\varphi) \\
\varphi \to \psi & \triangleq & \neg\varphi \vee \psi & \exists^{\mathbb{W}} w(\varphi) & \triangleq & \neg\forall^{\mathbb{W}} w(\neg\varphi) \\
P = R & \triangleq & P \subseteq R \ \wedge \ R \subseteq P & P = \emptyset & \triangleq & \forall^{\mathbb{P}} R(P \subseteq R) \\
Sing(P) & \triangleq & \neg(P = \emptyset) \ \wedge \ \forall^{\mathbb{P}} R(R \subseteq P \to (R = \emptyset \ \vee \ R = P)) & & & \\
p \in R & \triangleq & Sing(p) \wedge p \subseteq R & & & \\
p \leq r & \triangleq & \forall^{\mathbb{P}} T((p \in T \wedge \forall^{\mathbb{P}} u(u \in T \to \exists^{\mathbb{P}} v(v = u + 1 \wedge v \in T))) \to r \in T) & & & \\
p < r & \triangleq & p \leq r \wedge \neg(p = r) & & & \\
x = 1 & \triangleq & \forall^{\mathbb{P}} u(u \leq x \to u = x) & x = \$ & \triangleq & \forall^{\mathbb{P}} u(x \leq u \to u = x) \\
w_1[P] = w_2[R] & \triangleq & \bigvee_{a \in \Gamma} (w_1[P] = a \wedge w_2[R] = a) & & &
\end{array}
$$

对于所有 $j \geq 1$ 以及任意项 γ:

$$
\begin{array}{rcl}
P = R + j & \triangleq & \exists^{\mathbb{P}} S_1 \ldots \exists^{\mathbb{P}} S_j (S_1 = P + 1 \wedge S_2 = S_1 + 1 \wedge \ldots \wedge R = S_j + 1) \\
p = j & \triangleq & \exists^{\mathbb{P}} z(z = 1 \wedge p = z + (j - 1)) \\
w[P + j] = \gamma & \triangleq & \exists^{\mathbb{P}} S(S = P + j \wedge w[S] = \gamma) \\
w[P - j] = \gamma & \triangleq & \exists^{\mathbb{P}} S(P = S + j \wedge w[S] = \gamma) \\
w[\$ - j] = \gamma & \triangleq & \exists^{\mathbb{P}} r \exists^{\mathbb{P}} s(r = \$ \wedge r = s + j \wedge w[s] = \gamma) \\
w[j] = \gamma & \triangleq & \exists^{\mathbb{P}} p(p = j \wedge w[p] = \gamma) \\
p \geq j & \triangleq & \exists^{\mathbb{P}} s(s = j \wedge p \geq s)
\end{array}
$$

图4: MSO(Str)的语法糖

$\{(x_1, \ldots\ldots, x_k) \in (\Gamma^*)^k \mid \{w_1 \mapsto x_1, \ldots, w_k \mapsto x_k\} \models \varphi\}$。在 $k = 1$ 的特殊情况下， φ 表示一种语言 $\mathcal{L}(\varphi) \subseteq \Gamma^*$。

命题 2 (Thatcher 和 Wright（1968）)。 由含有1 个自由字符串变量的 MSO(Str) 字符串公式所表示的语言类，恰好是正则语言类。此外，对于 $k > 1$，由含有 k 个自由字符串变量的 MSO(Str) 字符串公式表示的关系类，恰好是正则关系类。

.....的语法糖 MSO(Str)。在图4中，我们定义了标准语法糖，以便更简洁地书写 MSO(Str) 公式。大多数语法糖都是标准的，不过我们还是来解释一些不太标准的记法：$Sing(P)$表示 P 是一个单位置集合， $p \leq r$ 表示 p 和 r 都是单个位置，且 p 小于或等于 r， $x = 1$ 和 $x = \$$分别表示x 是第一个位置和最后一个位置，而 $P = R + j$ 表示 P等于 R 的所有位置都增加 j后的结果。我们还扩展了语法，以允许使用一阶变量（我们滥用记法，使用与二阶变量相同的量词记法，但用小写字母表示一阶变量

$$\frac{\alpha u = \alpha v}{u = v}\ (\text{trim}) \qquad \frac{xu = v}{u[x \mapsto \varepsilon] = v[x \mapsto \varepsilon]}\ (x \hookrightarrow \varepsilon)$$

$$\frac{xu = \alpha v}{x(u[x \mapsto \alpha x]) = v[x \mapsto \alpha x]}\ (x \hookrightarrow \alpha x)$$

图 5：Nielsen 变换的规则，其中包含 $x \in \mathbb{X}$、$\alpha \in \Sigma_{\mathbb{X}}$ 和 $u, v \in \Sigma_{\mathbb{X}}^*$。对称规则省略。

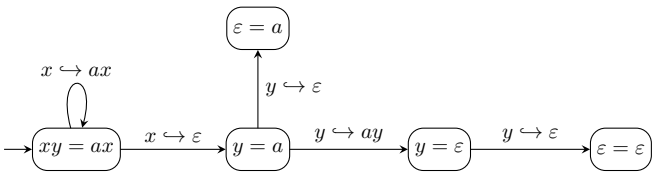


图 6：由 Nielsen 变换生成的方程 $xy = ax$ 的证明图。

小写字母):

$$\forall^{\mathbb{P}} p(\varphi) \triangleq \forall^{\mathbb{P}} P(Sing(P) \to \varphi[p \mapsto P])$$
$$\exists^{\mathbb{P}} p(\varphi) \triangleq \exists^{\mathbb{P}} P(Sing(P) \wedge \varphi[p \mapsto P])$$

其中，$\varphi[p \mapsto P]$ 表示将 φ 中所有自由出现的 p 代换为 P。

2.4. 尼尔森变换

如引言中已经简要提到的，尼尔森变换可用于检查字方程合取式的可满足性。我们使用图 5 所示的三条规则；除了引言中介绍过的规则 $x \hookrightarrow \alpha x$ 和 $x \hookrightarrow \varepsilon$ 外，还有一条（trim）规则，用于从方程两侧移除公共前缀。

给定一个字方程组，可能有多个尼尔森变换适用于该系统，从而得到不同的变换后的方程，并可对这些方程执行其他尼尔森变换，如图 1 所示。尝试所有可能的变换会生成一棵树（一般情况下为一张图），其节点包含字方程合取式，其边标记为所应用的变换。当且仅当图中至少有一个叶节点是重言式时，树根处的字方程合取式才是可满足的；也就是说，该叶节点包含形如 $\varepsilon = \varepsilon \wedge \cdots \wedge \varepsilon = \varepsilon$ 的合取式。作为示例，考虑可满足的方程 $xy = ax$，其中 x，y 是字变量，而 a 是一个符号，其证明图如图 6 所示。

引理 3（参见 Makanin（1977）；Diekert（2002））。尼尔森变换是可靠且完备的。此外，如果字方程组是二次的，证明图就是有限的。

即使我们使用以下策略构造证明树，引理 3仍然成立：每次应用 $x \hookrightarrow \alpha x$ 或 $x \hookrightarrow \varepsilon$ 后，尽可能多次应用（trim）规则。我们用 $x \rightarrowtail \alpha x$ 表示一次 $x \hookrightarrow \alpha x$ 规则的应用后尽可能多次应用（trim），并用$x \rightarrowtail \varepsilon$ 表示应用 $x \hookrightarrow \varepsilon$ 后再应用（trim）。

2.5. 正则模型检测

正则模型检测（RMC）（参见 Kesten 等人（2001年）；Wolper和 Boigelot（1998）；Bouajjani 等人（2000）；Abdulla（2012）；Bouajjani 等人（2012））是用于验证无限状态系统的一个框架。在 RMC 中，每个系统配置都表示为字母表 Σ上的一个词。初始配置集合 $\mathcal{I}$ 和目标配置 $\mathcal{D}$ 被表示为 Σ上的正则语言。转移关系 $\mathcal{T}$ 被表示为 Σ^*上的二元有理关系。一个正则模型检测可达性问题由三元组（$\mathcal{I}$，$\mathcal{T}$，$\mathcal{D}$）表示，并询问 $\mathcal{T}^{rt}$（$\mathcal{I}$）$\cap \mathcal{D} \neq \emptyset$是否成立，其中 $\mathcal{T}^{rt}$ 表示 $\mathcal{T}$的自反传递闭包。解决该问题的一种方法是开始计算序列 $\mathcal{T}$(0)（$\mathcal{I}$） $\mathcal{T}$(1)（$\mathcal{I}$） $\mathcal{T}$(2)（$\mathcal{I}$）.....，其中 $\mathcal{T}$(0)（$\mathcal{I}$）$= \mathcal{I}$，且 $\mathcal{T}^{(n+1)}$（$\mathcal{I}$）$= \mathcal{T}$（$\mathcal{T}^{(n)}$（$\mathcal{I}$））。在计算该序列的过程中，我们可以检查是否找到与 $\mathcal{D}$重叠的 $\mathcal{T}^{(i)}$（$\mathcal{I}$）；若找到，则可推知 $\mathcal{D}$ 是可达的。另一方面，如果得到一个满足$\bigcup_{0\leq i<n} \mathcal{T}^i$（$\mathcal{I}$） $\supseteq \mathcal{T}^n$（$\mathcal{I}$）的序列，我们便知道已探索所有可能的系统配置而未到达 $\mathcal{D}$，因此 $\mathcal{D}$ 不可达。RMC 可达性问题一般是不可判定的（例如，这可以通过从图灵机配置可达性问题进行归约轻易证明）。

3. 使用 RMC 求解字方程

本节将介绍一种基于符号 RMC 的框架，用于求解字符串约束。该框架将字符串约束编码为正则语言，并将尼尔森变换的步骤编码为有理关系。由此，字符串约束的可满足性可归约为 RMC 的可达性问题。

3.1. 尼尔森变换作为字操作

下面介绍如何将单个字方程的尼尔森变换表示为对词的操作。我们将一个字方程 eq: $\mathsf{t}_\ell = \mathsf{t}_r$ 视为一对字项 $e_{eq} = (\mathsf{t}_\ell, \mathsf{t}_r)$，分别对应方程的左侧和右侧；因此 $e_{eq} \in \Sigma_{\mathbb{X}}^* \times \Sigma_{\mathbb{X}}^*$。不失一般性，我们假设 $\mathsf{t}_\ell[1] \neq \mathsf{t}_r[1]$；若不满足此条件，则通过应用（trim）尼尔森变换（参见图 5）对方程进行预处理，以裁去 t_ℓ 和 t_r的公共前缀。

示例 1. 字方程 eq: $xay = yx$ 由字项对 $e_1 = (xay, yx)$ 表示。应用尼尔森变换生成的完整证明图如图 7 所示。 □

尼尔森变换的一条规则（参见第2.4节）用一个（部分）函数 τ: $(\Sigma_{\mathbb{X}}^* \times \Sigma_{\mathbb{X}}^*) \to (\Sigma_{\mathbb{X}}^* \times \Sigma_{\mathbb{X}}^*)$ 表示。给定字方程 eq 的一对字项 $(\mathsf{t}_\ell, \mathsf{t}_r)$，函数 τ 会将其变换为字方程 eq' 的一对字项，该字方程可通过对 eq 执行尼尔森变换的相应步骤得到。在表述尼尔森变换的规则之前，我们先定义执行相应代换的函数。对于 $x \in \mathbb{X}$ 和 $\alpha \in \Sigma_{\mathbb{X}}$，我们定义

$$\begin{aligned} \tau_{x\mapsto\alpha x} &= \{\, (\mathsf{t}_\ell, \mathsf{t}_r) \mapsto (\mathsf{t}'_\ell, \mathsf{t}'_r) \mid \mathsf{t}'_\ell = \mathsf{t}_\ell[x \mapsto \alpha x] \wedge \mathsf{t}'_r = \mathsf{t}_r[x \mapsto \alpha x] \,\} \text{ and} \\ \tau_{x\mapsto\varepsilon} &= \{\, (\mathsf{t}_\ell, \mathsf{t}_r) \mapsto (\mathsf{t}'_\ell, \mathsf{t}'_r) \mid \mathsf{t}'_\ell = \mathsf{t}_\ell[x \mapsto \varepsilon] \wedge \mathsf{t}'_r = \mathsf{t}_r[x \mapsto \varepsilon] \,\}. \end{aligned} \tag{1}$$

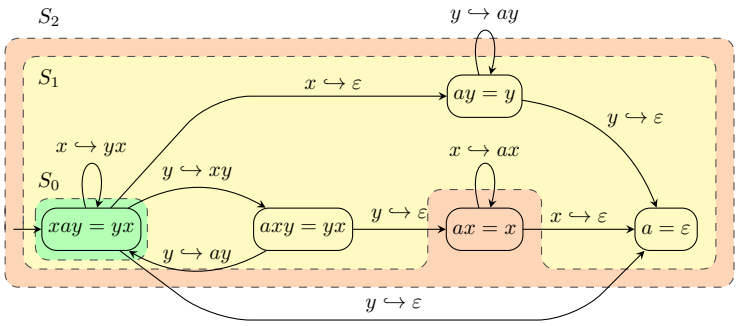


图 7：对方程 $xay = yx$ 运行尼尔森变换所得到的证明图。集合 S_0、S_1 和 S_2 分别是在算法运行 0、1 和 2 步时探索的节点集合。

函数 $\tau_{x\mapsto\alpha x}$ 执行代换 $x \mapsto \alpha x$，而函数 $\tau_{x\mapsto\varepsilon}$ 执行代换 $x \mapsto \varepsilon$。

示例 2. 考虑示例 1 中的字项对 $e_1 = (xay, yx)$。应用 $\tau_{x\mapsto yx}(e_1)$ 将得到字项对 $e_2 = (yxay, yyx)$，应用 $\tau_{x\mapsto\varepsilon}(e_1)$ 将得到字项对 $e_3 = (ay, y)$。 □

上文引入的函数没有考虑等式两边的首个符号，也不会删除等式两边的公共前缀，而这是一项使尼尔森变换能够终止的必要操作。因此，我们定义以下函数，用于裁去等式两边字项中匹配的（最长）前缀：

$$\tau_{trim} = \{\, (\mathsf{t}_\ell, \mathsf{t}_r) \mapsto (\mathsf{t}'_\ell, \mathsf{t}'_r) \mid \exists i \geq 1 \forall j < i \big(\mathsf{t}_\ell[i] \neq \mathsf{t}_r[i] \wedge \mathsf{t}_\ell[j] = \mathsf{t}_r[j] \\ \wedge \mathsf{t}'_\ell = \mathsf{t}_\ell[i:] \wedge \mathsf{t}'_r = \mathsf{t}_r[i:]\big) \,\}. \tag{2}$$

例 3. 继续沿用我们的运行示例，应用 $\tau_{trim}(e_2)$ 会产生数对 $e'_2 = (xay,\ yx)$；此外，$\tau_{trim}(e_3)$ 会产生数对 $e'_3 = (ay,\ y)$。 □

现在我们可以定义与尼尔森变换规则相对应的函数了。特别是，对于 $x \in \mathbb{X}$ 和 $\alpha \in \Sigma_{\mathbb{X}}$（参见第2.4节），规则 $x \rightarrowtail \alpha x$ 及其对称变体（即 x 是某个方程左侧或右侧的第一个符号）可以用函数表示为

$$\tau_{x \rightarrowtail \alpha x} = \tau_{trim} \circ \{\, (\mathsf{t}_\ell, \mathsf{t}_r) \mapsto \tau_{x\mapsto\alpha x}(\mathsf{t}_\ell, \mathsf{t}_r) \mid (\mathsf{t}_r[1] = x \wedge \mathsf{t}_\ell[1] = \alpha) \vee \\ (\mathsf{t}_r[1] = \alpha \wedge \mathsf{t}_\ell[1] = x) \,\} \tag{3}$$

而对于 $x \in \mathbb{X}$，规则 $x \rightarrowtail \varepsilon$ 及其对称变体可以表示为函数

$$\tau_{x \rightarrowtail \varepsilon} = \tau_{trim} \circ \{\, (\mathsf{t}_\ell, \mathsf{t}_r) \mapsto \tau_{x\mapsto\varepsilon}(\mathsf{t}_\ell, \mathsf{t}_r) \mid (\mathsf{t}_\ell[1] = x \vee \mathsf{t}_r[1] = x) \}. \tag{4}$$

如果我们继续将上文定义的函数应用于各个字项对，同时搜索对 $(\varepsilon,\ \varepsilon)$——它表示原方程 eq 存在解的情况——就会得到 Nielsen 变换图（参见第2.4节）。接下来，我们将展示如何在一个字方程集合的表示上，同时以符号方式执行这些步骤。

3.2. 单词方程的符号算法

本节将介绍我们用于求解字方程的符号算法的核心思想。我们首先关注单个字方程的情形，并在后续各节中将该算法扩展到更丰富的类别。

我们的算法并非针对单个方程应用变换规则，而是一次应用于整个方程集合。为此，我们定义关系 $\mathcal{T}_{x \rightarrowtail \alpha x}$ 和 $\mathcal{T}_{x \rightarrowtail \varepsilon}$，将所有可能的 $x \in \mathbb{X}$ 和 $\alpha \in \Sigma_{\mathbb{X}}$ 对应的 $\tau_{x \rightarrowtail \alpha x}$ 和 $\tau_{x \rightarrowtail \varepsilon}$ 的各个版本汇总起来。这些关系的签名为 $(\Sigma_{\mathbb{X}}^* \times \Sigma_{\mathbb{X}}^*) \times (\Sigma_{\mathbb{X}}^* \times \Sigma_{\mathbb{X}}^*)$，定义如下：

$$\mathcal{T}_{x \rightarrowtail \alpha x} = \bigcup_{y \in \mathbb{X}, \alpha \in \Sigma_{\mathbb{X}}} \tau_{y \rightarrowtail \alpha y} \qquad \mathcal{T}_{x \rightarrowtail \varepsilon} = \bigcup_{y \in \mathbb{X}} \tau_{y \rightarrowtail \varepsilon} \tag{5}$$

请注意这些关系的以下两个性质：(i) 它们会生成所有可能的、适用于方程两侧首符号的 Nielsen 变换步骤的输出；(ii) 它们包含裁剪操作。

我们将引入的关系合成为一个关系，记作 $\mathcal{T}_{step}$，并定义为 $\mathcal{T}_{step} = \mathcal{T}_{x \rightarrowtail \alpha x} \cup \mathcal{T}_{x \rightarrowtail \varepsilon}$。随后，关系 $\mathcal{T}_{step}$ 可用于一步计算方程中一组字项的所有后继。对于一组字项 S，我们可以计算 S 的 $\mathcal{T}_{step}$-像，从而得到 S 中所有字项对的后继。给定字方程 eq：$\mathsf{t}_\ell = \mathsf{t}_r$，其初始配置为集合 $E_{eq} = \{(\mathsf{t}_\ell, \mathsf{t}_r)\}$。

例4. 将运行示例提升到集合上的已介绍概念后，我们从集合 $E_{eq} = S_0 = \{e_1 = (xay, yx)\}$ 开始。在对 E_{eq} 应用 $\mathcal{T}_{step}$ 后，我们得到集合 $S_1 = \{e_2' = (xay, yx)\ e_3' = (ay, y), (axy, yx), (a, \varepsilon)\}$。$e_2'$ 和 e_3' 这两对已在前文描述，(axy, yx) 这一对由变换 $\tau_{y \rightarrowtail xy}$ 得到，而 (a, ε) 这一对由变换 $\tau_{y \rightarrowtail \varepsilon}$ 得到。如果继续计算 $\mathcal{T}_{step}(S_1)$，我们会得到集合 $S_2 = S_1 \cup \{(ax, x)\}$，如图 7 所示（$(ax, x)$ 这一对是使用变换 $\tau_{y \rightarrowtail \varepsilon}$ 从 (axy, yx) 得到的）。 □

利用符号表示，我们可以将检查字方程 eq 的可满足性问题表述为以下任务：

- 要么测试 $(\varepsilon, \varepsilon) \in \mathcal{T}_{step}^{rt}(E_{eq})$ 是否成立；若该成员关系成立，则意味着约束 eq 是可满足的；或者
- 找到一个集合（称为不可满足不变式）E_{inv}，使得 $E_{eq} \subseteq E_{inv}$、$(\varepsilon, \varepsilon) \notin E_{inv}$，且 $\mathcal{T}_{step}(E_{inv}) \subseteq E_{inv}$，从而表明 eq 不可满足。

在接下来的章节中，我们将展示如何将该问题编码到 RMC 框架中。

例5。在我们的运行示例中，为了继续进行，当我们将 $\mathcal{T}_{step}$ 应用于 S_2 时，得到 $\mathcal{T}_{step}(S_2) \subseteq S_2$。由于 $e_1 \in S_2$ 且 $(\varepsilon, \varepsilon) \notin S_2$，集合 S_2 就是我们的不可满足不变式，这意味着 eq 不可满足。 □

3.3. 迈向符号编码

现在我们来讨论将字方程可满足性问题编码到 RMC 的几种可能方法。回想一下，我们的任务是找到一种编码，使得

被编码的方程（对应于 RMC 中的初始配置）和可满足性条件（对应于目标配置）都是正则语言，并且变换（转移）关系是有理关系。我们首先介绍两种行不通的编码方法，分析它们为何不可用，然后介绍一种可行的编码方法，也就是我们实际采用的方法。

将一组字方程编码为正则语言的第一个想法，是将一对 $e_{eq} = (\mathsf{t}_\ell,\ \mathsf{t}_r)$ 编码为一个词 $\mathsf{t}_\ell \cdot = \cdot \mathsf{t}_r$，其中 $\ominus \notin \Sigma_{\mathbb{X}}$。人们很快会发现，不管变换 $\tau_{x \rightarrowtail \alpha x}$ 和 $\tau_{x \rightarrowtail \varepsilon}$ 是有理的（即可以用转换器表示），但变换 τ_{trim} 并非如此；该变换会从两边移除最长的匹配前缀（需要一个带有无界内存的转换器来记住这个前缀）。

第二种编码尝试可以是将 $e_{eq} = (\mathsf{t}_\ell, \mathsf{t}_r)$ 编码为一个有理二元关系，例如用一个双带转换器表示（不一定是长度保持的；一条带用于 t_ℓ，另一条用于 t_r），并用四带转换器表示这些变换（两条带用于 t_ℓ 和 t_r，另外两条用于 t'_ℓ 和 t'_r）。实现 $\tau_{x \rightarrowtail yx}$ 和 $\tau_{x \rightarrowtail \varepsilon}$ 的转换器很容易构造，实现 τ_{trim} 的转换器也同样如此，因此这个方案看起来颇具吸引力。然而，人们很快就会意识到，计算 $\mathcal{T}_{step}(E_{eq})$ 时会遇到问题。具体来说，由于 E_{eq} 和 $\mathcal{T}_{step}$ 都表示为有理关系，因此需要首先计算的交 $(E_{eq} \times \Sigma_{\mathbb{X}}^* \times \Sigma_{\mathbb{X}}^*) \cap \mathcal{T}_{step}$，可能不再是有理关系。为什么呢？假设 $E_{eq} = \{\, (a^m b^n, c^m) \mid m, n \geq 0 \,\}$ 且 $\mathcal{T}_{step} = \{\, (a^m b^n, c^n, \varepsilon, \varepsilon) \mid m, n \geq 0 \,\}$。那么，交 $(E_{eq} \times \Sigma_{\mathbb{X}}^* \times \Sigma_{\mathbb{X}}^*) \cap \mathcal{T}_{step} = \{\, (a^n b^n, c^n, \varepsilon, \varepsilon) \mid n \geq 0 \,\}$ 显然不再是有理关系。

3.4. 将二次方程符号编码到 *RMC*

因此，我们最终采用以下方法，用正则语言表示词方程。词项对的集合由带填充符号 $\Sigma^2_{\mathbb{X},\ \square}$ 的双轨字母表上的正则语言表示，其中 $\Sigma_{\mathbb{X},\ \square} = \Sigma_{\mathbb{X}} \cup \{\square\}$，并使用有限自动机。例如，$e_1 = (xay,\ yx)$ 将由正则语言 $\left[\begin{smallmatrix} x & a & y \\ y & x & \square \end{smallmatrix}\right]\left[\begin{smallmatrix} \square \\ \square \end{smallmatrix}\right]^*$ 表示。换言之，方程 e_1 有许多种编码，它们的区别在于填充方式，其中 $\left[\begin{smallmatrix} x & a & y \\ y & x & \square \end{smallmatrix}\right]$ 是最短的编码。方程的有效表示包含所有这些编码。另一方面，尼尔森变换由双轨字母表上的（二进制且通常不保长度的）有理关系表示 $\Sigma^2_{\mathbb{X},\ \square}$（每个有序对的第一项指方程的一种编码，第二项指应用于该编码的特定变换）。例如，变换 $\tau_{x \rightarrowtail \varepsilon}$ 由一个有理关系表示，该关系包含例如有序对 $\left(\left[\begin{smallmatrix} x & a & y & \square \\ y & x & \square & \square \end{smallmatrix}\right],\ \left[\begin{smallmatrix} a & y & \square \\ y & \square & \square \end{smallmatrix}\right]\right)$；变换 $\tau_{y \rightarrowtail xy}$ 则由一个有理关系表示，该关系包含例如有序对 $\left(\left[\begin{smallmatrix} x & a & y & \square \\ y & x & \square & \square \end{smallmatrix}\right],\ \left[\begin{smallmatrix} a & x & y \\ y & x & \square \end{smallmatrix}\right]\right)$。

形式化地说，我们首先定义方程编码函数 eqencode：$(\Sigma_{\mathbb{X}}^*)^2 \to (\Sigma^2_{\mathbb{X},\ \square} \setminus \{\left[\begin{smallmatrix} \square \\ \square \end{smallmatrix}\right]\})^*$，使得对于一对字项 $\mathsf{t}_\ell = a_1 \ldots a_n$ 和 $\mathsf{t}_r = b_1 \ldots b_m$（不失一般性地假设 $n \geq m$），有 $\left[\begin{smallmatrix} a_1 & \cdots & a_m & a_{m+1} & \cdots & a_n \\ b_1 & \cdots & b_m & \square & \cdots & \square \end{smallmatrix}\right] \in$ eqencode$(\mathsf{t}_\ell,\ \mathsf{t}_r)$。我们还将 eqencode 提升到词项对集合 $S \subseteq \Sigma_{\mathbb{X}}^* \times \Sigma_{\mathbb{X}}^*$ 上，即 eqencode$(S) = \{$eqencode$(\mathsf{t}_\ell,\ \mathsf{t}_r) \mid (\mathsf{t}_\ell,\ \mathsf{t}_r) \in S\}$。

设 σ 为一个符号。对于一个词 w，我们相对于 σ 定义其**填充**为语言 $\mathsf{pad}_\sigma = \{(w,\ w') \mid w' \in \{w\}.\{\sigma\}^*\}$，也就是说，它是通过在 w 后任意添加若干个 σ 而得到的词的集合。此外，我们还构造了一个（不保持长度的）转换器 T_{trim}，用于对其输入进行裁剪；通过一个双状态变换器可以很容易地实现这一点：对于 $\beta \in \Sigma_{\mathbb{X},\ \square}$，将形如 $\left[\begin{smallmatrix}\beta\\ \beta\end{smallmatrix}\right]$ 的符号前缀替换为 ε。我们将函数 encode 定义为 $\mathsf{encode}\ = T_{trim} \circ \mathsf{pad}_{\left[\begin{smallmatrix}\square\\ \square\end{smallmatrix}\right]} \circ \mathsf{eqencode}$，用于将字方程编码为正则语言；也就是说，它获取方程的编码，添加填充，并裁剪方程两侧的最大公共前缀。例如，$\mathsf{encode}(bxay,\ byx) = \left[\begin{smallmatrix}x & a & y\\ y & x & \square\end{smallmatrix}\right]\left[\begin{smallmatrix}\square\\ \square\end{smallmatrix}\right]^*$。此外，对于方程 $e \in \Sigma_{\mathbb{X}}^* \times \Sigma_{\mathbb{X}}^*$、词 $w \in \mathsf{encode}(e)$ 以及 Nielsen 规则 ρ，我们使用 $w[\rho]$ 来表示集合 $\mathsf{encode}(\tau_\rho(e))$。

引理 4。 *给定字方程 eq：$\mathtt{t}_\ell = \mathtt{t}_r$ 对于 $\mathtt{t}_\ell, \mathtt{t}_\ell \in \Sigma_{\mathbb{X}}^*$，集合 $\mathsf{编码}(eq)$ 是正则的。*

证明。 不失一般性，我们假设 $|\mathtt{t}_r| \le |\mathtt{t}_\ell|$。我们给出如下 $\mathrm{MSO}(\mathrm{S{\scriptstyle TR}})$ 公式，用于对 eq 进行编码：

$$\varphi_{eq}(w,w') \triangleq \bigwedge_{1\le k\le|\mathtt{t}_\ell|} w[k]=\mathtt{t}_\ell[k] \ \wedge \bigwedge_{1\le k\le|\mathtt{t}_r|} w'[k]=\mathtt{t}_r[k] \ \wedge \bigwedge_{|\mathtt{t}_r|<k\le|\mathtt{t}_\ell|} w'[k]=\square \ \wedge \\ \forall^{\mathbb{P}} p((p>|\mathtt{t}_\ell|)\to(w[p]=\square\wedge w'[p]=\square)) \tag{6}$$

由命题 2 可知，$\mathcal{L}(\varphi_{eq})$ 是一个正规二元关系，而且还可以将其解释为定义在复合字母表 $\Sigma^2_{\mathbb{X},\square}$ 上的正则语言。由于正则语言关于有理关系（实现 T_{trim}）的像也是正则的（参见命题 1），因此 $\mathsf{encode}(\mathcal{L}(\varphi_{eq}))$ 也是正则的。

□

使用上述编码来表达 $\tau_{x\rightarrowtail\alpha x}$ 和 $\tau_{x\rightarrowtail\varepsilon}$ 变换时，我们却遇到了需要无界内存的问题。例如，对于语言 $L = \left[\begin{smallmatrix}x\\ y\end{smallmatrix}\right]^*$，实现 $\tau_{x\rightarrowtail yx}$ 的转换器需要记住它在输入的第一轨道上看到 x 的次数（事实上，L 关于 $\tau_{x\rightarrowtail yx}$ 的像，即集合 $\{\ \mathsf{encode}(u,v) \mid \exists n\ge 0\colon\ u=(yx)^n \wedge v = y^n\square^n\ \}$，不再是正则的）。

我们分几步来解决这个问题：首先，对于方程 eq 为二次的情形，我们给出一种能够正确表示变换规则的有理关系，并在第 4 节将我们的算法扩展到变量出现次数更多的方程。下面定义对 $\tau_{x\rightarrowtail\alpha x}$ 的一个更一般的限制，适用于变量 x 出现次数不超过 $i \in \mathbb{N}$ 的方程：

$$\tau^{\le i}_{x\rightarrowtail\alpha x} = \tau_{x\rightarrowtail\alpha x} \cap \{\ ((\mathtt{t}_\ell,\mathtt{t}_r),(w,w')) \mid w,w' \in \Sigma_{\mathbb{X}}^*, |\mathtt{t}_\ell.\mathtt{t}_r|_x \le i\ \}. \tag{7}$$

我们以类似的方式定义 $\tau^{\le i}_{x\rightarrowtail\varepsilon}$、$\tau^{\le i}_{x\mapsto\alpha x}$ 和 $\tau^{\le i}_{x\mapsto\varepsilon}$。

3.4.1. 将 *Nielsen* 变换编码为有理关系

接下来，为了能够在 RMC 框架内操作 $\tau^{\le i}_{x\rightarrowtail\varepsilon}$、$\tau^{\le i}_{x\mapsto\varepsilon}$ 和 $\tau^{\le i}_{x\rightarrowtail\alpha x}$ 编码时，我们需要将它们编码为有理关系。本节定义了有理关系 $\mathcal{T}^{\le i}_{x\rightarrowtail\varepsilon}$ 和 $\mathcal{T}^{\le i}_{x\rightarrowtail\alpha x}$，以完成这一编码。我们通过以下连续步骤得到这些关系：

定义若干中间公式，并以下标表示变换、以上标表示变量数量，例如 $\varphi^{\leq n}_{x\rightarrowtail\varepsilon}$ 和 $\psi^{n}_{x\rightarrowtail\alpha x}$。

我们首先为字符串变量 w、单词约束变量 x以及位置 $k_1,\ldots,k_m$定义一些有用的 MSO(Str) 谓词。

$$ordered(k_1,\ldots,k_m) \triangleq \bigwedge_{1\leq i<m} k_i < k_{i+1} \tag{8}$$

$$alleq^{w}_{x}(k_1,\ldots,k_m) \triangleq \bigwedge_{1\leq i\leq m} w[k_i] = x \tag{9}$$

$$\begin{aligned} occur^{w}_{x}(k_1,\ldots,k_m) \triangleq\ & ordered(k_1,\ldots,k_m) \wedge alleq^{w}_{x}(k_1,\ldots,k_m) \\ & \wedge \forall^{\mathbb{P}} j\Big(w[j]=x \rightarrow \bigvee_{1\leq i\leq m} j = k_i\Big) \end{aligned} \tag{10}$$

我们使用以下 MSO(Str) 公式来定义单个字符串中 x 的出现次数 n 所对应的变换 $x\mapsto\varepsilon$。该公式猜测 n 个 x的位置，然后确保 w' 中的所有符号都正确移位。具体来说，小于 i_1 的位置上的符号会从 w 中原样复制。w 中位于 i_ℓ 和 $i_{\ell+1}$ 之间的位置上的符号，在 w'中向左移 ℓ 个位置。而 w' 中其余的位置则填入 $\square$。

$$\begin{aligned} \psi^{n}_{x\mapsto\varepsilon}(w,w') \triangleq \exists^{\mathbb{P}} i_1,\ldots,i_n\Big(& occur^{w}_{x}(i_1,\ldots,i_n) \wedge \forall^{\mathbb{P}} j(j<i_1 \rightarrow w'[j]=w[j]) \wedge \\ & \bigwedge_{1\leq k<n} \forall^{\mathbb{P}} j((i_k<j<i_{k+1}) \rightarrow w'[j-k]=w[j]) \wedge \\ & \forall^{\mathbb{P}} j(i_n<j \rightarrow w'[j-n]=w[j]) \wedge \\ & \bigwedge_{1\leq k\leq n} w'[\$-k]=\square\Big), \end{aligned} \tag{11}$$

其中，$w'[\$-k]=\square$ 表示公式 $\exists^{\mathbb{P}} r \exists^{\mathbb{P}} s(r=\$\wedge r=s+k\wedge w'[s]=\square)$（参见图4）。我们扩展 $\psi^{n}_{x\mapsto\varepsilon}$，以描述字符串对上的关系：

$$\psi'^{n}_{x\mapsto\varepsilon}(\mathsf{t}_\ell,\mathsf{t}_r,\mathsf{t}'_\ell,\mathsf{t}'_r) \triangleq \bigvee_{0\leq k\leq n} \psi^{k}_{x\mapsto\varepsilon}(\mathsf{t}_\ell,\mathsf{t}'_\ell) \wedge \psi^{n-k}_{x\mapsto\varepsilon}(\mathsf{t}_r,\mathsf{t}'_r) \tag{12}$$

$$\psi'^{\leq n}_{x\mapsto\varepsilon}(\mathsf{t}_\ell,\mathsf{t}_r,\mathsf{t}'_\ell,\mathsf{t}'_r) \triangleq \bigvee_{0\leq k\leq n} \psi'^{k}_{x\mapsto\varepsilon}(\mathsf{t}_\ell,\mathsf{t}_r,\mathsf{t}'_\ell,\mathsf{t}'_r) \tag{13}$$

$$\varphi^{\leq n}_{x\mapsto\varepsilon}(\mathsf{t}_\ell,\mathsf{t}_r,\mathsf{t}'_\ell,\mathsf{t}'_r) \triangleq (\mathsf{t}_\ell[1]=x \vee \mathsf{t}_r[1]=x) \wedge \psi'^{\leq n}_{x\mapsto\varepsilon}(\mathsf{t}_\ell,\mathsf{t}_r,\mathsf{t}'_\ell,\mathsf{t}'_r) \tag{14}$$

接下来，我们定义针对单个字符串中 n 次出现的变换 $x\mapsto\alpha x$。该公式猜测 w 中 x 的 n 个位置。然后，它确保 w' 中的 $i_\ell+\ell$ 位置上是符号 α，并将 w 中的所有其他符号复制到 w' 中的正确位置。特别是，w 中位于 i_ℓ 之间位置上的符号

以及 $i_{\ell+1}$ 在 w' 中向右移动了 ℓ 个位置。

$$\begin{aligned}
&\psi^n_{x\mapsto\alpha x}(w,w') \triangleq \\
&\quad \exists^{\mathbb{P}} i_1,\dots,i_n \Big(occur^w_x(i_1,\dots,i_n) \wedge \forall^{\mathbb{P}} j (j \le i_1 \to w'[j] = w[j]) \wedge \\
&\qquad \bigwedge_{1\le k<n} w'[i_k+k] = \alpha \wedge \forall^{\mathbb{P}} j((i_k < j \le i_{k+1}) \to w'[j+k] = w[j]) \wedge \\
&\qquad\quad w'[i_n+n] = \alpha \wedge \forall^{\mathbb{P}} j (i_n < j \to w'[j+n] = w[j]) \wedge \\
&\qquad \bigwedge_{1\le k\le n} w[\$-k] = \square \Big)
\end{aligned} \tag{15}$$

我们扩展 $\psi^n_{x\mapsto\alpha x}$，以描述字符串对上的关系：

$$\psi'^n_{x\mapsto\alpha x}(\mathsf{t}_\ell,\mathsf{t}_r,\mathsf{t}'_\ell,\mathsf{t}'_r) \triangleq \bigvee_{0\le k\le n} \psi^k_{x\mapsto\alpha x}(\mathsf{t}_\ell,\mathsf{t}'_\ell) \wedge \psi^{n-k}_{x\mapsto\alpha x}(\mathsf{t}_r,\mathsf{t}'_r) \tag{16}$$

$$\psi'^{\le n}_{x\mapsto\alpha x}(\mathsf{t}_\ell,\mathsf{t}_r,\mathsf{t}'_\ell,\mathsf{t}'_r) \triangleq \bigvee_{0\le k\le n} \psi'^{k}_{x\mapsto\alpha x}(\mathsf{t}_\ell,\mathsf{t}_r,\mathsf{t}'_\ell,\mathsf{t}'_r) \tag{17}$$

$$\begin{aligned}
\varphi^{\le n}_{x\mapsto\alpha x}(\mathsf{t}_\ell,\mathsf{t}_r,\mathsf{t}'_\ell,\mathsf{t}'_r) \triangleq &\big((\mathsf{t}_\ell[1] = x \wedge \mathsf{t}_r[1] = \alpha) \vee (\mathsf{t}_\ell[1] = \alpha \wedge \mathsf{t}_r[1] = x)\big) \\
&\wedge \psi'^{\le n}_{x\mapsto\alpha x}(\mathsf{t}_\ell,\mathsf{t}_r,\mathsf{t}'_\ell,\mathsf{t}'_r)
\end{aligned} \tag{18}$$

所构造的公式 $\varphi^{\le n}_{x\mapsto\varepsilon}$ 和 $\varphi^{\le n}_{x\mapsto\alpha x}$ 描述了字母表 $\Sigma_{\mathbb{X},\square}$ 上元数为 4 的正则关系。由于它们是正则的（即长度保持），因此也可以将其解释为复合字母表 $\Sigma^2_{\mathbb{X},\square}$ 上的二元关系。只需对分别对应于 $\varphi^{\le n}_{x\mapsto\varepsilon}$ 和 $\varphi^{\le n}_{x\mapsto\alpha x}$ 的长度保持转换器进行修改，即可轻松实现这种解释：将每个转移 $q \xrightarrow{\{a_1,a_2,a_3,a_4\}} r$ 替换为转移 $q \xrightarrow{\{\left[\begin{smallmatrix}a_1\\a_2\end{smallmatrix}\right],\left[\begin{smallmatrix}a_3\\a_4\end{smallmatrix}\right]\}} r$（状态集合保持不变）。如此将它们解释为二元关系时，它们表示包含以下字符串对的关系（不失一般性，这里只展示 $\varphi^{\le n}_{x\mapsto\varepsilon}$）：

$$\left(\begin{bmatrix} u_1 & \cdots & u_m & u_{m+1} & \cdots & u_n \\ v_1 & \cdots & v_m & \square & \cdots & \square \end{bmatrix} \begin{bmatrix}\square\\\square\end{bmatrix}^i , \begin{bmatrix} w_1 & \cdots & w_k & w_{k+1} & \cdots & w_\ell \\ z_1 & \cdots & z_k & \square & \cdots & \square \end{bmatrix} \begin{bmatrix}\square\\\square\end{bmatrix}^j \right)$$

使得

- $\tau^{\le n}_{x\mapsto\varepsilon}((u_1\dots u_n, v_1\dots v_m)) = (w_1\dots w_\ell, z_1\dots z_k)$ for $\varphi^{\le n}_{x\mapsto\varepsilon}$,
- $\tau^{\le n}_{x\mapsto\alpha x}((u_1\dots u_n, v_1\dots v_m)) = (w_1\dots w_\ell, z_1\dots z_k)$ for $\varphi^{\le n}_{x\mapsto\alpha x}$, and
- $\max(m,n) + i = \max(k,\ell) + j$.

现在考虑 $\varphi^{\le n}_{x\mapsto\varepsilon}$。注意，对于每个 $((u,\ v),\ (w,\ z)) \in \tau^{\le n}_{x\mapsto\varepsilon}$，确实会有一个 $\varphi^{\le n}_{x\mapsto\varepsilon}$ 中与之对应的序对（实际上，这样的序对有无穷多个，它们使用的填充符号数量各不相同）。

为了更接近 $\tau_{x\rightarrowtail\varepsilon}$，我们需要修改 $\varphi^{\le n}_{x\mapsto\varepsilon}$的关系，使其也能对公共前缀执行裁剪。我们通过采用识别 $\varphi^{\le n}_{x\mapsto\varepsilon}$的（长度保持）双轨变换器 $T^{\le n}_{x\mapsto\varepsilon}$ 来进行这一修改（根据命题 2，它可以被构造出来）。此外，我们还创建了一个对其输入执行裁剪的（长度不保持）转换器 T_{trim}；通过一个将符号前缀替换为

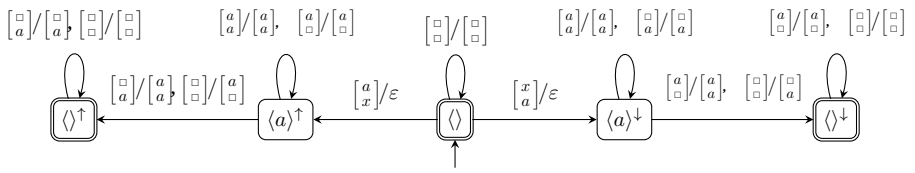


图8：实现变量集 $\mathbb{X}=\{x\}$ 上、字母表为 $\Sigma=\{a\}$ 的关系 $T^{\leq 1}_{x\mapsto\varepsilon}$ 的转换器示例。转换器状态的名称表示转换器记住并要在由符号 $\uparrow$ 和 $\downarrow$给出的带上输出的符号。

算法 1： 使用 RMC 求解字符串约束 φ

输入： 公式 φ （初始集合）的编码 $\mathcal{I}$，
变换器 $\mathcal{T}$ $x\rightarrowtail\alpha x$, $\mathcal{T}_{x\rightarrowtail\varepsilon}$, and
目标集合 $\mathcal{D}$

输出： 若 φ 可满足，则为 φ 的一个模型；否则为假

```
可达_0 := I;
已处理 := ∅;
T := T_{x↣αx} ∪ T_{x↣ε};
i := 0;
当 可达_i ⊈ 已处理 则
    如果 D ∩ 可达_i ≠ ∅ 则
        返回 提取模型({T}_{j∈{0,...,i}}, D, 可达_0, ..., 可达_i);
    已处理 := 已处理 ∪ 可达_i;
    可达_{i+1} := 饱和 ∘ T(可达_i);          // 饱和• ∘ T(可达_i)
    i++;
返回 false;
```

形如 $\begin{bmatrix}\beta\\\beta\end{bmatrix}$ 的内容为 ε的双状态变换器，可以很容易地实现这一点，对于 $\beta\in\Sigma_{\mathbb{X},\square}$。通过组合这两个转换器，我们得到 $T^{\leq n}_{x\rightarrowtail\varepsilon}=T_{trim}\circ T^{\leq n}_{x\mapsto\varepsilon}$。图8展示了一个实现 $T^{\leq 1}_{x\mapsto\varepsilon}$的转换器示例。

我们可以以类似的方式对 $\varphi^{\leq n}_{x\mapsto\alpha x}$ 重复前面的推理，从而得到（长度不保持的）转换器 $T^{\leq n}_{x\rightarrowtail\alpha x}$。

引理5。 当且仅当 $\tau^{\leq n}_{x\rightarrowtail\varepsilon}((u_1\dots u_m, v_1\dots v_m))=(w_1\dots w_\ell, z_1\dots z_k)$ 成立

$$\left(\begin{bmatrix}u_1&\cdots&u_m&u_{m+1}&\cdots&u_n\\v_1&\cdots&v_m&\square&\cdots&\square\end{bmatrix}\begin{bmatrix}\square\\\square\end{bmatrix}^i,\ \begin{bmatrix}w_1&\cdots&w_k&w_{k+1}&\cdots&w_\ell\\z_1&\cdots&z_k&\square&\cdots&\square\end{bmatrix}\begin{bmatrix}\square\\\square\end{bmatrix}^j\right)\in\mathcal{L}(T^{\leq n}_{x\rightarrowtail\varepsilon})$$

对于某些 $i,j\in\mathbb{N}$。

此外，当且仅当 $\tau^{\leq n}_{x\rightarrowtail\alpha x}((u_1\dots u_m, v_1\dots v_m))=(w_1\dots w_\ell, z_1\dots z_k)$ 成立

$$\left(\begin{bmatrix}u_1&\cdots&u_m&u_{m+1}&\cdots&u_n\\v_1&\cdots&v_m&\square&\cdots&\square\end{bmatrix}\begin{bmatrix}\square\\\square\end{bmatrix}^i,\ \begin{bmatrix}w_1&\cdots&w_k&w_{k+1}&\cdots&w_\ell\\z_1&\cdots&z_k&\square&\cdots&\square\end{bmatrix}\begin{bmatrix}\square\\\square\end{bmatrix}^j\right)\in\mathcal{L}(T^{\leq n}_{x\rightarrowtail\alpha x})$$

对于某些 $i,j\in\mathbb{N}$。

证明。 证明可由上述转换器 $T^{\leq n}_{x\rightarrowtail\alpha x}$和 $T^{\leq n}_{x\rightarrowtail\varepsilon}$的构造得出。

□

函数提取模型($\{\mathcal{T}_j\}_{j\in\{0,\dots,i\}}$，$\mathcal{D}$，可达$_0$，……，可达$_i$)

1 $I(a) := a$ 对每个 $a \in \Sigma$，$I(x) := \varepsilon$ 对每个 $x \in \mathbb{X}$;
2 令 $w_i \in \mathcal{D} \cap$可达$_i$;
3 **对** $\ell = i$ **递减至**1 **执行**
4 　令 $w_{\ell-1} \in \mathcal{T}_\ell^{-1}(w_\ell) \cap$可达$_{\ell-1}$;
5 　令 ρ 为满足 $w_\ell \in w_{\ell-1}[\rho]$ 的规则;
6 　**如果** $\rho = y \rightarrowtail \alpha y$ **那么**
7 　　$I(y) := I(\alpha) \cdot I(y)$;
8 **返回** I;

3.4.2. 二次方程的 RMC

在算法 1 中，我们给出一种使用 RMC 求解字符串约束的高层次算法。该算法由以下参数确定：(i) 编码公式 φ（初始集合）的正则语言 $\mathcal{I}$，(ii) 由转换器 $\mathcal{T}_{x\rightarrowtail\alpha x}$和 $\mathcal{T}_{x\rightarrowtail\varepsilon}$给出的有理关系，以及 (iii) 目标集合 $\mathcal{D}$（同样以正则语言给出）。该算法通过迭代展开第 3 行计算出的转移关系 $\mathcal{T}$，尝试求解 RMC 问题（$\mathcal{I}$，$\mathcal{T}_{x\rightarrowtail\alpha x} \cup \mathcal{T}_{x\rightarrowtail\varepsilon}$，$\mathcal{D}$），并寻找来自 $\mathcal{D}$的元素 w_i。如果在集合 可达配置$_i$中找到这样的元素，我们就调用函数 提取模型，从 w_i开始进行反向运行，以提取原始字方程的一个模型；具体做法是计算关于转换器 $\mathcal{T}_{x\rightarrowtail\alpha x}$和 $\mathcal{T}_{x\rightarrowtail\varepsilon}$的原像 w_{i-1}、……、w_1（对每个 w_j，将其限制在可达配置 j 范围内），同时根据所执行的变换更新变量的值。请注意，提取模型 使用了更通用的接口，允许为每个反向步骤指定一个转换器（函数 提取模型的第 4 行）。这一点将在第 4 节中用到；此处我们只传递 i 个 $\mathcal{T}$的副本。算法 1 还对可达配置集合进行饱和，其定义如下：

$$satur(L) = \left\{ u \;\middle|\; w \in L, w \in u.\left[\begin{smallmatrix}\square\\ \square\end{smallmatrix}\right]^* \right\}. \tag{19}$$

直观地说，饱和(L) 会从 L 中所有词末尾移除一些填充符号（也可能一个都不移除）。如果 L 是正则语言，则 饱和(L) 也是正则的（给定一个表示 L 的有限自动机，我们可以通过对其转移关于填充符号进行饱和，得到 饱和 (L)）。我们对可达配置集合进行饱和，是因为希望保留最短的词（即不含填充符号的词）——例如，变换器 $\mathcal{T}^{eq}_{x\rightarrowtail\varepsilon}$ 不必生成所有最短的词。

算法 1 遵循广度优先搜索 （BFS）策略：从初始集合 $\mathcal{I}$ 出发，我们同时对 $\mathcal{I}$ 的所有元素应用变换器 $\mathcal{T}_{x\rightarrowtail\alpha x}$ 和 $\mathcal{T}_{x\rightarrowtail\varepsilon}$，然后反复对结果应用这些变换器。如果我们一次对 $\mathcal{I}$ 的一个元素应用变换器，这就对应于以广度优先的方式应用变换器。

我们对该算法的首次实例化用于检查单个二次词方程 eq：$\mathsf{t}_\ell = \mathsf{t}_r$ 的可满足性。我们将 RMC 问题实例化为元组（$\mathcal{I}^{eq}$，$\mathcal{T}^{eq}_{x\rightarrowtail\alpha x} \cup \mathcal{T}^{eq}_{x\rightarrowtail\varepsilon}$，$\mathcal{D}^{eq}$），其中

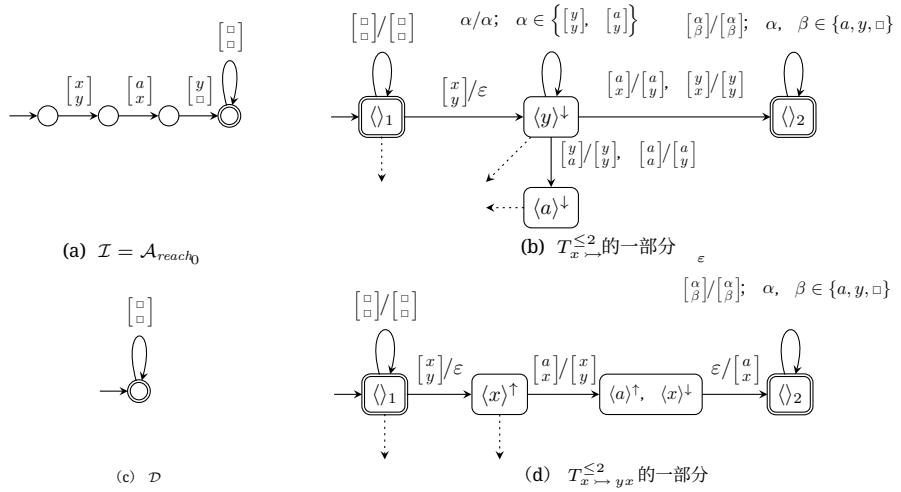


图9：用于求解字方程 $xay = yx$ 的算法 1 的输入。 $\mathcal{T}^{eq}_{x \rightarrowtail \varepsilon}$ 和 $\mathcal{T}^{eq}_{x \rightarrowtail yx}$ 的规模会大得无法承受，因此我们只给出转换器 $T^{\leq 2}_{x \rightarrowtail \varepsilon}$ 和 $T^{\leq 2}_{x \rightarrowtail yx}$ 的相关部分。某些转移使用了具有显然含义的简写表示法。

$$\mathcal{I}^{eq} = \mathsf{encode}(\mathsf{t}_\ell, \mathsf{t}_r) \qquad \mathcal{T}^{eq}_{x \rightarrowtail \alpha x} = \bigcup_{y \in \mathbb{X}, \alpha \in \Sigma_{\mathbb{X}}} T^{\leq 2}_{y \rightarrowtail \alpha y} \qquad \mathcal{D}^{eq} = \left\{ \begin{bmatrix} \square \\ \square \end{bmatrix} \right\}^*$$

$$\mathcal{T}^{eq}_{x \rightarrowtail \varepsilon} = \bigcup_{y \in \mathbb{X}} T^{\leq 2}_{y \rightarrowtail \varepsilon}$$

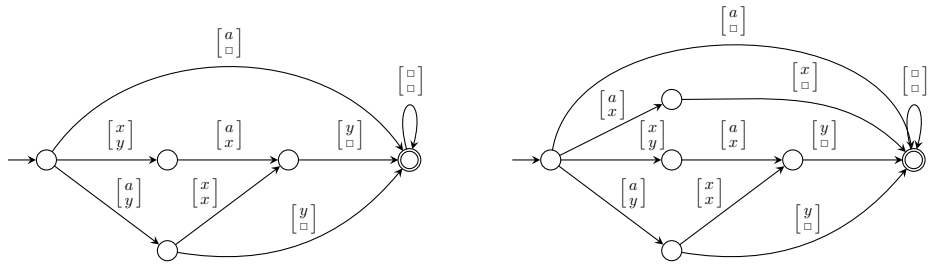
引理6。 若 *eq*： $\mathsf{t}_\ell = \mathsf{t}_r$ 是二次的，则将算法 *1* 具体化为 $(\mathcal{I}^{eq},\ \mathcal{T}^{eq}_{x \rightarrowtail \alpha x} \cup \mathcal{T}^{eq}_{x \rightarrowtail \varepsilon},\ \mathcal{D}^{eq})$ 后是可靠的、完备的且终止的。

证明。 我们将 Nielsen 证明图的节点编码为字符串。初始节点 $\mathsf{t}_\ell = \mathsf{t}_r$ 对应于来自 $\mathcal{I}^{eq}$ 的字符串。由于填充，最终节点 $\varepsilon = \varepsilon$ 对应于来自 $\mathcal{D}^{eq}$ 的字符串。关系 $\mathcal{T}^{eq}_{x \rightarrowtail \alpha x}$ 和 $\mathcal{T}^{eq}_{x \rightarrowtail \varepsilon}$ 实现了 Nielsen规则 $x \rightarrowtail \alpha x$ 和 $x \rightarrowtail \varepsilon$。根据引理 3，我们知道 Nielsen 证明图是有限的。由于我们的方法实现了 广度优先搜索（BFS）策略，因此它是可靠的、完备的，并且会终止。 □

例4。 考虑第3.1节中的字方程*eq*: $xay = yx$ 。我们对编码后的方程 $\mathcal{I} = \mathsf{encode}(xay, yx) = \begin{bmatrix} x & a & y \\ y & x & \square \end{bmatrix} \begin{bmatrix} \square \\ \square \end{bmatrix}^*$应用算法 1。该算法的输入见图9。在主循环的第一次迭代中，正则集$processed$由其最小有限自动机 $\mathcal{A}_{reach_1}$ 表示，如图10a所示（该有限自动机也对应于$reach_1$ ，并表示例4中的集合 S_1 ）。特别地，例如，

$$\left(\begin{bmatrix} x & a & y \\ y & x & \square \end{bmatrix}, \begin{bmatrix} a & y & \square \\ y & \square & \square \end{bmatrix} \right) \in \mathcal{T}^{eq}_{x \rightarrowtail \varepsilon}.$$

饱和后，我们得到 $\begin{bmatrix} a & y \\ y & \square \end{bmatrix} \in$ 可达$_1$。在第二次（也是最后一次）迭代中，集合 已处理 由图10b中的最小有限自动机 $\mathcal{A}_{reach_2}$ 给出（它也对应于可达$_2$ 和可达$_3$，以及例4中的集合 S_2 ）。



(a) $\mathcal{A}_{reach_1}$ (b) $\mathcal{A}_{reach_2}$

图10：使用算法 1 求解字方程 $xay = yx$ 时，分别表示一步和两步内可达的配置集合的有限自动机 $\mathcal{A}_{reach_1}$、 $\mathcal{A}_{reach_2}$ 示例。

由于可达$_3 \subseteq$ 已处理，算法以 false 终止，从而证明eq_1不可满足。

□

4. 使用 RMC 求解字方程组

在上一节中，我们描述了如何在 RMC 框架中求解单个二次字方程。本节将重点介绍如何扩展这一方法，以处理形如以下形式的字方程组

$$\Phi\colon \mathsf{t}_\ell^1 = \mathsf{t}_r^1 \wedge \mathsf{t}_\ell^2 = \mathsf{t}_r^2 \wedge \ldots \wedge \mathsf{t}_\ell^n = \mathsf{t}_r^n. \tag{20}$$

第一步是将系统 Φ 编码为正则语言。为此，我们将编码函数扩展到字方程组，定义为

$$\mathsf{encode}(\Phi) = \mathsf{encode}(\mathsf{t}_\ell^1, \mathsf{t}_r^1).\left\{\begin{bmatrix}\#\\\#\end{bmatrix}\right\}.\ldots.\left\{\begin{bmatrix}\#\\\#\end{bmatrix}\right\}.\mathsf{encode}(\mathsf{t}_\ell^n, \mathsf{t}_r^n), \tag{21}$$

其中 # 是分隔符，$\# \notin \Sigma_{\mathbb{X},\,\square}$。由引理 4 可知，对于所有 $1 \le i \le n$，$\mathsf{encode}(\mathsf{t}_\ell^i,\ \mathsf{t}_r^i)$ 都是正则的。此外，由于正则语言在连接运算下封闭（命题 1），集合 $\mathsf{encode}(\Phi)$ 也为正则。由于现在每个方程都由分隔符隔开，我们需要将目标集合扩展为 $\left\{\begin{bmatrix}\square\\\square\end{bmatrix},\ \begin{bmatrix}\#\#\\\end{bmatrix}\right\}^*$。

对于转移关系，我们需要将上一节中的 $\tau_{x \rightarrowtail \alpha x}^{\le i}$ 和 $\tau_{x \rightarrowtail \varepsilon}^{\le i}$ 扩展为支持分隔符。规则 $x \rightarrowtail \alpha x$ 应用于方程组时，可描述如下：将规则 $x \rightarrowtail \alpha x$ 应用于第一个非空方程，并根据代换 $x \mapsto \alpha x$修改其余方程。无论其他方程的首个符号是什么，都会对它们执行代换。规则 $x \rightarrowtail \varepsilon$的过程类似。连续应用这些规则可以减少方程的数量，进而得到编码中的一个字符串，其前缀来自 $\left\{\begin{bmatrix}\square\\\square\end{bmatrix},\ \begin{bmatrix}\#\#\\\end{bmatrix}\right\}^*$。在编码后的方程组上实现 $x \rightarrowtail \alpha x$ 或 $x \rightarrowtail \varepsilon$ 的关系会跳过此前缀。形式上，对于每个方程中每个变量的出现次数至多为 i 的方程组，规则 $x \rightarrowtail \alpha x$ 由以下关系给出：

$$T_{x \rightarrowtail \alpha x}^{\wedge eq, i} = T_{skip}.T_{x \rightarrowtail \alpha x}^{\le i}.\left(\left\{\begin{bmatrix}\#\\\#\end{bmatrix} \mapsto \begin{bmatrix}\#\\\#\end{bmatrix}\right\}.(T_{trim} \circ S_{x \mapsto \alpha x}^{\le i})\right)^*, \tag{22}$$

其中 $T_{skip} = \left\{\left[\begin{smallmatrix}\square\\\square\end{smallmatrix}\right] \mapsto \left[\begin{smallmatrix}\square\\\square\end{smallmatrix}\right], \left[\#\#\right] \mapsto \left[\#\#\right]\right\}^*$，而 $S^{\leq i}_{x\mapsto\alpha x}$ 是表示公式 $\psi'^{\leq i}_{x\mapsto\alpha x}$ 的二进制变换器，该公式来自第 3.4.1 节（它不会查看带上的第一个符号）。关系 $T^{\wedge eq}_{,\ x\rightarrowtail\varepsilon}{}^{i}$ 以类似方式定义。根据构造以及有理关系的闭包性质（参见命题 1），显然 $T^{\wedge eq}_{,\ x\rightarrowtail\alpha x}{}^{i}$ 和 $T^{\wedge eq}_{,\ x\rightarrowtail\varepsilon}{}^{i}$ 都是有理关系。

4.1. 二次情形

当 Φ 为二次的时，其可满足性问题可归约为一个 RMC 问题（$\mathcal{I}^{\wedge eq}_{\Phi}$, $\mathcal{T}^{\wedge eq}_{x\rightarrowtail\alpha x}\cup\mathcal{T}^{\wedge eq}_{x\rightarrowtail\varepsilon}$, $\mathcal{D}^{\wedge eq}$），其实例化了算法 1，其中

$$\mathcal{I}^{\wedge eq}_{\Phi} = \mathsf{encode}(\Phi) \qquad \mathcal{T}^{\wedge eq}_{x\rightarrowtail\alpha x} = \bigcup_{y\in\mathbb{X},\alpha\in\Sigma_{\mathbb{X}}} T^{\wedge eq,2}_{y\rightarrowtail\alpha y} \qquad \mathcal{D}^{\wedge eq} = \left\{\left[\begin{smallmatrix}\square\\\square\end{smallmatrix}\right], \left[\begin{smallmatrix}\#\\\#\end{smallmatrix}\right]\right\}^*$$
$$\mathcal{T}^{\wedge eq}_{x\rightarrowtail\varepsilon} = \bigcup_{y\in\mathbb{X}} T^{\wedge eq,2}_{y\rightarrowtail\varepsilon}$$

$\mathcal{T}^{\wedge eq}_{x\rightarrowtail\alpha x}$ 和 $\mathcal{T}^{\wedge eq}_{x\rightarrowtail\varepsilon}$ 的有理性直接由命题 1 得出。此外，我们还需要修改算法 1，使得在第 9 行中，我们将 饱和替换为定义如下的 饱和$^\bullet$：

$$satur^\bullet(L) = \left\{u_1.u_2\dots u_k \;\middle|\; w\in L, w\in u_1.\left[\begin{smallmatrix}\square\\\square\end{smallmatrix}\right]^+.u_2.\left[\begin{smallmatrix}\square\\\square\end{smallmatrix}\right]^+\dots u_k \text{ for some } k\in\mathbb{N}\right\}.$$

直观地说，饱和$^\bullet(L)$ 修改 饱和 ，以考虑到我们现在处理的是编码成单个词的多个字方程，而这些方程由分隔符分隔。现在，饱和$^\bullet$ 不仅会移除词尾的填充（当 $k=1$ 时），还会移除中间的填充。

以下引理总结了我们针对二次词方程系统的过程的可靠性和完备性。

引理 7. 若 Φ 是二次的，则将算法 *1* 实例化为 $(\mathcal{I}^{\wedge eq}_{\Phi}, \mathcal{T}^{\wedge eq}_{x\rightarrowtail\alpha x}\cup\mathcal{T}^{\wedge eq}_{x\rightarrowtail\varepsilon}, \mathcal{D}^{\wedge eq})$ 后是可靠、完备且终止的。

证明. 我们将 Nielsen 证明图的节点编码为字符串。初始节点 Φ 对应于来自 $\mathcal{I}^{\wedge eq}_{\Phi}$ 的一个字符串（合取可以看作分隔符 #）。由于填充，最终节点 $\varepsilon = \varepsilon\wedge\dots\wedge\varepsilon = \varepsilon$ 对应于来自 $\mathcal{D}^{\wedge eq}$ 的一个字符串。关系 $\mathcal{T}^{\wedge eq}_{x\rightarrowtail\alpha x}$ 和 $\mathcal{T}^{\wedge eq}_{x\rightarrowtail\varepsilon}$ 实现了 Nielsen 规则 $x\rightarrowtail\alpha x$ 和 $x\rightarrowtail\varepsilon$。根据引理 3，Nielsen 证明图是有限的（因此潜在的最终节点位于有限深度处）。由于我们的方法实现了 BFS 策略，因此它既可靠、完备，又能终止。

□

4.2. 一般情形

现在我们来考虑系统 Φ不是二次的这一一般情形。在本节中，我们将证明，这一一般情形也可以归约到扩展版 RMC。

我们首先对一般字符串约束系统应用算法 2，以得到一个可满足性等价的三次字方程组 Φ'。如果变换的输入是一个含有 n 个符号的方程组，那么在最坏情况下，变换的输出将包含 $\frac{n}{2}$ 个额外的字方程和 $\frac{n}{2}$ 个额外的文字，因此该变换是线性的。然后，我们可以使用转移关系 $T^{\wedge eq}_{,\ x\rightarrowtail\alpha x}{}^{3}$ 和 $T^{\wedge eq}_{,\ x\rightarrowtail\varepsilon}{}^{3}$ 来构造编码系统 Φ'的变换。

算法 2： 转换为三次方程组

输入： 字方程组 Φ

输出： 可满足性等价的三次字方程组 Ψ

$\Psi := \Phi$;

当 $\exists x \in \mathbb{X}$ 使得 x 在 Ψ 中出现次数超过三次**执行**

  将 Φ 中的 x 的两次出现替换为一个新词变量 x'，以便获得一个新的系统 Ψ';

  $\Psi := (\Psi' \wedge x = x')$;

返回 Ψ;

引理 8。 任何字方程组都可以通过算法 *2* 转换为可满足性等价的三次字方程组。

证明。 设 Φ 为输入的字方程组。注意，在算法 2 的每次迭代中，变量 x 的出现次数减少一次，并引入一个恰好出现三次的新变量 x'。

□

我们还需要解决一个问题，即确保算法的每一步处理的都是三次字方程组。某种类型为 $x \rightarrowtail yx$ 的变换可能会使变量 y 的出现次数增加一次；因此，如果变换前 y 已经出现三次，变换后的系统就不再是三次的了。具体来说，设三次字方程组 $x.\mathsf{t}_\ell = y.\mathsf{t}_r \wedge \Phi$，其中 x 和 y 是不同的字变量，t_ℓ 和 t_r 是字项。若应用变换 $x \rightarrowtail yx$，则会得到 $x(\mathsf{t}_\ell[x \mapsto yx]) = \mathsf{t}_r[x \mapsto yx] \wedge \Phi[x \mapsto yx]$。注意：(i) y 的出现次数首先减少一次，因为右侧的 $x.\mathsf{t}_\ell = y.\mathsf{t}_r$ 中第一个 y 被删除；(ii) y 的出现次数最多增加两次，因为 t_ℓ、t_r 和 Φ 中最多有两处 x。因此，经过变换 $x \rightarrowtail yx$ 后，三次字方程组可能会变成*(y-)*四次字方程组（变量 y 最多出现四次，任何其他变量最多出现三次）。为此，我们需要在每次变换后都应用向三次系统的转换。

给定一个新的变量 v，我们用 $\mathcal{C}_v$ 表示从单四次字方程组到使用新变量 v 的三次方程组的变换。

引理 9. 对编码后的单四次方程组执行变换 $\mathcal{C}_v$ 的关系 $T_{\mathcal{C}_v}$ 是有理的。

证明。 我们将说明如何构造一个转换器，实现从单四次字方程组到三次字方程组的变换。

第一步，我们创建转换器 $\mathcal{T}^{sq}_{x,x_i}$，它只接受作为 x 次字方程组编码的输入。我们可以通过使用状态来追踪变量的出现次数来实现这一点（只需计数到四即可）。对于 x 次字方程组的编码，转换器 $\mathcal{T}^{sq}_{x,x_i}$ 返回的编码，是将输入中 x 的前两次出现替换为 x_i，并在末尾连接语言 $\left[\#\#\right]\left[\begin{smallmatrix} x \\ x_i \end{smallmatrix}\right]\left[\begin{smallmatrix} \square \\ \square \end{smallmatrix}\right]^*$ 后得到的结果。

算法3： 使用 RMC 求解一般字符串约束 φ

输入： 公式 φ 的编码 $\mathcal{I}$（初始集合），
索引的有序集合 $\mathbb{V} = \{v_1, v_2, \dots\}$，
参数化变换器 $\mathcal{T}^v$ $_{x \rightarrowtail \alpha x}$，$\mathcal{T}^v_{x \rightarrowtail \varepsilon}$ 其中 $v \in \mathbb{V}$，且
目标集合 $\mathcal{D}$

输出： 若 φ 可满足，则为 φ 的一个模型；否则为 false

1 可达$_0$ ：$=\mathcal{I}$;
2 已处理 ：$=\emptyset$;
3 $\mathcal{T}^v := \mathcal{T}^v_{x \rightarrowtail \alpha x} \cup \mathcal{T}^v_{x \rightarrowtail \varepsilon}$;
4 i:= 0;
5 **当** $reach_i \not\subseteq$已处理 **时执行**
6 **如果** $\mathcal{D} \cap reach_i \neq \emptyset$ **则**
7 **返回** $ExtractModel(\{\mathcal{T}^{v_j}\}_{j\in\{0,\dots,i\}}, \mathcal{D}, reach_0, \dots, reach_i)$;
8 已处理 := 已处理 $\cup reach_i$;
9 $reach_{i+1}$:= 饱和$^\bullet \circ \mathcal{T}^{v_i}(reach_i)$;
10 $\mathbb{X} := \mathbb{X} \cup \{v_i\}$;
11 i++;
12 **返回**false;

第二步，我们创建转换器 $\mathcal{T}_{cub}$，它只接受三次字方程组的编码，并返回相同的编码。现在我们有

$$T_{\mathcal{C}_v} = \mathcal{L}(\mathcal{T}_{cub}) \cup \bigcup_{x \in \mathbb{X}} \mathcal{L}(\mathcal{T}^{sq}_{x,v}). \tag{23}$$

该引理随后由命题 1 得证。 □

为了用（修正的）RMC 来表示求解字符串约束系统 Φ，我们首先将 Φ（使用算法 2）转换为一个等可满足的三次系统 Φ'。字方程组 Φ的可满足性可以归约为一个修正的 RMC 问题（$\mathcal{I}^{\wedge eq}_\Phi$，$\mathcal{T}^{v_i,}$ $_{x \rightarrowtail \alpha x} \cup \mathcal{T}^{v_i,}$ $_{x \rightarrowtail \varepsilon}$，$\mathcal{D}^{\wedge eq}$），通过使用以下组件实例化算法 1：

$$\mathcal{I}^{\wedge eq}_\Phi = \mathsf{encode}(\Phi') \qquad \mathcal{T}^{v,\wedge eq}_{x \rightarrowtail \alpha x} = T_{\mathcal{C}_v} \circ \bigcup_{y \in \mathbb{X}, \alpha \in \Sigma_\mathbb{X}} T^{\wedge eq,3}_{y \rightarrowtail \alpha y} \qquad \mathcal{D}^{\wedge eq} = \left\{ \left[\begin{smallmatrix} \square \\ \square \end{smallmatrix}\right], \left[\begin{smallmatrix} \# \\ \# \end{smallmatrix}\right] \right\}^*$$

$$\mathcal{T}^{v,\wedge eq}_{x \rightarrowtail \varepsilon} = T_{\mathcal{C}_v} \circ \bigcup_{y \in \mathbb{X}} T^{\wedge eq,3}_{y \rightarrowtail \varepsilon}$$

对于修正的 RMC 算法，我们需要假设 $\mathbb{V} = \{v_1, v_2, \dots\} \cap \Sigma_\mathbb{X} = \emptyset$，其中 $\mathbb{V}$ 是一个新索引变量集合。我们还需要将算法 1 的第 3 行更新为 $\mathcal{T}^v$:
$= \mathcal{T}^v_{x \rightarrowtail \alpha x} \cup \mathcal{T}^v_{x \rightarrowtail \varepsilon}$，并将第 9 行更新为 $reach_{i+1} := \mathcal{T}^{v_i}(reach_i)$； $\mathbb{X} := \mathbb{X} \cup \{v_i\}$; 以允许在每次迭代中使用一个新变量 v_i（这里，在算法的每次迭代中，$\mathcal{T}^v$ 都将用 v 参数的新值实例化）。完整算法如算法3所示。 $\mathcal{T}^{v,}$ $_{x \rightarrowtail \alpha x}$和 $\mathcal{T}^{v,}$ $_{x \rightarrowtail \varepsilon}$的合理性直接由命题 1 得出。

引理 10。 若 Φ 是三次的，则将算法*3*实例化为$(\mathcal{I}^{\wedge eq}_\Phi, \mathbb{V}, \mathcal{T}^{v,}$ $_{x \rightarrowtail \alpha x} \cup \mathcal{T}^{v,x \rightarrowtail \varepsilon}$, $\mathcal{D}^{\wedge eq})$ 是可靠的。

证明。 我们再次将 Nielsen 证明图的节点编码为字符串。初始节点和最终节点分别对应于编码后的初始语言 $\mathcal{I}_{\Phi}^{\wedge eq}$和最终语言 $\mathcal{D}^{\wedge eq}$中的字符串。关系 $\mathcal{T}^{v,}{}_{x\rightarrowtail\alpha x}^{\wedge eq}$和 $\mathcal{T}^{v,}{}_{x\rightarrowtail\varepsilon}^{\wedge eq}$实现了 Nielsen规则 $x\rightarrowtail\alpha x$ 和 $x\rightarrowtail\varepsilon$。对于任意字方程组，Nielsen 证明图可能是无限的。然而，由于变换 $\mathcal{C}_v$保持可满足性，该过程是可靠的。 □

完备性。 由于前一种方法在每一步都可能引入一个新的方程（因为转换器 T_{C_v} 会将方程组转换为三次方程），因此通常无法保证完备性。为了获得针对三次方程的可靠且完备的过程，必须使用重写符号数量不断增加的转换器。具体而言，我们需要使用实现以下关系的转换器：

$$\mathcal{T}_{x\rightarrowtail\alpha x}^{i,\wedge eq\star}=\bigcup_{y\in\mathbb{X},\alpha\in\Sigma_{\mathbb{X}}}T_{y\rightarrowtail\alpha y}^{\wedge eq,2^{i+1}}\qquad\qquad\mathcal{T}_{x\rightarrowtail\varepsilon}^{i,\wedge eq\star}=\bigcup_{y\in\mathbb{X}}T_{y\rightarrowtail\varepsilon}^{\wedge eq,2^{i+1}}$$

其中 $i\in\mathbb{N}_1$ 为迭代次数（此外，还需要跳过算法3中的第10行）。每一步之后，系统中某个变量出现次数的最大值在最坏情况下会翻倍。因此，在第 i 步中，系统中某个变量的出现次数最多可达 2^{i+1}，这可以由 $\mathcal{T}^{i,}{}_{x\rightarrowtail\alpha x}^{\wedge eq\star}$和 $\mathcal{T}^{i,}{}_{x\rightarrowtail\varepsilon}^{\wedge eq\star}$处理。

引理11。 若 Φ 是三次的，则将算法3应用于 $(\mathcal{I}_{\Phi}^{\wedge eq},\mathbb{N},\mathcal{T}^{i,}{}_{x\rightarrowtail\alpha x}^{\wedge eq\star}\cup\mathcal{T}^{i,}{}_{x\rightarrowtail\varepsilon}^{\wedge eq\star},\mathcal{D}^{\wedge eq})$并按上述方式修改后，所得算法是可靠且完备的。

证明。 与前述情况一样，我们将 Nielsen 证明图的节点编码为字符串。由于转换器 $T^{\wedge eq,}{}_{y\rightarrowtail\alpha y}^{\ell}$能正确实现 Nielsen 规则$y\rightarrowtail\alpha y$，适用于每个变量至多出现 ℓ 次的系统，因此只需证明，在算法 3 的第 i 次迭代中，每个变量的出现次数都有上界 2^{i+1}。算法3的 BFS 策略和引理 3 保证了可靠性和完备性。

考虑一个方程组 Φ，其中每个变量至多出现 n 次。应用形式为 $y\rightarrowtail\varepsilon$ 的规则不会增加 y的出现次数。对于形式为 $y\rightarrowtail\alpha y$ 且满足$\alpha\in\mathbb{X}$的规则，情况则有所不同。此外，注意到在修改后的系统中，y 至多出现 n 次，而 α 出现了 $(n-1)+(n-1)=2n-2$ 次（每个 y都被替换为 αy，并且从修改后的系统中移除了 α 的第一次出现）。因此，算法3第 i次迭代中变量出现次数的最大值有上界 2^{i+1}（注意这是一个宽松的上界，因为最坏情况下出现次数会增加到 $2n-2\leq 2n$），前提是 Φ 是三次的。

□

终止性。 由于尼尔森变换无法保证一般情形下终止，我们的算法也无法保证终止。未来的工作将研究完备算法（例如 Makanin 算法（Makanin（1977））））的可能符号编码。

5. 处理字符串约束的布尔组合

在本节中，我们将把处理字方程合取的过程扩展为能够处理其任意布尔组合的过程。一种显而易见的方法是，将我们在

第 3 节和第 4 节中提出的解法与标准的基于 DPLL(T) 的求解器结合起来，并用我们的过程处理字符串理论。不过，我们可以利用本节提出的编码，通过我们的过程求解整个公式。尽管我们并不指望所提出的解法能够与高度优化的基于 DPLL(T) 的求解器相竞争，但它（同时考虑第 6 节中的扩展）通过为相当一般的一类约束提供统一的、基于自动机的编码，使我们的框架更加稳健。例如，当尝试通过使用抽象（参见 Bouajjani 等人（2012））来加速终止性，或按照 Neider 和 Jansen（2013）的思路学习不变式时，仍可在 RMC 框架内以统一的方式处理整个约束系统的编码。

字方程的否定可以按照标准方式处理。例如，我们可以采用 Abdulla 等人（2014）给出的方法，将否定的字方程 $\mathsf{t}_\ell \neq \mathsf{t}_r$ 转换为以下字符串约束：

$$\bigvee_{c\in\Sigma} (\mathsf{t}_\ell = \mathsf{t}_r \cdot cx \vee \mathsf{t}_\ell \cdot cx = \mathsf{t}_r) \vee \bigvee_{c_1,c_2\in\Sigma, c_1\neq c_2} (\mathsf{t}_\ell = yc_1x_1 \wedge \mathsf{t}_r = yc_2x_2) \quad (24)$$

该约束的第一部分表示，t_ℓ 是 t_r 的真前缀，或者反之。第二部分表示，t_ℓ 和 t_r 具有公共前缀 y，并在接下来的字符 c_1 和 c_2处开始不同。对于通过 $\wedge$ 和 $\vee$连接的字方程，我们应用分配律，得到一个与原公式等价的合取范式（CNF）公式，其规模至多呈原公式规模的指数增长。注意，我们不能使用 Tseitin 变换 Tseitin（1983），因为它可能引入新的否定变量，而使用公式（24）消除这些变量会破坏合取范式的形式。

现在，我们来关注如何使用（修改后的）RMC，表示由任意布尔组合的字方程构成的字符串约束 Φ的求解。我们首先利用式（24）消去 Φ 中的不等式，然后将不含不等式的系统转换为合取范式，最后应用引理 8 中的过程，将合取范式公式转换为等可满足的三次合取范式 Φ'。为了用 RMC 判定 Φ' 的可满足性，转移关系和目标集合均与上一节（一般情形）相同。唯一的区别在于初始配置，因为该系统不再是项的合取，而是合取范式的一般公式。为此，我们将 encode 的定义扩展到子句 c：$(\mathsf{t}_\ell^1 = \mathsf{t}_r^1 \vee \ldots \vee \mathsf{t}_\ell^n = \mathsf{t}_r^n)$，定义 $\mathsf{encode}(c) = \bigcup_{1\leq j\leq n} \mathsf{encode}(\mathsf{t}_\ell^j, \mathsf{t}_r^j)$。于是，$\Phi'$ 的初始配置定义为

$$\mathcal{I}_{\Phi'}^{\wedge\vee eq} = \mathsf{encode}(c_1).\left\{\begin{bmatrix}\#\\\#\end{bmatrix}\right\}.\ldots.\left\{\begin{bmatrix}\#\\\#\end{bmatrix}\right\}.\mathsf{encode}(c_m), \quad (25)$$

其中 Φ'的形式为 Φ'：$c_1 \wedge \ldots \wedge c_m$，且每个子句 c_i：$(\mathsf{t}_\ell^1 = \mathsf{t}_r^1 \vee \ldots \vee \mathsf{t}_\ell^{n_i} = \mathsf{t}_r^{n_i})$。我们可直接由命题 1 得到以下引理。

引理12。初始集合 $\mathcal{I}_{\Phi'}^{\wedge\vee eq}$ 是正则的。

转移关系和目标集合与上一节中的相同，即 $\mathcal{T}^{v,\ \wedge\vee eq}_{x\rightarrowtail\alpha x} = \mathcal{T}^{v,\ \wedge eq}_{x\rightarrowtail\alpha x}$，$\mathcal{T}^{v,\ \wedge\vee eq}_{x\rightarrowtail\varepsilon} = \mathcal{T}^{v,\ \wedge eq}_{x\rightarrowtail\varepsilon}$，以及 $\mathcal{D}^{\wedge\vee eq} = \mathcal{D}^{\wedge eq}$。以下引理总结了我们的过程对于字方程布尔组合的可靠性。与第 4.2 节一样，可以通过使用重写符号数量不断增加的转换器来实现完备性。

引理13. 给定一个词方程的布尔组合 Φ，使用$(\mathcal{I}_{\Phi'}^{\wedge\vee eq}, \mathbb{V}, \mathcal{T}^{v,\ \wedge\vee eq}_{x\rightarrowtail\alpha x} \cup \mathcal{T}^{v,x\rightarrowtail^{\wedge\vee eq}_{\varepsilon}}, \mathcal{D}^{\wedge\vee eq})$ 实例化的算法*3*是可靠的。

证明. 根据上述步骤，可以将一个完整字方程系统转换为一个与之等可满足的合取范式系统 Ψ: $\bigwedge_{i=1}^{n} c_i$ 其中每个 c_i 都是等式的析取。于是，若存在某个 ϕ: $\bigwedge_{i=1}^{n} \mathrm{t}_\ell^i = \mathrm{t}_r^i$，使得对所有 i均有($\mathrm{t}_\ell^i = \mathrm{t}_r^i$) $\in c_i$，则 Ψ 可满足。此外，我们有 encode(ϕ) $\in \mathcal{I}_\Phi^{\wedge\vee eq}$。根据引理 10（以及 RMC 的 BFS 策略），我们得出，该算法在证明 Φ 可满足时是可靠的。 □

如果像第 4.2 节中那样考虑转换器 $\mathcal{T}^{i,\wedge eq\star}$ for Φ，则可以保证完备性。关于终止性，由于第 4.2 节中针对我们在此考虑的字符串方程布尔组合的一个特殊情况已有相应结果，因此无法保证终止性。

6. 扩展

本节将讨论如何扩展基于 RMC 的框架，以支持以下两种原子约束:

(i) 长度约束 φ_i 是一个 Presburger 算术公式，其变量取值为 $|x|$ for $x \in \mathbb{X}$，其中 $|\cdot|\colon \mathbb{X} \to \mathbb{N}$ 是词长函数（为简化记法，我们使用一个含有自由变量 $\mathbb{X}$ 的 Presburger 算术公式，并注意到赋给 $x \in \mathbb{X}$ 的值实际上对应于 $|x|$）。

(ii) 正则约束 φ_r 是形如 $x \in \mathcal{L}(\mathcal{A})$ 的原子的合取（或其否定），其中 x 是一个字变量，$\mathcal{A}$ 是表示一个正则语言的有限自动机。

6.1. 长度约束

为了将我们的框架扩展到求解带长度约束的字方程，我们将这些约束编码为正则语言，并将尼尔森变换对变量长度的影响编码为正则关系。首先定义原子长度约束:

$$\varphi_{len} ::= a_1 x_1 + \cdots + a_n x_n \leq c$$

对于字符串变量 x_1、.....、$x_n \in \mathbb{X}$和整数 a_1、.....、a_n，$c \in \mathbb{Z}$（我们也会使用限制较少的形式的公式，这些公式始终可以通过标准算术规则转换为所定义的形式）。给定一个变量赋值 $I\colon \mathbb{X} \to \Sigma^*$，当且仅当 I 是 φ_{len}的模型（记作 $I \models \varphi_{len}$），即 $a_1 \cdot |I(x_1)| + \cdots + a_n \cdot |I(x_n)| \leq c$。我们注意到，仅包含通过布尔连接词连接的原子长度约束（即不含字方程）的字符串约束，其可满足性等价于整数线性规划中约束的布尔组合的可满足性；这是一个 NP-完备问题（Karp（1972））。

我们将说明，长度约束可以使用处理 Presburger 算术中约束的标准自动机技术，编码到我们的框架中（Presburger（1929）；Wolper 和 Boigelot（2000，2002））。首先，定义如何在 $\mathrm{MSO}(\mathrm{Str})$ 中表示取值范围为 $\mathbb{N}$ 的一阶变量。令 $LSBF\colon \mathbb{N} \to 2^{\mathbb{N}}$ 为一个函数，用于表示数字的最低有效位优先二进制编码；对于 $n \in \mathbb{N}$，我们定义 $LSBF(n)$ 为

满足 $n = \sum_{i \in S} 2^i$ 的有限集合 $S \subseteq \mathbb{N}$。例如，$LSBF(42) = \{1, 3, 5\}$，因为 $42 = 2^1 + 2^3 + 2^5$。此外，我们将数字 $n \in \mathbb{N}$ 的位置最低有效位优先二进制编码定义为 $LSBF_{\mathcal{P}}(n) = \{\ell + 1 \mid \ell \in LSBF(n)\}$。

命题 14. 设 $\varphi_{len}(x_1, \ldots, x_n)$ 是一个原子长度约束。则存在一个 $\mathrm{MSO}(\mathrm{STR})$ 公式 $\psi_{len}(X_1, \ldots, X_n)$，其自由位置变量为 $X_1, \ldots, X_n$，使得赋值 $\sigma\colon \{x_1, \ldots, x_n\} \to \mathbb{N}$ 是 φ_{len} 的模型，当且仅当赋值 $\sigma' = \{X_i \mapsto LSBF_{\mathcal{P}}(v_i) \mid \sigma(x_i) = v_i\}$ 是 ψ_{len} 的模型。

证明。例如，Glenn和Gasarch（1996）给出了将 φ_{len} 编码为 ψ_{len} 的一种可能方式。 □

回想一下，在 Presburger算术的基于自动机的方法中（例如 Glenn和 Gasarch（1996）；Wolper 和 Boigelot（2000，2002）），具有 k 个自由变量的公式 φ会被转换为一个在字母表 $\mathbb{B}^k$上、针对 $\mathbb{B} = \{0, 1\}$的自动机 $\mathcal{A}_\varphi$。φ 的一个模型表示为 $\mathcal{A}_\varphi$ 语言中的一个词 $w \in \left(\mathbb{B}^k\right)^*$，其中从中投影出变量 x_i 对应的轨道后，得到的就是模型中 x_i取值的$LSBF$编码。例如，若 $x{:}y\colon \left[\begin{smallmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{smallmatrix}\right] \in \mathcal{L}(\mathcal{A}_\varphi)$，则赋值 $\{x \mapsto 9, y \mapsto 5\}$ 是 φ 的一个模型，因为 1001 是数字 9 的 LSBF 二进制编码，而 1010 编码的是数字 5。

为了将处理长度约束的方式编码进我们的框架，仅凭命题 14 并不足够：我们还需要能够表示变换如何修改这些约束，以及约束如何限制可能解空间。接下来的段落将详细介绍我们的方法。

考虑一个编码赋值 $\sigma\colon \mathbb{X} \to \mathbb{N}$ 的词 w_σ。对 w_σ 应用针对 $x, y \in \mathbb{X}$ 的变换$x \rightarrowtail yx$，会得到一个词 $w_{\sigma'}$，它编码赋值$\sigma' = \sigma \lhd \{x \mapsto \sigma(x) - \sigma(y)\}$，前提是 $\sigma(x) \geq \sigma(y)$；其中，$\sigma' = \sigma \lhd \{x \mapsto n\}$ 对所有 $y \neq x$ 定义为 $\sigma'(x) = n$，且 $\sigma'(y) = \sigma(y)$。针对$a \in \Sigma$ 的变换 $x \rightarrowtail ax$ 会得到一个词 $w_{\sigma'}$，它编码赋值 $\sigma' = \sigma \lhd \{x \mapsto \sigma(x) - 1\}$，前提是$\sigma(x) \geq 1$。最后，变换 $x \rightarrowtail \varepsilon$ 不会改变词 w_σ，但会施加限制 $\sigma(x) = 0$。形式上，这些变换由以下公式描述：

$$\begin{aligned} \varphi^{len}_{x \rightarrowtail yx}(x, y, x') &\triangleq x \geq y \wedge x' = x - y, \\ \varphi^{len}_{x \rightarrowtail ax}(x, x') &\triangleq x \geq 1 \wedge x' = x - 1, \text{ and} \\ \varphi^{len}_{x \rightarrowtail \varepsilon}(x) &\triangleq x = 0. \end{aligned} \tag{26}$$

根据命题 14 和命题 2，可知这些公式所表示的关系是正则的。我们将表示这些关系的转换器分别记为 $T^{len}_{x \rightarrowtail yx}$、$T^{len}_{x \rightarrowtail ax}$ 和 $T^{len}_{x \rightarrowtail \varepsilon}$。

现在我们来讨论如何为带有长度约束 $\varphi_i(\mathbb{X})$、且含有自由变量 $\mathbb{X}$ 的方程调整初始集合和目标集合。初始集合通过加入 φ_i 的所有编码模型来扩展。形式化地说，与长度约束相关的初始集合部分表示为 $\mathcal{I}_{\varphi_i} = \mathcal{L}(\varphi_i)$（它是 $\left(\mathbb{B}^{|\mathbb{X}|}\right)^*$ 的子集），而目标集合部分表示为 $\mathcal{D}_{len} = \left(\mathbb{B}^{|\mathbb{X}|}\right)^*$。

二次方程 $eq\colon \mathsf{t}_\ell = \mathsf{t}_r$ 在长度约束 φ_i下的可满足性，随后可表示为 RMC 问题（$\mathcal{I}^{len}_{\varphi_i}$, $\mathcal{T}^{len}_{x \rightarrowtail ax} \cup \mathcal{T}^{len}_{x \rightarrowtail \varepsilon}$, $\mathcal{D}^{len)}_{\varphi_i}$），通过使用以下给定项实例化算法 1（注意这里使用了一个新的

长度约束分隔符 $\#_{len}$）：

$$\mathcal{I}^{len}_{\varphi_i} = \mathcal{I}^{eq}.\{\#_{len}\}.\mathcal{I}_{\varphi_i} \qquad \mathcal{D}^{len}_{\varphi_i} = \mathcal{D}^{eq}.\{\#_{len}\}.\left(\mathbb{B}^{|\mathbb{X}|}\right)^*$$

$$\mathcal{T}^{len}_{x \rightarrowtail \alpha x} = \bigcup_{y\in\mathbb{X}, z\in\mathbb{X}} T^{\leq 2}_{y \rightarrowtail zy}.\{\#_{len} \mapsto \#_{len}\}.T^{len}_{y \rightarrowtail zy} \cup \bigcup_{y\in\mathbb{X}, a\in\Sigma} T^{\leq 2}_{y \rightarrowtail ay}.\{\#_{len} \mapsto \#_{len}\}.T^{len}_{y \rightarrowtail ay}$$

$$\mathcal{T}^{len}_{x \rightarrowtail \varepsilon} = \bigcup_{y\in\mathbb{X}} T^{\leq 2}_{y \rightarrowtail \varepsilon}.\{\#_{len} \mapsto \#_{len}\}.T^{len}_{y \rightarrowtail \varepsilon}$$

将扩展应用于一个方程系统，以及（更一般地）约束的布尔组合，可以采用与第 4 节和第 5 节相同的方式。

$\mathcal{T}^{len}_{x \rightarrowtail \alpha x}$和 $\mathcal{T}^{len}_{x \rightarrowtail \varepsilon}$的有理性可直接由命题 1 得出。我们的算法的可靠性和完备性由引理 15 总结。即使对于二次方程，终止性仍是一个开放问题（参见 B¨uchi 和 Senger（1990））。

引理 15。 给定一个带有长度约束 φ_i的二次词方程 eq：$\mathsf{t}_\ell = \mathsf{t}_r$，将算法 *1* 实例化为（$\mathcal{I}^{len}_{\varphi_i}$、$\mathcal{T}^{len}_{x \rightarrowtail \alpha x} \cup \mathcal{T}^{len}_{x \rightarrowtail \varepsilon}$、$\mathcal{D}^{len}_{\varphi_i}$） 是可靠且完备的。

证明。 我们可以将 Nielsen 证明图中的节点推广为形如 $(\mathsf{t}'_\ell = \mathsf{t}'_r,\ f)$ 的二元组，其中 f是一个将变量长度赋值为 $\mathbb{X}$中元素的映射。变换规则可以直接推广，以同时考虑长度。初始节点是形如 $(\mathsf{t}_\ell = \mathsf{t}_r,\ f)$ 的二元组，其中 f是 φ_i的一个模型。终止节点是形如 $(\varepsilon = \varepsilon,\ g)$ 的节点，其中 g 任意。请注意，即使对于二次方程，推广后的图也未必是有限的。尽管如此，如果该方程可满足，则存在一条从初始节点到终止节点的有限路径。

直接根据 $\mathcal{I}^{len}_{\varphi_i}$的定义可知，广义证明图的初始节点是来自 $\mathcal{I}^{len}_{\varphi_i}$的编码字符串，而终止节点对应于 $\mathcal{D}^{len}_{\varphi_i}$。我们还可以看到，变换规则对应于编码关系 $\mathcal{T}^{len}_{x \rightarrowtail \alpha x}$和 $\mathcal{T}^{len}_{x \rightarrowtail \varepsilon}$。由于算法 1 中的搜索采用广度优先搜索策略，因此我们的（半）算法在证明可满足性时是可靠且完备的。

□

对于一般（非二次）情形以及约束的布尔组合情形，我们可以通过使用转换器 $\mathcal{T}^{i,\wedge eq\star}$，并按照第 4.2 节中的相同方式，同时相应地修改引理 15 的证明，得到一个可靠且完备（但不终止）的过程。

6.2. 正则约束

我们对该框架的第二项扩展，是支持将正则约束表示为形如 $x \in \mathcal{L}(\mathcal{A})$ 的原子的合取，其中 $\mathcal{A}$ 是定义在 Σ 上的有限自动机（注意，原子 $x \notin \mathcal{L}(\mathcal{A})$ 的否定可以转换为正原子 $x \in \mathcal{L}(\mathcal{A})$，其中 $\mathcal{A}$ 是有限自动机 $\mathcal{A}$ 的补）。具体而言，我们假设正则约束表示为以下形式的 ℓ 个原子的合取 φ_r

$$\varphi_r \triangleq \bigwedge_{i=1}^{\ell} (x_i \in \mathcal{L}(\mathcal{A}_i)), \tag{27}$$

其中，对每个 $1 \leq i \leq \ell$， $\mathcal{A}_i$ 都是一个有限自动机。不失一般性，我们假设公式 φ_r 中出现的自动机的状态集两两不交；此外，我们用 $\mathcal{A}_r = (Q_r, \Sigma, \delta_r, I_r, F_r)$ 表示由公式 φ_r 中出现的所有自动机的不交并构造出的自动机。注意，两个有限自动机 $\mathcal{A}_1 = (Q_1, \Sigma, \Delta_1, Q_i^1, Q_f^1)$ 和 $\mathcal{A}_1 = (Q_2, \Sigma, \Delta_2, Q_i^2, Q_f^2)$ 的不交并是有限自动机 $\mathcal{A}_1 \uplus \mathcal{A}_2 = (Q_1 \uplus Q_2, \Sigma, \Delta_1 \uplus \Delta_2, Q_i^1 \uplus Q_i^2, Q_f^1 \uplus Q_f^2)$。

我们在此开发的方法受到 Norn 中方法的启发（参见 Abdulla 等人（ 2015）），但为了适应我们（更注重证明的）框架，必须对这一思路进行重大修改。具体而言，我们将正则约束编码为由形如 $\langle x, p, q \rangle$ where $x \in \mathbb{X}$ and $p, q \in Q_r$ 的符号构成的词。我们将所有此类符号组成的集合记为 $\Gamma_{\mathbb{X},\mathcal{A}_r}$。此外，我们将词视为符号集合，因此假设符号上存在固定的线性序 $\preccurlyeq$ ，以确保表示唯一。具体而言，对于词 $w \in \Gamma^*_{\mathbb{X},\mathcal{A}_r}$ ，我们用 $w_{\preccurlyeq}$ 表示按 $\preccurlyeq$ 排序且不含重复符号的字符串。单个原子 $x \in \mathcal{L}(\mathcal{A}_i)$（其中 $\mathcal{A} = (Q_i, \Sigma, \delta_i, I_i, F_i)$）可以编码为一个词集合 $\mathsf{encode}(x \in \mathcal{L}(\mathcal{A}_i)) = \{\langle x, p, q \rangle \mid p \in I_i, q \in F_i\}$。该集合表示 $\mathcal{A}_i$ 中初始状态和终止状态的所有可能组合。于是，初始集合 $\mathcal{I}_{\varphi_r}$ 定义为

$$\mathcal{I}_{\varphi_r} = \{w_{\preccurlyeq} \in \Gamma^*_{\mathbb{X},\mathcal{A}_r} \mid w \in \mathsf{encode}(x_1 \in \mathcal{L}(\mathcal{A}_1)) \dots \mathsf{encode}(x_\ell \in \mathcal{L}(\mathcal{A}_\ell))\}. \quad (28)$$

注意，对于有限的 $\mathbb{X}$， $\mathcal{I}_{\varphi_r}$ 是有限的，因此它是一个正则语言。

现在我们来描述尼尔森变换对正则约束部分的影响。考虑一个词 w ，它编码了来自 $\Gamma_{\mathbb{X},\ \mathcal{A}_r}$ 的一组符号。然后，将变换 $x \rightarrowtail yx$ 应用于 $x,\ y \in \mathbb{X}$，会生成词 w'，它们编码的集合中，每次出现的符号 $\langle x, p, q \rangle$ 都被替换为所有可能的符号对 $\langle y, p, s \rangle$ 和 $\langle x, s, q \rangle$，其中 $p \rightsquigarrow s$ 和 $s \rightsquigarrow q$ 在 $\mathcal{A}_r$ 中（我们用 $q \rightsquigarrow q'$ 表示在 $\mathcal{A}_r$ 的转移图中存在一条从 q 到 q' 的路径）。类似地，将变换 $x \rightarrowtail ax$ 应用于 $x \in \mathbb{X}$, $a \in \Sigma$，会生成词 w'，它们编码的集合中，每次出现的符号 $\langle x, p, q \rangle$ 都被替换为所有可能的符号 $\langle x, r, q \rangle$，其中 $p \xrightarrow{a} r$ 在 $\mathcal{A}_r$中。最后，通过变换 $x \rightarrowtail \varepsilon$，只有当与变量 x 相关的 w 中所有符号的形式均为 $\langle x, q, q \rangle$（其中 $q \in Q$）时，我们才能得到字符串 $w' = w$。形式化地，我们首先定义针对变量 x 和 y 扩展单个符号的函数如下

$$exp_{x,y}(\sigma) = \begin{cases} \{\, \langle y, p, r \rangle . \langle x, r, q \rangle \mid p \rightsquigarrow r \rightsquigarrow q \text{ in } \mathcal{A}_r \,\} & \text{if } \sigma = \langle x, p, q \rangle, \\ \{\sigma\} & \text{otherwise.} \end{cases} \quad (29)$$

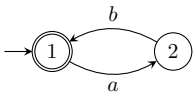
类似地，我们定义扩展

$$exp_{x,a}(\sigma) = \begin{cases} \{\, \langle x, r, q \rangle \mid p \xrightarrow{a} r \rightsquigarrow q \text{ in } \mathcal{A}_r \,\} & \text{if } \sigma = \langle x, p, q \rangle, \\ \{\sigma\} & \text{otherwise.} \end{cases} \quad (30)$$

那么，变换 $x \rightarrowtail yx$、 $x \rightarrowtail ax$ 和 $x \rightarrowtail \varepsilon$ 可以由以下关系描述:

$$\begin{aligned} T^{reg}_{x \rightarrowtail yx} &= \{\, (w, u_{\preccurlyeq}) \mid u \in exp_{x,y}(w[1]) \dots exp_{x,y}(w[|w|]) \,\}, \\ T^{reg}_{x \rightarrowtail ax} &= \{\, (w, u_{\preccurlyeq}) \mid u \in exp_{x,a}(w[1]) \dots exp_{x,a}(w[|w|]) \,\}, \text{ and} \\ T^{reg}_{x \rightarrowtail \varepsilon} &= \left\{\, (w, w) \mid \forall 1 \leq i \leq |w| \colon \forall p, q \in Q \colon w[i] = \langle x, p, q \rangle \Rightarrow p = q \,\right\}. \end{aligned} \quad (31)$$

例7。 考虑如下给出的正则约束 $x \in \mathcal{L}(\mathcal{A}_x)$，其中 $\mathcal{A}_x$ 。



然后，相应的值如下：

$$\begin{aligned}\mathsf{encode}(x \in \mathcal{L}(\mathcal{A}_x)) &= \{\langle x,1,1\rangle\}\\ exp_{x,y}(\langle x,1,1\rangle) &= \{\langle y,1,1\rangle.\langle x,1,1\rangle,\ \langle y,1,2\rangle.\langle x,2,1\rangle\}\\ exp_{x,a}(\langle x,1,1\rangle) &= \{\langle x,2,1\rangle\}\end{aligned}$$

此外，$T^{reg}_{x \rightarrowtail yx}$ 将例如包含形如 $(\langle x,1,1\rangle.\langle y,1,2\rangle,\ u_{\preccurlyeq})$ 的配对，其中

$$u_{\preccurlyeq} \in \{\langle x,1,1\rangle.\langle y,1,1\rangle.\langle y,1,2\rangle,\ \langle x,2,1\rangle.\langle y,1,2\rangle\}$$

（我们假设 $\preccurlyeq$ 是各分量上的字典序）。注意，词中的符号按 $\preccurlyeq$ 排序，并已去重。类似地，$T^{reg}_{x \rightarrowtail ax}$ 将例如包含配对 $(\langle x,1,1\rangle.\langle y,1,2\rangle,\ \langle x,2,1\rangle.\langle y,1,2\rangle)$，而 $T^{reg}_{x \rightarrowtail \varepsilon}$ 将包含 $(\langle x,1,1\rangle.\langle y,1,2\rangle,\ \langle x,1,1\rangle.\langle y,1,2\rangle)$。 □

下面的引理表明这些变换是有理的。在证明中，我们首先构造 MSO(Str) 公式，以实现字符串上的必要集合运算以及扩展函数的作用。基于这些公式，我们借助关系 $\mathsf{pad}_{\square}(T)$，构造实现这些变换的公式；该关系在 w 和 w' 后追加（$w,\ w'$）$\in T$ 任意数量的符号 $\square$（参见第 3.4 节）。

引理 16. 关系 $\mathsf{pad}_{\square}(T^{reg}_{x \rightarrowtail yx})$、$\mathsf{pad}_{\square}(T^{reg}_{x \rightarrowtail ax})$ 和 $\mathsf{pad}_{\square}(T^{reg}_{x \rightarrowtail \varepsilon})$ 都是有理的。

证明。在此证明中，我们将 $\preccurlyeq$ 在 $\Gamma_{\mathbb{X},\ \mathcal{A}_r}$ 上的全序扩展为 $\Gamma_{\mathbb{X},\ \mathcal{A}_r} \cup \{\square\}$ 上的全序，使得 $\forall \sigma \in \Gamma_{\mathbb{X},\ \mathcal{A}_r}\colon\ \sigma \preccurlyeq \square$。（注意，在 MSO(Str) 中对 $\preccurlyeq$ 进行编码很简单。）我们使用 MSO(Str) 定义关系 $T^{reg}_{x \rightarrowtail yx}$ 和 $T^{reg}_{x \rightarrowtail \varepsilon}$。关系 $T^{reg}_{x \rightarrowtail ax}$ 可以类似于 $T^{reg}_{x \rightarrowtail yx}$ 来定义。

$$\begin{aligned}\psi^{reg}_{x \rightarrowtail yx}(w,w') \triangleq \exists^{\mathbb{W}} u_1,u_2,u_3\Big(&\mathit{filter}_x(u_1,u_2,w) \wedge \mathit{expand}^y_x(u_1,u_3) \wedge\\ &\mathit{union}(u_2,u_3,w') \wedge \mathit{ordSet}(w')\Big)\end{aligned} \tag{32}$$

$$\psi^{reg}_{x \rightarrowtail \varepsilon}(w,w') \triangleq \forall^{\mathbb{P}} i\Big(w[i] = w'[i] \wedge \bigvee_{\substack{\xi \in \Gamma_{\mathbb{X}\setminus\{x\},\mathcal{A}_r \cup}\\ \{(x,q,q)\mid q \in Q_r\}}} w'[i] = \xi\Big) \tag{33}$$

其中，筛选$_x(u,\ v,\ w)$将 w 中的符号划分为 u 和 v，使得 u 包含形式为 $\langle x,-,-\rangle$的符号，即对任意 q和 s，形式为 $\langle x,q,s\rangle$ 的符号；而 v 包含其余符号；扩展$^y_x(u,\ v)$将 u 中的每个符号$\langle x,p,q\rangle$ 替换为 v中的 $\langle y,p,s\rangle$ 和 $\langle x,s,q\rangle$；而并是类似集合的并。这些谓词（包括辅助谓词）定义如下：

$$\sigma \in w \triangleq \exists^{\mathbb{P}} i(w[i] = \sigma) \tag{34}$$

$$\mathit{set}(u) \triangleq \neg\exists^{\mathbb{P}} i,j(i \neq j \wedge u[i] = u[j]) \tag{35}$$

$$\mathit{ordSet}(u) \triangleq \mathit{set}(u) \wedge \forall^{\mathbb{P}} i,j(i < j \rightarrow u[i] \preccurlyeq u[j]) \tag{36}$$

$$\mathit{filter}_x(u,v,w) \triangleq \forall^{\mathbb{P}} i \Bigg(\bigwedge_{q,s\in Q_r} (w[i] = \langle x,q,s\rangle \to (u[i] = \langle x,q,s\rangle \wedge v[i] = \square)) \\ \wedge \bigwedge_{\substack{z\in\mathbb{X}\setminus\{x\}\\ q,s\in Q_r}} (w[i] = \langle z,q,s\rangle \to (u[i] = \square \wedge v[i] = \langle z,q,s\rangle)) \Bigg) \tag{37}$$

$$\mathit{expand}_x^y(u,v) \triangleq \bigwedge_{\substack{s,q\in Q_r, s\rightsquigarrow q\\ \xi'=\langle x,s,q\rangle}} \Big(\xi' \in v \to \bigvee_{\substack{p\in Q_r, p\rightsquigarrow s\\ \xi=\langle x,p,q\rangle\\ \xi''=\langle y,p,s\rangle}} \xi \in u \wedge \xi'' \in v\Big) \wedge \\ \bigwedge_{\substack{p,s\in Q_r, p\rightsquigarrow s\\ \xi''=\langle y,p,s\rangle}} \Big(\xi'' \in v \to \bigvee_{\substack{q\in Q_r, s\rightsquigarrow q\\ \xi=\langle x,p,q\rangle\\ \xi'=\langle x,s,q\rangle}} \xi \in u \wedge \xi' \in v\Big) \wedge \\ \bigwedge_{\substack{p,q\in Q_r\\ \xi=\langle x,p,q\rangle}} \Big(\xi \in u \to \bigvee_{\substack{s\in Q_r, p\rightsquigarrow s\rightsquigarrow q\\ \xi''=\langle y,p,s\rangle\\ \xi'=\langle x,s,q\rangle}} \xi'' \in u \wedge \xi' \in v\Big) \tag{38}$$

$$\mathit{union}(u,v,w) \triangleq \bigwedge_{\xi\in\Gamma_{\mathbb{X},\mathcal{A}_r}} \xi \in w \leftrightarrow (\xi \in u \vee \xi \in v) \tag{39}$$

直观而言，在筛选$_x$(*u*, *v*, *w*)的定义中，第一部分从 w 中选取包含 x 的符号，并将它们加入 u ；第二部分则从 w 中选取其他符号，并将它们加入 v。另一方面，在扩展$_x^y$(*u*, *v*)的定义中，我们用 s 表示从状态 p 到状态 q 的路径上的分裂状态。于是，公式的第一部分和第二部分表示 $\langle y,p,s\rangle$和 $\langle x,s,q\rangle$ 属于 v，而最后一部分表示 $\langle x,p,q\rangle$ 属于 u。

我们进一步考虑关系 $\tau_{+\mathsf{pad}} = \{(w,w') \mid w \in (\Gamma_{\mathbb{X},\mathcal{A}_r} \cup \{\square\})^*, w' \in w.\{\square\}^*\}$ 和 $\tau_{-\mathsf{pad}} = \{(w,w') \mid w' \in (\Gamma_{\mathbb{X},\mathcal{A}_r} \cup \{\square\})^*, w \in w'.\{\square\}^*\}$，它们分别用于添加和移除填充。这些关系是有理的。接着，注意到 $\mathsf{pad}_\square(T^{reg}_{x\rightarrowtail yx}) = \tau_{+\mathsf{pad}} \circ \tau_{-\mathsf{pad}} \circ \mathcal{L}(\psi^{reg}_{x\rightarrowtail yx}) \circ \tau_{+\mathsf{pad}}$。回忆一下，关系 $\mathsf{pad}_\square(T)$ 会向 w' in $(w,w') \in T$ 追加任意数量的符号 $\square$（参见第 3.4 节）。根据命题 1 和命题 2，我们可知 $\mathsf{pad}_\square(T^{reg}_{x\rightarrowtail yx})$ 是有理的（$\mathsf{pad}_\square(T^{reg}_{x\rightarrowtail \varepsilon})$ 也同样如此）。 □

最后缺少的部分是定义一个包含所有可满足正则约束的目标集合。对于变量 $x \in \mathbb{X}$，我们将可满足的 x约束集定义为 $L^x = \{ w_{\preccurlyeq} \mid w = \langle x,q_1,r_1\rangle \cdots \langle x,q_n,r_n\rangle \in \Gamma^*_{\mathbb{X},\mathcal{A}_r}, \bigcap_{i=1}^n \mathcal{L}_{\mathcal{A}_r}\ q_i\ r_i\ \neq (,) \}$。然后，变量集的目标集合表示为$\mathcal{D}_{reg} = \{ w_{\preccurlyeq} \mid w \in L^{x_1} \cdots L^{x_k} \}$。与 $\mathcal{I}_{\varphi_r}$ 的情况一样，集合 $\mathcal{D}_{reg}$ 是有限的，因此也是正则的。 $\mathbb{X} = \{x_1,\ldots,x_k\}$

带正则约束 φ_r 的二次词方程 eq: $\mathsf{t}_\ell = \mathsf{t}_r$ 的可满足性可以在 RMC 框架中表示为 ($\mathcal{I}^{reg}_{\varphi_r}, \mathcal{T}^{reg}_{x\rightarrowtail \alpha x} \cup \mathcal{T}^{reg}_{x\rightarrowtail \varepsilon}, \mathcal{D}^{reg}_{\varphi_r}$)，将算法 1 实例化为以下各项（注意，我们使用一个新的

分隔符 $\#_{reg}$）：

$$\mathcal{I}^{reg}_{\varphi_r} = \mathcal{I}^{eq}.\{\#_{reg}\}.\mathsf{pad}_{\square}(\mathcal{I}_{\varphi_r}) \qquad \mathcal{D}^{reg}_{\varphi_r} = \mathcal{D}^{eq}.\{\#_{reg}\}.\mathsf{pad}_{\square}(\mathcal{D}_{reg})$$

$$\mathcal{T}^{reg}_{x \rightarrowtail \alpha x} = \bigcup_{y\in\mathbb{X}, z\in\mathbb{X}} T^{\leq 2}_{y \rightarrowtail zy}.\{\#_{reg} \mapsto \#_{reg}\}.\mathsf{pad}_{\square}(T^{reg}_{y \rightarrowtail zy}) \cup \bigcup_{y\in\mathbb{X}, a\in\Sigma} T^{\leq 2}_{y \rightarrowtail ay}.\{\#_{reg} \mapsto \#_{reg}\}.\mathsf{pad}_{\square}(T^{reg}_{y \rightarrowtail ay})$$

$$\mathcal{T}^{reg}_{x \rightarrowtail \varepsilon} = \bigcup_{y\in\mathbb{X}} T^{\leq 2}_{y \rightarrowtail \varepsilon}.\{\#_{reg} \mapsto \#_{reg}\}.\mathsf{pad}_{\square}(T^{reg}_{y \rightarrowtail \varepsilon})$$

$\mathcal{T}^{reg}_{x \rightarrowtail \alpha x}$ 和 $\mathcal{T}^{reg}_{x \rightarrowtail \varepsilon}$ 的有理性可直接由命题 1 得出。我们的过程的可靠性和完备性由以下引理概括。

引理 17. 给定一个二次词方程 eq：$\mathsf{t}_\ell = \mathsf{t}_r$ 及一个正则约束 φ_r，使用 $(\mathcal{I}^{reg}_{\varphi_r}, \mathcal{T}^{reg}_{x \rightarrowtail \alpha x} \cup \mathcal{T}^{reg}_{x \rightarrowtail \varepsilon}, \mathcal{D}^{reg}_{\varphi_r})$ 实例化的算法 *1* 是可靠的、完备的且终止的。

证明。 与引理 15 的证明类似，我们可以将 Nielsen 证明图中的节点推广为形如 $(\mathsf{t}'_\ell = \mathsf{t}'_r, S)$ 的有序对，其中 $S \subseteq \Gamma_{\mathbb{X},\mathcal{A}_r}$。变换规则可以直接推广，以同时考虑由 $\Gamma_{\mathbb{X},\mathcal{A}_r}$ 的子集表示的正则约束。由于 $\Gamma_{\mathbb{X},\mathcal{A}_r}$ 是有限的，且 eq 是二次的，广义证明图是有限的。广义证明图的初始节点恰好是来自 $\mathcal{I}^{reg}_{\varphi_r}$ 的编码字符串，终止节点对应于 $\mathcal{D}^{reg}_{\varphi_r}$，而变换规则对应于编码关系 $\mathcal{T}^{reg}_{x \rightarrowtail \alpha x}$ 和 $\mathcal{T}^{reg}_{x \rightarrowtail \varepsilon}$。由于我们的 RMC 框架采用 BFS 策略，由此可知，我们的过程在证明可满足性时是可靠的、完备的且终止的。

□

与长度约束的情形一样，带正则约束的字方程 eq 的可满足性可以推广到带正则约束的方程组 Φ。与 eq 对应的语言/关系将替换为与 Φ 对应的语言/关系。对于带正则约束的字方程组，我们的算法仍然是可靠的（若考虑转换器 $\mathcal{T}^{i, \wedge eq\star}$ for Φ，则也是完备的）。

讨论。 在上一节中，我们详细阐述了针对包括长度和正规扩展在内的各种字符串约束所提出的 RMC 框架。表 1 总结了所取得的结果。如果方程组是二次的（即使进一步加入正规扩展），那么我们的 RMC 方法是可靠、完备且终止的。这基本上是由于以下事实：算法 1 中针对二次方程定制的变换所处理的语言，在有限步之后会达到饱和。在所有其他情况下，我们的 RMC 方法是可靠且完备的（但通常不会终止）。对于适当编码的变换，如果系统是可满足的，我们能够在有限步之后得到一个解。但一般而言，算法 3 中这些变换所处理的语言并不能保证最终达到饱和。

表 1：所提出方法在带扩展的各种类型字符串约束上的总结。 S 表示 *sound*， C 表示 *complete*， T 表示 *terminating*。

	二次系统	一般系统	布尔组合
pure	SCT（第 4.1 节）	SC（第 4.2 节）	SC（第 5 节）
长度	SC（第 6.1 节）	SC（第 6.1 节）	SC（第 6.1 节）
正则	SCT（第 6.2 节）	SC（第 6.2 节）	SC（第 6.2 节）

7. 实现

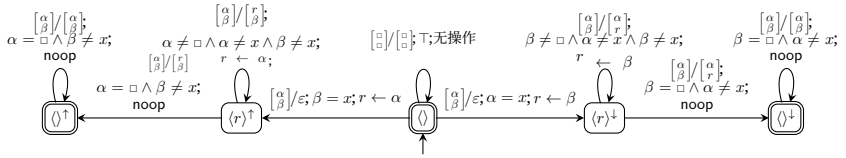
我们创建了一个名为 RETRO[1] 的 Python 工具原型，并在其中实现了用于求解字方程组的符号过程。 RETRO实现了对算法 1 中 RMC循环的修改版本。具体而言，它使用所谓的有限字母表寄存器变换器 （FRT），而不是第 2 节定义的标准转换器；这种变换器能够更简洁地表示一个有理关系。

非正式地说，FRT 是一种寄存器自动机（依照 Kaminski 和 Francez（1994）；Demri 和 Lazic（2009）的定义），其字母表是有限的。字母表的有限性意味着 FRT 的表达能力与有理语言类相同，但使用 FRT 的优势在于，它们比普通转换器能更简洁地表示内容。可以将 FRT 视为 Veanes 等人（2012）的带寄存器符号变换器在有限字母表上的限制。处理这些转换器的操作因此只是 Veanes 等人（2012）所述操作的直接限制，故此处不再赘述。

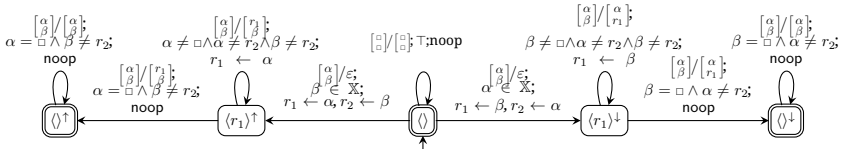
特别是，与变换器 $\mathcal{T}_{x \rightarrowtail \alpha x}$ 和 $\mathcal{T}_{x \rightarrowtail \varepsilon}$ 对应的（不带寄存器的）转换器，会在开头针对 x 和 α 的每种选择产生分支。尤其是在字母表很大的情况下，这会导致转换器规模庞大（例如，考虑包含超过 100 万个符号的 Unicode字母表）。使用 FRT 可以得到小得多的自动机，因为 x 和 α 的选择会存入寄存器，然后以符号方式处理。为了说明使用寄存器的效果，请看图8中的转换器，它实现了针对 $\mathbb{X} = \{x\}$ 和 $\Sigma = \{a\}$ 的编码关系 $T^{\leq 1}_{x \mapsto \varepsilon}$。对于大型字母表，完整的转换器需要针对每个 $\left[\begin{smallmatrix} u \\ v \end{smallmatrix}\right]$ 进行分支，其中 u， $v \in \Sigma_{\mathbb{X}}$，并且需要大量状态来存储具体的移位符号。具体而言，每个 $v \in \Sigma_{\mathbb{X}}$ 至少需要一对状态 $\langle v \rangle^{\uparrow}$ 和 $\langle v \rangle^{\downarrow}$，这对于非常大的 $\Sigma_{\mathbb{X}}$ 来说是不可行的。另一方面，图11a中的 FRT 将移位符号存储在寄存器 r 中，以符号转移取代分支，因此只需少量状态和转移。此外，通过使用额外的寄存器 r_2 存储待替换的变量，我们可以高效地表示关系 $\bigcup_{x \in \mathbb{X}} T^{\leq 1}_{x \mapsto \varepsilon}$，如图11b所示。

具体来说， RETRO 实现了一般字方程和长度约束系统的判定过程（即第 3、4 节及第 6.1 节所述过程）。它不实现（i）约束的布尔组合（第 5 节）和（ii）正则约束（第 6.2 节），因为这两类约束的处理效率很低；加入它们只是为了让我们的方法在处理大量输入约束时拥有更扎实的理论基础——

[1]可在 https://github.com/VeriFIT/retro获取



(a) 任意变量集 $\mathbb{X}$ 和字母表 Σ 上关系 $T^{\leq 1}_{x\to\varepsilon}$ 的 FRT



(b) 对于任意变量集 $\mathbb{X}$ 和字母表 Σ 上的关系 $\bigcup_{x\in\mathbb{X}} T^{\leq 1}_{x\to\varepsilon}$，一个 FRT

图 11：FRT (a) 实现了关系 $T^{\leq 1}_{x\mapsto\varepsilon}$，适用于任意变量集 $\mathbb{X}$ 和字母表 Σ。寄存器 r 用于存储移位后的符号。FRT (b) 实现了关系 $\bigcup_{x\in\mathbb{X}} T^{\leq 1}_{x\mapsto\varepsilon}$，适用于任意变量集 $\mathbb{X}$ 和字母表 Σ。寄存器 r_1 用于存储移位后的符号，寄存器 r_2 用于存储应被移除的变量。在图中，α 和 β 是表示输入符号的变量，r，r_1 和 r_2 是寄存器，转移的形式为 $action;condition;register\ update$。

（参见第 5 节开头的讨论）。

作为另一项特性，RETRO 使用确定性有限自动机来表示算法 1 中的配置。它还使用急切自动机最小化，因为这对性能有很大影响，尤其是在检查 RMC 算法的终止条件时；该检查通过测试当前配置与截至目前所有已处理配置之间的语言包含关系来完成。

8. 实验评估

我们将本方法（在 RETRO中实现）的性能与四种支持字符串理论的当前最先进 SMT 求解器进行了比较：Z3 4.8.14（de Moura 和 Bjørner（2008））、Z3STR3RE（Berzish 等人（2021））、CVC4 1.8（Barrett 等人（2011））和 CVC5 1.0.1（Barbosa 等人（2022））。至于我们所知的其他求解器，Abdulla 等人（2015）的 NORN 和 Chen 等人（2019b）的 OSTRICH 的性能远不如所考察的工具；Z3STR4（Mora 等人（2021））的性能与 Z3STR3RE 相近；Hol´ık 等人（2018）的 SLOTH 在所考察的片段上不可靠（它仅支持所谓的直线片段）。

第一组基准测试是 Kepler_{22}，，取自 Le 和 He（2018年）。Kepler_{22} 包含 600 个人工构造的字符串约束，由带长度约束的二次词方程组成。图 12 展示了各求解器在 Kepler_{22} 基准测试集上的结果仙人掌图，超时时间为 20 秒。在仙人掌图中，求解器的曲线越靠近右侧和底部边界，其表现就越好。在超时时间内求解的基准测试总数分别为：Z3 为 243 个，Z3STR3RE为 282 个，CVC4 为 259 个，CVC5 为 352 个，RETRO为 443 个。我们可以通过比较每对工具各自求解的案例来细化这些数字，如表 2 所示。从表中可以看出，RETRO 求解的基准测试显著多于所有

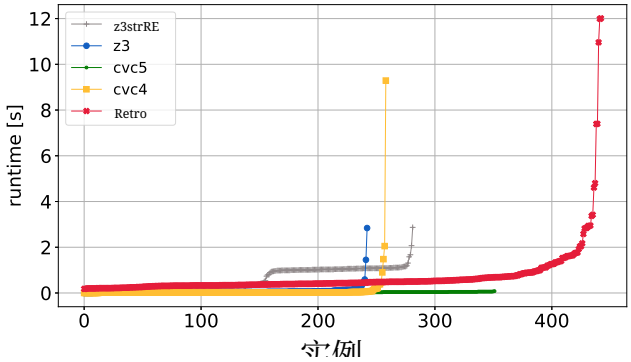


图 12：仙人掌　　比较 Retro、CVC4、CVC5、Z3 和 Z3Str的仙人掌图 3RE关于 $\texttt{Kepler}_{22}$

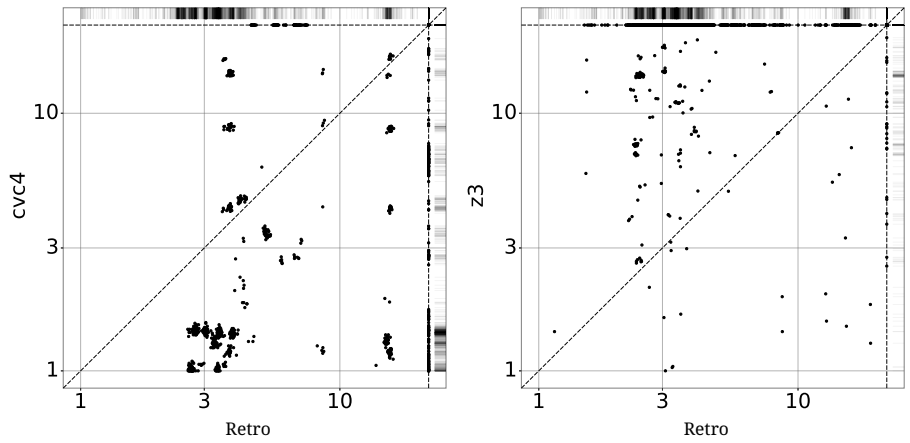
其他最先进的工具，与 CVC5 相比多 92 个，与 Z3 相比多 201 个。只有一个案例中， Retro求解失败（ Z3Str3RE也同样失败），而 CVC4、CVC5 和 Z3 均成功。除 CVC4 与 CVC5 的比较外，其他工具之间的比较结果均不明确，因为表中的 $(X,\ Y)$ 和 $(Y,\ X)$ 两项数值都很大。

我们尝试的另一组基准测试是 PyEx-Hard。在这里，我们想考察将 Retro 集成到基于 DPLL(T) 的字符串求解器（如 Z3 或 CVC4 ）中，作为专用字符串理论求解器的潜力。该组件的输入是原子字符串公式的合取（例如 $xy = zb \wedge z = ax$），它是顶层公式布尔结构的一个模型。随后，多个字符串理论求解器会分层处理这些原子字符串公式的合取，并添加更多冲突子句或返回一个模型。为了评估 Retro 是否适合用作 Z3 或 CVC4 字符串求解器的 “ 其中一层 ”，我们分析了 Reynolds 等人（2017）的 PyEx 基准测试，并从中筛选出 967 个困难实例，这些实例中 CVC4 和 Z3 均无法在 10 秒内求解。对于这些实例，我们获得了 20,020 个由 Z3 的 DPLL(T) 算法在尝试求解时发送给其字符串理论求解器的字方程合取。我们将这 20,020 个字方程合取称为 PyEx-Hard。接着，我们在 PyEx-Hard 上以 20 秒为超时时间评估了其他求解器。其中，Z3 无法求解 3,001 个实例，Z3Str3RE 814，CVC4 无法求解 152 个，CVC5 无法求解 171 个，而 Retro 无法求解 3,079 个实例。

现在让我们仔细看看 PyEx-Hard 基准测试集中较难的实例，尤其是其他工具无法求解的实例。

表 2： $\texttt{Kepler}_{22}$ 基准测试上已解决案例的细分。第 X 行第 Y 列中的数字表示求解器 X 能够解决而求解器 Y 无法解决的案例数量。

	Retro	CVC4	CVC5	Z3	Z3Str3RE
Retro	—	185	92	201	161
CVC4	1	—	2	104	63
CVC5	1	95	—	197	145
Z3	1	88	88	—	23
Z3Str3RE	0	86	75	62	—



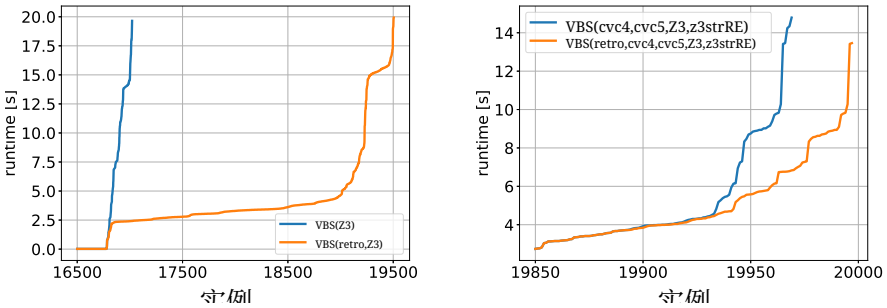
（a） Retro与 CVC4 对比　　（b） Retro与 Z3 对比

图 13： Retro 与 CVC4 和 Z3 在 PyEx-Hard上的比较。我们展示了耗时超过 1 秒才完成的困难实例。时间以秒为单位，坐标轴采用对数刻度。

这些基准测试无法通过这些工具中实现的（多层）快速启发式方法来处理；虽然这些方法足以求解许多基准测试，但无需开始应用分情况规则。[2] 图 13 对比了 Retro 与 CVC4 和 Z3 的运行时间，特别关注较难的实例（运行时间超过 1 秒）。CVC5 和 Z3Str3RE 的图表呈现出类似的趋势。从图中可以看出，在许多较难的实例上， Retro 给出答案的速度远快于其他工具，尤其是相较于 Z3。比较各求解器在其他工具无法求解的示例上的表现，可以发现，在 CVC4 求解失败的示例中， Retro求解了 86 个（56.2%）；在 CVC5 求解失败的示例中，求解了 130 个（76.02%）；在 Z3 求解失败的示例中，求解了 2,484 个（82.7%）；在 Z3Str3RE 求解失败的示例中，求解了 519 个（63.75%）。此外， Retro 还有 28 个实例是唯一能够求解的工具。最后，我们考察 Retro 对虚拟最佳求解器的影响：给定一组求解器 S，我们用 $VBS(S)$表示这样一种求解器：对于给定基准测试中的每一项，都选取在该项上运行速度最快的求解器。在图 14 中，我们给出了仙人掌图，展示了 Retro 对两种虚拟最佳求解器实例的影响； Z3Str3RE、CVC4 和 CVC5 的图表也呈现出类似趋势，其中 Z3Str3RE 的图表更接近图 14a，而 CVC4/5 的图表则更接近图 14b。由此可见，这些图表表明，我们的方法可以显著帮助求解器处理困难的方程。

讨论。 从得到的结果可以看出，我们的方法在困难情形下表现良好；在这些情况下，最先进求解器中实现的快速启发式方法不足以快速判定公式，而这尤其会发生在（不可）满足性证明复杂时。我们的方法能够利用证明树的符号表示，并借此减少执行变换时的冗余。

[2]例如，当 Z3 接收到字方程 $xy = yax$时，它会推断出长度约束$|x| + |y| = |y| + 1 + |x|$，这意味着该字方程不可满足，无需开始应用分情况规则。



(a) *VBS*(RETRO, Z3)　　(b) *VBS*(RETRO, CVC4, CVC5, Z3, Z3STR3RE)

图 14：仙人掌图比较了（a）Z3 在有和没有 RETRO 时的虚拟最佳求解器，以及（b）所有工具在有和没有 RETRO 时针对 PYEX-HARD 基准测试的表现。我们仅展示其中最难的基准测试（共 20,020 个）。

在这些情形下，我们仍然可以用 Python 原型击败经过高度优化、以 C++ 编写的 Z3 Z3STR3RE、CVC4 和 CVC5。我们相信，将我们的符号算法作为最先进 SMT 求解器的一部分加以实现，将进一步拓展字符串求解的适用范围，尤其是针对结构复杂的字符串约束；这类约束需要求解多个 DPLL(T) 查询，才能确定字符串公式是否可满足。

9. 相关工作

关于字符串约束求解的研究可追溯至 1946 年，当时 Quine（1946）证明了字方程一阶理论的不可判定性。Makanin（1977）取得了里程碑式的成果，证明无量词字方程类是可判定的。此后，多项研究，例如 Plandowski（1999，2006）；Matiyasevich（2008）；Robson 和 Diekert（1999）；Schulz（1990）；Ganesh 等人（2012）；Ganesh 和 Berzish（2016）；Abdulla 等人（2014）；Barcel´o 等人（2013年）；Lin 和 Barcel´o（2016）；Chen 等人（2018，2019b）；Abdulla 等人（2019），探讨了不同类别字符串约束的可判定性和复杂度。高效求解字符串约束的可满足性是一项具有挑战性的问题。此外，字方程与形如 $|x| = |y|$ 的长度约束相结合时，其可满足性问题的可判定性已悬而未决 20 余年，见 B¨uchi 和 Senger（1990）。

强烈的实际需求推动了多种字符串约束求解器的兴起，这些求解器专注于解决实际问题实例。基于 *DPLL(T)* 的 字符串求解器通常采用这样的过程：根据解的对齐方式，将约束拆分为更简单的子情况，并结合强大的布尔推理技术，高效地探索由此产生的指数规模搜索空间。分情况规则通常以显式方式执行。采用这种方法的求解器包括 NORN(Abdulla 等人（2014年、2015年）)、 TRAU (Abdulla 等人（2018年）)、 OSTRICH (Chen 等人（2019b）)、 SLOTH (Hol´ık 等人（2018年）)、CVC4（Barrett 等人（2011））、CVC5（Barbosa 等人（2022））、 Z3STR2 (Zheng 等人（2017）)、 Z3STR3 (Berzish

等人（2017）)、 Z3str4 (Mora 等人（2021）)、 Z3Str3RE (Berzish 等人（2021）)、S3（Trinh 等人（2014年））、S3P（Trinh 等人（2016））。相比之下，我们的方法以符号方式执行分情况拆分。

自动机和转换器已用于许多字符串求解方法和工具中，例如 Norn (Abdulla 等人（2014年，2015年）)、 Trau(Abdulla 等人（2018年）)、 Ostrich (Chen 等人（2019b）)、 Sloth(Hol´ík 等人（2018年）)、 Slog(Wang 等人（2016）)、 Slent (Wang 等人（2018年）)或 Z3Str3RE(Berzish 等人（2021）)；它们也用于分析字符串处理程序的字符串求解器中，例如 ABC（Aydin 等人（2018年））和 Stranger（Yu 等人（2010）），后者使用转换器对字符串约束进行可靠的过近似（Yu 等人（2016））。这些方法与我们的方法之间的主要区别在于，它们使用转换器编码字符串约束的可能模型（解），而我们使用自动机和转换器编码字符串约束变换。

解决字符串约束的其他方法包括将约束归约为位向量的SMT理论（例如 Zheng 等人（2017）；Berzish 等人（2017）；Mora 等人（2021）；Kiezun 等人（2012））、数组理论（例如 Li 和 Ghosh（2013））或 SAT求解（例如 Day 等人（2019）；Amadini 等人（2017）；Scott 等人（2017））。基于代数方法的方案并不多见，例如尼尔森变换。除我们的方法外，Le 和 He（2018）的工作也以此为基础。另一方面，尼尔森变换（Nielsen（1917））被一些实现不同方法、用于求解二次方程的工具所采用（例如 Chen 等人（2019b）的 Ostrich ）。在处理 CVC5 中的正则约束时，也会使用复杂的重写规则（例如 N¨otzli 等人（2022））。

致谢

我们感谢 Mohamed Faouzi Atig 与我们讨论这一主题。本研究部分得到了国家自然科学基金（项目编号：61836005）、中国科学院基础研究青年科学家项目（项目编号：YSBR-040）、捷克科学基金会项目 19-24397S、布尔诺理工大学信息技术学院内部项目 FIT-S-20-6427，以及台湾科学技术部项目（项目编号：MOST-109-2628-E-001-001-MY3）的资助。

本研究是欧盟 Horizon 2020 研究与创新计划的一部分，获得了 Marie Sklodowska-Curie 项目资助（项目编号：101008233）。

参考文献

Abdulla, P.A., 2012. 正则模型检测。STTT 14, 109–118. doi:10.1007/ s10009-011-0216-8.

Abdulla, P.A., Atig, M.F., Chen, Y., Diep, B.P., Hol´ık, L., Rezine, A., R¨ummer, P., 2017. 展开并征服：一种用于高效分析字符串约束的框架，载于：第 38 届 ACM SIGPLAN 编程语言设计与实现会议论文集（PLDI 2017），ACM。第 602–617 页。doi:10.1145/3062341.3062384.

Abdulla, P.A., Atig, M.F., Y.F. Chen, Diep, B.P., Hol´ık, L., Rezine, A., R¨ummer, P., 2018年。Trau：用于字符串约束的SMT求解器，载于：2018年计算机辅助设计中的形式化方法（FMCAD），IEEE。第1–5页。

Abdulla, P.A., Atig, M.F., Y.F. Chen, Hol´ık, L., Rezine, A., R¨ummer, P., Stenman, J., 2014年。用于验证的字符串约束，载于：国际计算机辅助验证会议，施普林格。第150–166页。

Abdulla, P.A., Atig, M.F., Y.F. Chen, Hol´ık, L., Rezine, A., R¨ummer, P., Stenman, J., 2015年。Norn：一种用于字符串约束的SMT求解器，载于：国际计算机辅助验证会议，施普林格。第462–469页。

Abdulla, P.A., Atig, M.F., Diep, B.P., Hol´ık, L., Jank˚u, P., 2019。无链字符串约束，载于：ATVA，施普林格。第277–293页。

Amadini, R., Gange, G., P.J. Stuckey, Tack, G., 2017。一种新颖的字符串约束求解方法，载于：Beck, J.C.（编），《约束编程的原理与实践——第23届国际会议，CP 2017》，澳大利亚维多利亚州墨尔本，2017年8月28日至9月1日，论文集，施普林格。第3–20页。网址：https://doi.org/10.1007/978-3-319-66158-2_1, doi:10.1007/978-3-319-66158-2_1。

Aydin, A.、Eiers, W.、Bang, L.、Brennan, T.、Gavrilov, M.、Bultan, T.、F. Yu，2018年。《字符串与数值约束的参数化模型计数》，载于：2018年第26届ACM欧洲软件工程大会与软件工程基础研讨会联合会议论文集，ACM，第400–410页。

Barbosa, H., Barrett, C., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mo- hamed, A., Mohamed, M., Niemetz, A., N¨otzli, A., Ozdemir, A., Preiner, M., Reynolds, A., Sheng, Y., Tinelli, C., Zohar, Y., 2022. CVC5：一种多功能、具备工业级能力的SMT求解器，载于：Fisman, D., Rosu, G.（编），系统构造与分析工具和算法，施普林格国际出版，Cham。第415–442页。Barcel´o, P., Figueira, D., Libkin, L., 2013年。具有有理关系的图逻辑。arXiv预印本 arXiv:1304.4150。Barrett, C.W., Conway, C.L., Deters, M., Hadarean, L., Jovanovic, D., King, T., Reynolds, A., Tinelli, C., 2011。CVC4，载于：Gopalakrishnan, G., Qadeer, S.（编），计算机辅助验证——第23届国际会议，CAV 2011，Snowbird，犹他州，美国，2011年7月14日至20日。论文集，施普林格。第171–177页。网址：https://doi.org/10.1007/978-3-642-22110-1_14, doi:10.1007/978-3-642-22110-1_14。Berstel, J., 1979。转换与上下文无关语言。*Teub-ner Studienb¨ucher : Informatik*第38卷。Teubner。网址：http://www.worldcat.org/ oclc/06364613。Berzish, M., Ganesh, V., Zheng, Y., 2017。Z3str3：一种采用理论感知启发式方法的字符串求解器，载于：2017年计算机辅助设计中的形式化方法，FMCAD 2017，第55–59页。doi:10.23919/FMCAD.2017.8102241。Berzish, M., Kulczynski, M., Mora, F., Manea, F., Day, J.D., Nowotka, D., Ganesh, V., 2021。一种用于正则表达式以及字符串长度上线性算术的SMT求解器，载于：国际计算机辅助验证会议，施普林格。第289–312页。

Bjørner, N., Tillmann, N., Voronkov, A., 2009。字符串处理程序的路径可行性分析，载于：系统构造与分析工具和算法，第15届国际会议，TACAS 2009，施普林格。第307–321页。doi:10.1007/978-3-642-00768-2_27。

Blotsky, D., Mora, F., M. Berzish, Y. Zheng, Kabir, I., V. Ganesh, 2018年。Stringfuzz：一种用于字符串求解器的模糊测试器，载于：国际计算机辅助验证会议，施普林格出版社，第45–51页。

Bouajjani, A., Habermehl, P., Rogalewicz, A., Vojnar, T., 2012。抽象正则（树）模型检测。STTT 14，167–191。URL：https://doi.org/10.1007/s10009-011-0205-y，doi:10.1007/s10009-011-0205-y。Bouajjani, A., Jonsson, B., Nilsson, M., Touili, T., 2000。正则模型检测，载于：E merson, E.A., Sistla, A.P.（编），计算机辅助验证，Springer Berlin Heidelberg，柏林，海德堡。第403–418页。B¨uchi, J.R., 1960。弱二阶算术与有限自动机。《数学逻辑季刊》6，66–92。doi:https://doi.org/10.1002/malq. 19600060105。B¨uchi, J.R., Senger, S., 1990。连接的存在理论及该理论的不可判定扩展中的可定义性，载于：《J. Richard B¨uchi文集》。施普林格，第671–683页。Cadar, C., Ganesh, V., Pawlowski, P.M., Dill, D.L., Engler, D.R., 2006。Exe：自动生成致命输入，载于：第13届ACM计算机与通信安全会议论文集，计算机协会。第322–335页。doi:10.1145/1180405.1180445。Chen, T., Chen, Y., Hague, M., Lin, A.W., Wu, Z., 2018。关于带有ReplaceAll函数的字符串约束，哪些内容是可判定的。PACMPL 2，3:1–3:29。doi:10.1145/3158091。Chen, T., Hague, M., Lin, A.W., R¨ummer, P., Wu, Z., 2019a。带复杂操作的字符串处理程序路径可行性的判定过程。PACMPL 3，49:1–49:30。doi:10.1145/3290362。Chen, T., Hague, M., Lin, A.W., R¨ummer, P., Wu, Z., 2019b。带复杂操作的字符串处理程序路径可行性的判定过程。《ACM编程语言会议论文集》3，1–30。Chen, Y., Havlena, V., Leng´al, O., Turrini, A., 2020。字符串约束求解中分情况规则的符号算法，载于：d. S. Oliveira, B.C.（编），《编程语言与系统——第18届亚洲研讨会》，APLAS 2020，日本福冈，2020年11月30日至12月2日，论文集，施普林格。第343–363页。URL：https://doi.org/10.1007/978-3-030-64437-6_18，doi:10.1007/978-3-030-64437-6_18。Day, J.D., Ehlers, T., Kulczynski, M., Manea, F., Nowotka, D., Poulsen, D.B., 2019。使用布尔可满足性问题求解字方程，载于：Filiot, E., Jungers, R.M., Potapov, I.（编），《可达性问题——第13届国际会议》，RP 2019，比利时布鲁塞尔，2019年9月11日至13日，论文集，施普林格。第93–106页。URL：https://doi.org/10.1007/978-3-030-30806-3_8，doi:10.1007/978-3-030-30806-3_8。

Demri, S., Lazic, R., 2009。带冻结量词和寄存器自动机的线性时序逻辑。ACM计算逻辑汇刊10，16:1–16:30。URL：https://doi. org/10.1145/1507244.1507246, doi:10.1145/1507244.1507246。

Diekert, V.，2002。Makanin 算法。剑桥大学出版社。《数学及其应用百科全书》，第 387–442 页。doi:10.1017/CBO9781107326019.013。

Durnev, V.G.，Zetkina, O.V.，2009。带有特定解约束的自由半群中的单方程。《数学科学杂志》158，671–676。doi:10.1007/s10958-009-9409-z。

Ganesh, V.，Berzish, M.，2016。字符串理论、关于长度的线性算术以及字符串与数值转换的不可判定性。arXiv 预印本 arXiv:1605.09442。

Ganesh, V.，Minnes, M.，Solar-Lezama, A.，Rinard, M.，2012。带长度约束的字方程：哪些是可判定的？，载于：海法验证会议，施普林格。第 209–226 页。

Glenn, J., Gasarch, W.I., 1996. 通过有限自动机实现弱单一后继理论，载于：Raymond, D.R., Wood, D., Yu, S.（编），《自动机实现》，第一届自动机实现国际研讨会，WIA ’96，加拿大安大略省伦敦，1996年8月29日至31日，修订论文集，施普林格。第50–63页。URL: https://doi.org/10.1007/3-540-63174-7_5, doi:10.1007/ 3-540-63174-7_5. Godefroid, P., Klarlund, N., Sen, K., 2005. DART：定向自动随机测试。SIGPLAN通报 40, 213–223. doi:10.1145/1064978.1065036. Gulwani, S., Jha, S., Tiwari, A., Venkatesan, R., 2011. 无循环程序的综合。SIGPLAN通报 46, 62–73. doi:10.1145/1993316.1993506. Gulwani, S., Srivastava, S., Venkatesan, R., 2008. 将程序分析视为约束求解，载于：PLDI’08。Hol´ík, L., Jank˚u, P., Lin, A.W., R¨ummer, P., Vojnar, T., 2018. 高效求解包含连接和转换器的字符串约束。Proc. ACM Program. Lang. 2, 4:1–4:32. URL: https://doi.org/10.1145/3158092, doi:10.1145/3158092. Kaminski, M., Francez, N., 1994. 有限存储自动机。Theor. Comput. Sci. 134, 329–363. URL: https://doi.org/10.1016/0304-3975(94)90242-9, doi:10.1016/0304-3975(94)90242-9. Karp, R.M., 1972. 组合问题之间的可约性，载于：Miller, R.E., Thatcher, J.W.（编），《计算复杂性专题讨论会论文集》，于1972年3月20日至22日在美国纽约州约克敦高地的IBM托马斯·J·沃森研究中心举行，普莱纳姆出版社，纽约。第85–103页。URL: https://doi.org/10.1007/ 978-1-4684-2001-2_9, doi:10.1007/978-1-4684-2001-2_9。

Kesten, Y., Maler, O., Marcus, M., Pnueli, A., Shahar, E., 2001. 使用丰富的断言语言进行符号模型检验。Theor. Comput. Sci. 256, 93–112. URL: https://doi.org/10.1016/S0304-3975(00)00103-1, doi:10.1016/S0304-3975(00)00103-1. Kiezun, A., Ganesh, V., Artzi, S., Guo, P.J., Hooimeijer, P., Ernst, M.D., 2012. HAMPI：用于字符串上的字方程、正则表达式和上下文无关文法的求解器。ACM Trans. Softw. Eng. Methodol. 21, 25:1–25:28. doi:10.1145/2377656.2377662. King, J.C., 1976. 符号执行与程序测试。Commun. ACM 19, 385–394. doi:10.1145/360248.360252. Knoth, T., Wang, D., Polikarpova, N., Hoffmann, J., 2019. 资源引导的程序综合，载于：《第40届ACM SIGPLAN编程语言设计与实现会议论文集》，计算机协会，纽约，美国纽约州。第253–268页。URL: https: //doi.org/10.1145/3314221.3314602, doi:10.1145/3314221.3314602. Kosovskii, N.K., 1976. 自由半群中方程解的性质。J. Math. Sci. 6. doi:10.1007/BF01084074. Le, Q.L., He, M., 2018. 一种用于含二次方程、正则表达式和长度约束的字符串逻辑的判定过程，载于：Ryu, S.（编），《编程语言与系统——第16届亚洲研讨会，APLAS 2018》，新西兰惠灵顿，2018年12月2日至6日，论文集，施普林格。第350–372页。URL: https://doi.org/10.1007/978-3-030-02768-1_19, doi:10.1007/978-3-030-02768-1_19. Levi, F.W., 1944. 关于半群。Bulletin of the Calcutta Mathematical Society 36, 141–146. Li, G., Ghosh, I., 2013. PASS：使用参数化数组和区间自动机进行字符串求解，载于：Bertacco, V., Legay, A.（编），《硬件与软件：验证与测试——第九届国际海法验证会议，HVC 2013》，以色列海法，2013年11月5日至7日，论文集，施普林格。第15–31页。URL: https://doi.org/10.1007/978-3-319-03077-7_2, doi:10.1007/978-3-319-03077-7_2. Liang, T., Reynolds, A., Tinelli, C., Barrett, C.W., Deters, M., 2014. 用于字符串与正则表达式理论的DPLL(T)理论求解器，载于：《计算机辅助验证——第26届国际会议，CAV 2014》，施普林格。第646–662页。doi:10.1007/978-3-319-08867-9_43. Lin, A.W., Barcel´o, P., 2016. 使用字方程和转换器进行字符串求解：迈向用于分析变异型XSS的逻辑，载于：ACM SIGPLAN通报，ACM。第123–136页。Lin, A.W., Majumdar, R., 2018. 带长度约束的二次词方程、计数器系统以及带整除性的Presburger算术，载于：《自动化技术验证与分析——第16届国际研讨会，ATVA 2018》，施普林格。第352–369页。doi:10.1007/978-3-030-01090-4_21。

Loring, B.、Mitchell, D.、Kinder, J.，2019。动态符号执行中 JavaScript 的可靠的正则表达式语义，载于：第 40 届 ACM SIGPLAN 编程语言设计与实现会议论文集，第 425–438 页。

Makanin, G.S.，1977。自由半群中方程的可解性问题。Matematicheskii Sbornik 145，147–236。

Matiyasevich, Y.，2008。希尔伯特第十问题视角下的计算范式，载于：新计算范式。施普林格，第 59–85 页。

Matiyasevich, Y.V.，1968。字词方程与长度方程系统和希尔伯特第十问题之间的联系。Zap. Nauchnykh Semin. POMI 8，132–144。

Mora, F., Berzish, M., Kulczynski, M., Nowotka, D., Ganesh, V., 2021. Z3str4：一种多臂字符串求解器，载于：Huisman, M., Pasareanu, C.S., Zhan, N.（编），形式化方法——第24届国际研讨会 FM 2021，线上活动，2021年11月20日至26日，论文集，施普林格。第389–406页。网址：https://doi.org/10.1007/978-3-030-90870-6_21，doi:10.1007/978-3-030-90870-6_21。de Moura, L.M., Bjørner, N., 2008. Z3：一种高效的SMT求解器，载于：Ra- makrishnan, C.R., Rehof, J.（编），系统构造与分析工具和算法，第14届国际会议 TACAS 2008，作为软件理论与实践联合欧洲会议 ETAPS 2008 的一部分举行，布达佩斯，匈牙利，2008年3月29日至4月6日。论文集，施普林格。第337–340页。网址：https://doi.org/10.1007/978-3-540-78800-3_24，doi:10.1007/978-3-540-78800-3_24。Neider, D., Jansen, N., 2013. 使用求解器技术和自动机学习进行正则模型检测，载于：Brat, G., Rungta, N., Venet, A.（编），NASA形式化方法，第5届国际研讨会 NFM 2013，莫菲特机场，加利福尼亚州，美国，2013年5月14日至16日。论文集，施普林格。第16–31页。网址：https://doi.org/10.1007/978-3-642-38088-4_2，doi:10.1007/978-3-642-38088-4_2。Nielsen, J., 1917. 两个生成元的一般无限群的同构。Mathematische Annalen 78，385–397。doi:10.1007/ BF01457113。N¨otzli, A., Reynolds, A., Barbosa, H., Barrett, C.W., Tinelli, C., 2022. 字符串求解器中更快的冲突处理和更惰性的归约，载于：Shoham, S., Vizel, Y.（编），计算机辅助验证——第34届国际会议 CAV 2022，以色列海法，2022年8月7日至10日，论文集，第二部分，施普林格。第205–226页。网址：https://doi.org/10.1007/978-3-031-13188-2_11，doi:10.1007/978-3-031-13188-2_11。Osera, P.M., 2019. 基于约束的类型导向程序合成，载于：第4届 ACM SIGPLAN 类型驱动开发国际研讨会论文集，计算机协会，美国纽约州纽约。第64–76页。网址：https://doi.org/10.1145/3331554.3342608，doi:10.1145/3331554.3342608。

Pin, J. (Ed.), 2021. Handbook of Automata Th论。 E欧洲的 Mathematical 学会出版社，苏黎世，瑞士。网址：https://doi.org/10. 4171/Auto mata，doi:10.4171/Automata。Plandowski, W., 1999。含常量的词方程的可满足性属于 PSPACE，载于：第40届计算机科学基础年会（目录号 99CB37039），IEEE，第 495–500 页。Plandowski, W., 2006。求解词方程的高效算法，载于：第38届 ACM 计算理论年会论文集，ACM，第 467–476 页。Presburger, M., 1929。关于某个仅以加法为运算的整数算术系统的完备性，载于：斯拉夫国家数学家第一届大会，第 92–101 页。Quine, W.V., 1946。连接作为算术的基础。《符号逻辑杂志》11，105–114。Reynolds, A., Nötzli, A., Barrett, C., Tinelli, C., 2019。用于简化扩展字符串约束的高级抽象，载于：国际计算机辅助验证会议，施普林格，第 23–42 页。Reynolds, A., Woo, M., Barrett, C., Brumley, D., Liang, T., Tinelli, C., 2017。使用依赖上下文的简化扩展 DPLL(T) 字符串求解器，载于：国际计算机辅助验证会议，施普林格，第 453–474 页。Robson, J.M., Diekert, V., 1999。关于二次词方程，载于：计算机科学理论方面年会，施普林格，第 217–226 页。Schulz, K.U., 1990。用于字方程的 Makanin 算法——两项改进与一个推广，载于：词方程及相关主题国际研讨会，施普林格，第 85–150 页。Scott, J.D., Flener, P., Pearson, J., Schulte, C., 2017。有界长度序列变量的设计与实现，载于：Salvagnin, D., Lombardi, M.（编），人工智能与运筹学技术在约束编程中的集成——第14届国际会议 CPAIOR 2017，意大利帕多瓦，2017年6月5日至8日，会议论文集，施普林格，第 51–67 页。网址：https://doi.org/10.1007/978-3-319-59776-8_5，doi:10.1007/978-3-319-59776-8_5。Sen, K., Kalasapur, S., Brutch, T., Gibbs, S., 2013。Jalangi：一种面向 JavaScript 的选择性记录-重放与动态分析框架，载于：2013年第9届软件工程基础联合会议论文集，计算机协会，第 488–498 页。doi:10.1145/2491411.2491447。Stanford, C., Veanes, M., Bjørner, N., 2021。用于高效求解扩展正则表达式约束的符号布尔导数，载于：第42届 ACM SIGPLAN 编程语言设计与实现国际会议论文集，第 620–635 页。

Thatcher, J.W., Wright, J.B., 1968。广义有限自动机理论及其在二阶逻辑判定问题中的应用。《数学系统理论》2，57–81。网址：https://doi.org/10.1007/BF01691346，doi:10.1007/BF01691346。

Trinh, M.T., Chu, D.H., Jaffar, J., 2014年。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：ACM SIGSAC 计算机与通信安全会议论文集，ACM。第 1232–1243 页。

Trinh, M.T., Chu, D.H., Jaffar, J., 2016。递归定义字符串上的渐进式推理，载于：国际计算机辅助验证会议，施普林格。第 218–240 页。

Trinh, M.T., Chu, D.H., Jaffar, J., 2020。面向 SMT 字符串求解器的理论间依赖分析。《ACM编程语言会议论文集》4，1–27。

Tseitin, G.S., 1983。命题演算中推导复杂度，载于：推理自动化。施普林格，第 466–483 页。

Veanes, M., Hooimeijer, P., Livshits, B., Molnar, D., Bjørner, N.S., 2012。符号有限状态转换器：算法与应用，载于：Field, J., Hicks, M.（编），第 39 届 ACM SIGPLAN-SIGACT 编程语言原理研讨会论文集，POPL 2012，美国宾夕法尼亚州费城，2012 年 1 月 22–28 日，ACM。第 137–150 页。网址：https://doi.org/10.1145/2103656.2103674，doi:10.1145/2103656.2103674。

Wang, H., Chen, S., F. Yu, Jiang, J.R., 2018年。采用符号模型检测方法分析字符串和长度约束，载于：Huchard, M., K¨astner, C., Fraser, G.（编），第33届 ACM/IEEE 自动化软件工程国际会议论文集，ASE 2018，法国蒙彼利埃，2018年9月3日至7日，ACM，第623–633页。网址：https://doi.org/10.1145/3238147.3238189，doi:10.1145/3238147.3238189。

Wang, H., Tsai, T., Lin, C., F. Yu, Jiang, J.R., 2016。通过结合逻辑电路表示的自动机操作进行字符串分析，载于：计算机辅助验证——第28届国际会议，CAV 2016，施普林格，第241–260页。doi:10.1007/978-3-319-41528-4_13。

Wang, Y., Zhou, M., Jiang, Y., Song, X., Gu, M., Sun, J., 2017. 一种带有优化功能、用于可达性判定的静态分析工具，载于：2017年第32届IEEE/ACM 自动化软件工程国际会议（ASE），第925–930页。doi:10.1109/ASE.2017.8115706。Wolper, P., Boigelot, B., 1998. 验证具有无限但正则状态空间的系统，载于：Hu, A.J., Vardi, M.Y.（编），《计算机辅助验证》，第十届国际会议，CAV '98，温哥华，不列颠哥伦比亚省，加拿大，1998年6月28日至7月2日，论文集，施普林格。第88–97页。URL: https://doi.org/10.1007/ BFb0028736，doi:10.1007/BFb0028736。
Wolper, P., Boigelot, B., 2000. 关于从线性算术约束构造自动机，载于：Graf, S., Schwartzbach, M.I.（编），《工具

与系统构造和分析算法》，第六届国际会议，TACAS 2000，作为欧洲软件理论与实践联合会议 ETAPS 2000 的一部分举行，柏林，德国，2000年3月25日至4月2日，论文集，施普林格。第1–19页。URL: https: // doi.org/10.1007/3-540-46419-0_1，doi:10.1007/3-540-46419-0_1。

Wolper, P., Boigelot, B., 2002. 用有限自动机表示算术约束：综述，载于：Stuckey, P.J.（编），逻辑程序设计，第十八届国际会议，ICLP 2002，哥本哈根，丹麦，2002年7月29日至8月1日，论文集，施普林格。第1–19页。网址：https://doi.org/10. 1007/3-540-45619-8_1，doi:10.1007/3-540-45619-8_1。

Yu, F., Alkhalaf, M., Bultan, T., 2010. Stranger：一种用于 PHP 的基于自动机的字符串分析工具，载于：系统构造与分析工具和算法国际会议，施普林格。第154–157页。

Yu, F., Alkhalaf, M., Bultan, T., Ibarra, O.H., 2014年。用于漏洞检测的基于自动机的符号字符串分析。《系统设计形式化方法》44，44–70。doi:10.1007/s10703-013-0189-1。

F. Yu、C.Y. Shueh、C.H. Lin、Y.F. Chen、B.Y. Wang、T. Bultan，2016。用于 Web 应用程序漏洞修复的最优净化合成，载于：第 25 届软件测试与分析国际研讨会论文集，ACM，第 189–200 页。

Y. Zheng、V. Ganesh、S. Subramanian、O. Tripp、M. Berzish、J. Dolby、X. Zhang，2017。Z3str2：一种用于字符串、正则表达式和长度约束的高效求解器。《系统设计形式化方法》50，249–288。