arXiv:2302.06672v1 [cs.FL] 13 Feb 2023

无限词的确定性正则函数

OlivierCarton ！
巴黎西岱大学 ，CNRS, IRIF, F-75013 ，巴黎，法国
Gaëtan Douéneau-Tabot 巴黎西岱大学、法国国家科学研究中心、IRIF，F-75013，巴黎，法国；法国军备总局—项目工程部，巴黎，法国
Emmanuel Filiot
布鲁塞尔自由大学 & 比利时法语区国家科学研究基金会，布鲁塞尔，比利时
Sarah Winter
布鲁塞尔自由大学 & 比利时法语区国家科学研究基金会，布鲁塞尔，比利时

摘要

无限词上的正则函数是由带有无限前瞻的确定性双向转导器实现的（部分）函数。等价地，Alur 等人已证明，这些函数对应于由确定性 Muller 流式字符串转换器实现的函数，以及由单子二阶逻辑转导定义的函数。然而，正则函数通常不可计算（这里采用图灵可计算性对无限输入的一种经典扩展），本文考察由确定性正则函数构成的无限词函数类；这类函数由不带前瞻的确定性双向转换器实现。我们证明这是一个性质良好的函数类：其中的函数可计算、在复合下封闭，并且可由单子二阶逻辑转导的受保护片段、确定性 Büchi 流式字符串转换器、带有有限前瞻的确定性双向转换器，以及顺序函数与一种称为映射-复制-反转的固定基本函数的有限复合来刻画。

2012 年 ACM 学科分类 计算理论 → 形式语言与自动机理论

关键词和短语 无限词、流式字符串变换器、双向转换器、单子二阶逻辑、前瞻、因子分解森林

1 引言

转导器为自动机增添了输出机制，使有限状态机从语言识别器转变为函数计算模型。受 Engelfriet 和 Hoogeboom 的开创性工作启发 [21], ，过去十年间，研究者越来越关注如何刻画有限词上确定性双向转换器（2-dT）所定义的函数类；该函数类如今称为有限字的正则函数类。这个函数类有多种（有效）刻画方式：它对应于单子二阶逻辑转导所定义的函数 [21],、基于单子二阶逻辑的起源图逻辑所定义的函数 [14], 、由一种称为组合子表达式的正则表达式扩展所定义的函数 [4, 5, 19], ，以及由无复制流式字符串转换器（SST）计算的函数（这是一种确定性单向模型，使用寄存器存储并更新部分输出词 [2]）。此外，有限词上的正则函数类在复合下封闭 [10],，且其等价性问题是可判定的 [24]。

▶ **例1.1.** 设 Σ 为一个字母表，函数 map-copy-reverse : $(\Sigma \uplus \{|\})^* \to (\Sigma \uplus \{|\})^*$接受任意形如 $u_1|\dots|u_n$ 的字，其中每个 u_i 都不含 |，并输出 $u_1|\widetilde{u_1}|\dots|u_n|\widetilde{u_n}$，其中 $\widetilde{u_i}$ 是 u_i的镜像。函数 map-copy-reverse 是正则函数。

正则函数也可以刻画为顺序函数（由确定性单向有限转导器计算的函数）与映射-复制-反转 [6] 的复合。

无限词上的正则函数 正则函数这一类已扩展到 [3], 中的无限词，并定义为可由其上的单子二阶逻辑转导所定义的函数类

。等价地，已证明它们正是由带正则前瞻的确定性双向转换器以及带 Muller 选择条件的流式字符串转换器所实现的函数（持有最终输出词的寄存器取决于被无限次访问的状态集合）。与有限词的情形一样，无限词上的正则函数在复合运算下封闭，并且其等价性问题是可判定的 [3]。

▶ **例 1.2.** 设 $\Sigma = \{a, b, c\}$ 是一个字母表，并考虑函数 double：$\Sigma^\omega \to \Sigma^\omega$ 。该函数的行为与恒等函数相同，但如果某次出现的 a 之后还存在一个 b ，就将其替换为aa 。例如，$(ab)^\omega$ 映射为 $(aaab)^\omega$ ，而 $acaab(ac)^\omega$ 映射为 $aacaaaab(ac)^\omega$。函数 double 是正则函数，因为它可以由一个单向变换器实现：该变换器读取一个 a时，会使用正则前瞻来判断后续是否存在 b ，并据此输出 a 或 aa。

▶ 例 1.3. 设 $\Sigma' = \{a, b, 1, 2\}$，并考虑复制函数 copy，它将以下内容映射为：$u_1\sigma_1 u_2 \dots \sigma_n u \mapsto u_1^{\sigma_1}\sigma_1 \dots u_n^{\sigma_n}\sigma_n u$ 其中 $u_1 u_2 \dots u_n u \in \{a, b\}^\omega$ 且 $\sigma_1, \dots, \sigma_n \in \{1, 2\}$；$u_1\sigma_1 \dots u_i\sigma_i \cdots \mapsto u_1^{\sigma_1}\sigma_1 \dots u_i^{\sigma_i}\sigma_i \dots$（如果有无穷多个 $\sigma_i \in \{1, 2\}$）。例如，copy($ab2a1b^\omega$) $= abab2a1b^\omega$ ，而 copy($(a2)^\omega$) $= (aa2)^\omega$。复制函数 copy 是正则函数，例如可由确定性双向转换器实现。利用双向性，该转换器会对块 u_i进行一次或两次扫描，具体取决于其后是否跟着 $\sigma_i = 2$。在第一次扫描时，它总是输出所读取的内容；因此，如果之后再也没有读到分隔符 $\{1, 2\}$（这意味着它正在读取无限后缀 u），那么它就输出 u。

尽管无限词上的正则函数类具有稳健的性质，并且有多种刻画方式和算法性质作为佐证，但在可计算性方面，它们有一个严重的缺陷。事实上，存在不可计算的无限词上的正则函数。此处需要明确可计算性的含义，因为输入是无限的。关于可计算性的形式化定义，请读者参阅 [17, 18] （及其中的参考文献）；这里我们只给出直观解释。无限词上的函数 f 是可计算的，如果存在一台图灵机，它具有一条包含函数定义域中的某个无限输入词 u 的无限只读纸带、一条双向工作纸带，以及一条只写单向输出纸带；通过不断读取更长的输入前缀，该机器便能在输出纸带上写出 $f(u)$越来越长的前缀。简单来说，这是一种将输入作为流接收、并能将输出作为流生成的算法，使得机器能够在无限多个时刻至少输出一个符号。例如，上文的函数 double 不可计算。对于n不断增大的 ac^n 形式的前缀，它可以安全地输出一个 a 符号，但不能输出更多。事实上，如果它再输出一个 a，那么对于后续 c^ω 而言，这个输出就是错误的；如果它输出一个 c，那么对于后续 b^ω而言，这个输出也是错误的，因为 double(ac^nc^ω) $= ac^\omega$ ，而 double(ac^nb^ω) $= aac^nb^\omega$。用双向转换器实现该函数时，确实需要无限前瞻，以检查未来是否不会出现 b 。另一方面，copy 可由无需前瞻的确定性双向转换器实现，因此是可计算的。因此，带有（无限）前瞻的确定性双向转换器及其等价模型 Muller 流式字符串变换器不能被视为无限词函数的计算模型。 [18],中指出了这一点，其中证明了判定给定的无限词上的正则函数是否可计算这一问题是 PSPACE 完全的。另一方面，不带前瞻的确定性双向转换器是无限词函数的一种适当的计算模型。

无限词的确定性正则函数受后一观察的启发，不带前瞻的确定性双向转导器所计算的函数类，

被称为确定性正则函数，并于 [9], 中提出；该文证明，这些函数也等价地由 Büchi SST（BSST）计算。在BSST中，有一个专门指定的寄存器 $\mathfrak{out}$ 用于写入输出词，并且要求该寄存器无限多次至少更新一个新符号。例如，复制可以通过一个只有一个状态的BSST 实现，该 BSST 有两个寄存器 $\mathfrak{out}$ 和 $\mathfrak{r}$，并按如下方式更新。读取 $\sigma \in \{a, b\}$时，它执行更新 $\mathfrak{out} \mapsto \mathfrak{out}.\sigma$ 和 $\mathfrak{r} \mapsto \mathfrak{r}.\sigma$；读取 1 时，它执行 $\mathfrak{out} \mapsto \mathfrak{out}.1$ 和 $\mathfrak{r} \mapsto \varepsilon$；读取 2 时，它执行 $\mathfrak{out} \mapsto \mathfrak{out}.\mathfrak{r}.2$ 和 $\mathfrak{r} \mapsto \varepsilon$。

关于确定性正则函数类，仍有若干重要问题有待解决，例如它是否在复合下封闭、能否通过 MSO 转导的某个自然片段在逻辑上加以刻画，以及能否将其表示为“简单”函数的有限复合。本文对这些问题给出了肯定的回答。

贡献 关于确定性正则函数类，我们的主要结果如下：

- 它在复合下的有效封闭性；通过顺序函数的有限复合以及映射-复制-反转在无限词上的
- 扩展来刻画该类；通过 MSO 转导的一个自然句法片段——受限片段——对其进行逻辑刻画，称为 MSOT$_g$。

MSO 转导定义为一种 MSO解释，其中输出字结构的谓词（即后继关系和标签关系）分别由含有两个和一个自由一阶变量的 MSO 公式定义，并在输入的固定数量副本上进行解释。受限片段通过对构成 MSO 转导的 MSO 公式施加经典限制来定义（参见例如 [23]及其中的参考文献）。这些公式必须以前置存在量词 $\exists$g 开头，其中 g 是一个字位置，并且公式中的所有量词都受守卫 $x \leq$ g 的限制（对于任意集合变量X，还包括 $\forall x \in X, x \leq$ g）。因此，无限词上的受限 MSO 公式只能涉及有限前缀。再来考虑复制函数。为了处理块可能出现的重复，需要输入的两个副本；但是，输出词结构中两个节点之间是否存在后继边，只取决于局部性质，而这些性质可以用受限 MSO 公式定义。例如，这样的性质可以是：“如果位置 $x+1$ 标记为 2，那么 x 的第一个副本与其所属块的第一个位置的第二个副本之间存在一条后继边。”

一般而言，受限 MSO 公式能够检验非局部性质，这是本文中技术难点的主要来源。下一个例子对此进行了说明。

▶ 例 1.4. 函数 replace：$\{0, a, b\}^\omega \multimap \{a, b\}^\omega$，其定义域为 Dom(replace) $=\{u \in \{0, a, b\}^\omega : |u|_a = \infty$ 或 $|u|_b = \infty\}$，映射为 $0^{n_1}\sigma_1 0^{n_2}\sigma_2 \cdots \mapsto \sigma_1{}^{n_1+1}\sigma_2{}^{n_2+1}\cdots$ 若 $\sigma_i \in \{a, b\}$ 且 $n_i \in \mathbb{N}$，是确定性正则的。在位置 x 将零替换为 a 或 b，取决于 x之后未来出现的下一个非零符号；这个符号可能远在后方，但若 $u \in$Dom(replace)，则会出现在有限前缀中。该性质可以用受限 MSO 公式表达，该公式将包含这个非零符号的位置定义为一个守卫。

证明技术与其他结果 现在，我们概述用于证明这一逻辑刻画的证明技术，并介绍一些其他有趣且实用的结果。我们证明，确定性双向转换器（2-dT）在表达能力上等价于 MSOT$_g$。将 2-dT 转换为 MSOT$_g$ 是标准做法，沿用与 [21]相同的思路。反向转换则更为复杂，需要采用新的技术。首先，我们将 MSO$_g$-转导转换为带有有限前瞻 的确定性双向转换器（2-dT^{FLA}），这种转换器

能够处理前文所述的非局部但有限的性质。2-dT^{FLA} 在其转移上配备了有限词的正则语言，这些语言以如下方式充当有限前瞻：当读头位于无限词 u的某个位置 i，且处于某个状态 q时，如果存在一个位置 $j \geq i$（称为见证），使得从位置 i 开始、到位置 j结束的中缀 $u[i{:}j]$ 属于L，那么带有前瞻 L 的、从 q 出发的转移便会启用。如果状态 q下没有转移启用，计算就会失败。为保证确定性，若有多个转移启用，则只触发见证 j 最小（即最靠前）的转移；此外，前瞻之间的不相交性要求确保该见证 j 唯一。只考虑见证 j 最小的转移这一条件，对于确保 2-dT^{FLA} 仅定义可计算函数至关重要。事实上，2-dT^{FLA} 可以按如下方式执行：将所有有限前瞻（例如，假设它们由确定有限自动机有限表示）并行执行。根据 j 的最小性要求以及前瞻之间的不相交性，只要某个前瞻确定有限自动机接受一个前缀，便会触发相应的转移。

在有限词上，为了捕获 MSO 转导而向双向转换器中添加前瞻是标准做法 [21, 13], 例如，这是因为 MSO 转导的“移动”取决于非局部性质。随后，可以直接利用确定性双向转换器的复合封闭性 [10]来消除前瞻。据我们所知，无限词上的确定性双向转换器对于复合的封闭性尚属未知，因此我们改为直接证明有限前瞻消除。这是我们的主要技术成果：任意 2-dT^{FLA} 都能有效地等价于某个 2-dT。为了证明这一结果，Hopcroft-Ullman 构造 [1] 或树轮廓构造 [15] 等经典技术并不适用，因为它们严重依赖于单词是有限的这一事实。在我们的设定中，我们改用一种新技术：将前瞻的计算摘要为树，并证明这些树是有界的。作为有限前瞻消除的一个附带结果，我们证明 2-dT （因而确定性正则函数）对于复合是封闭的。经典地，MSO 转导对于复合的封闭性可通过公式替换 [13]直接得到。然而，这一技术在此并不适用，因为受限 MSO 公式在语法上并不对公式替换封闭，这使得 MSOT$_g$ 与 2-dT 之间的对应关系对于获得 MSO$_g$-转导的复合封闭性至关重要。

论文结构 第 2 节介绍确定性正则函数这一类。第 3 节证明其具有复合封闭性，并给出分解结果。第 4 节介绍受限 MSO 转导，并陈述逻辑刻画。由于该刻画的证明基于编译为带有限前瞻的确定性双向转换器，我们在第 5 节证明如何移除这些前瞻。最后，我们在第 6 节证明逻辑刻画。本文中的所有变换都是有效的。有些证明仅作了简要概述或直接省略，但详细证明可见附录。

2 确定性正则函数

本节介绍无限词的确定性正则函数，并回顾它可以由两种计算模型描述：确定性双向转换器 和确定性 *Büchi* 流式字符串转换器。

记号 字母 Σ, Γ 表示字母表，即由有限个字母组成的集合。集合 Σ^* （分别为 Σ^+、Σ^ω）表示字母表 Σ 上的有限词集（分别为非空有限词集、无限词集）。设 $\Sigma^\infty := \Sigma^* \cup \Sigma^\omega$。若 $u \in \Sigma^\infty$，则令 $|u| \in \mathbb{N} \cup \{\infty\}$ 表示其长度，$|u|_\sigma \in \mathbb{N} \cup \{\infty\}$表示 $\sigma \in \Sigma$ 出现的次数，$u[i] \in \Sigma$ 表示 u 的第 i 个字母，其中 $1 \leq i \leq |u|$。

若 $1 \le i \le j \le |u|$，则 $u[i{:}j]$ 表示 $u[i]\cdots u[j]$。我们将 $u[i{:}]$ 记作 $u[i{:}|u|]$。若 $j > |u|$，则令$u[i{:}j] := u[i{:}|u|]$。若 $j < i$，则令 $u[i{:}j] := \varepsilon$。本文中，函数默认为部分函数（即定义域可能不是全定义域）。从 S 到 T 的（部分）函数 f 记作 $f\colon S \rightharpoonup T$，其定义域记作 定义域$(f) \subseteq S$。从 S 到 T 的全函数记作 $f\colon S \to T$。

双向转换器我们先回顾双向转换器的语法。这里我们假设机器处理无限词，并具有 Büchi 接受条件。

▶ 定义 *2.1*（双向转换器）。确定性双向转换器（*2-dT*）记作 $T = (\Sigma, \Gamma, Q, q_0, F, \delta, \lambda)$，由以下部分组成：输入字母表 Σ 和输出字母表 Γ；有限状态集 Q，其中包含初始状态 $q_0 \in Q$ 和终态集 $F \subseteq Q$；转移函数 $\delta\colon Q \times (\Sigma \uplus \{\vdash\}) \rightharpoonup Q \times \{\triangleleft, \triangleright\}$；输出函数 $\lambda\colon Q \times (\Sigma \uplus \{\vdash\}) \rightharpoonup \Gamma^*$，其定义域与 δ 相同。

T 在 $u \in (\Sigma \cup \{\vdash\})^\infty$ 上的一个配置是一个元组 (q, i)，其中 $q \in Q$ 是当前状态，$1 \le i \le |u|$ 是读头的当前位置。转移关系 $\to$ 定义如下。给定配置 (q, i)，令 $(q', \star) := \delta(q, u[i])$。那么，当 $\star = \triangleleft$ 且 $i' = i - 1$（向左移动），或 $\star = \triangleright$ 且 $i' = i + 1$（向右移动）时，$(q, i) \to (q', i')$。在 u 上的一次运行是由连续配置 $(q_1, i_1) \to (q_2, i_2) \to \cdots$ 构成的（有限或无限）序列。

现在，我们定义转导器 T 以无限词 $u \in \Sigma^\omega$ 为输入时产生的无限输出。首先，令 $u[0] := \vdash$，也就是说，强制使用符号 $\vdash$ 标记输入的开头。接受运行是指从 $(q_0, 0)$ 开始，并且无限次访问形如 (q, i)、满足 $q \in F$ 的配置，同时当 $n \to \infty$ 时满足 $i_n \to \infty$ 的无限运行（若不满足最后这个条件，转导器可能会进入无限循环而不读完整个输入）。由 T 计算的偏函数 $f\colon \Sigma^\omega \rightharpoonup \Gamma^\omega$ 定义如下。设 $u \in \Sigma^\omega$ 满足存在一个（唯一的）接受运行 $(q_0^u, i_0^u) \to (q_1^u, i_1^u) \to \cdots$，其标签为 $\vdash u$。令 $v := \prod_{j=1}^{\infty} \lambda(q_j^u, (\vdash u)[i_j^u]) \in \Gamma^* \cup \Gamma^\omega$ 为沿此运行产生的输出的拼接。若 $v \in \Gamma^\omega$，则定义 $f(u) := v$。否则，$f(u)$ 未定义。

▶ **定义 2.2**。无限词上的确定性正则函数 类，是由确定性双向转换器计算的（部分）函数类。

我们在例 1.3 中解释了如何使用 2-dT（不带前瞻）计算复制函数。注意，例 1.4 中的 replace 函数也可以用类似方式计算。因此，这两个函数都是确定性正则的。

▶ **例2.3**。我们将例 1.1 中的映射-复制-反转函数扩展到无限词。设 Σ 为一个字母表，我们定义映射-复制-反转：$(\Sigma \uplus \{|\})^\omega \to (\Sigma \uplus \{|\})^\omega$ 如下：

map-copy-reverse$(u_1|u_2|\cdots) := u_1|\widetilde{u_1}|u_2|\widetilde{u_2}|\cdots$ 其中 $u_i \in \Sigma^*$ 对所有 $i \ge 0$均成立；
map-copy-reverse$(u_1|\cdots|u_n|u) := u_1|\widetilde{u_1}|\cdots|u_n|\widetilde{u_n}|u$，其中 $u_i \in \Sigma^*$ 且 $u \in \Sigma^\omega$。该函数是确定性正则函数，因为我们可以构造一个 2-dT，对每个$\vdash$自由因子处理两次（如果存在最后一个无限因子，则只需处理一次）。

Büchi流式字符串转换器 现在，我们描述一种带寄存器的单向机器模型，它能够刻画无限词的确定性正则函数。对于有限词，众所周知，确定性双向转换器等价于无复制流式字符串转换器 [2]。对于无限词上的正则函数这一类，也存在类似的等价关系：它们既可以由带正则前瞻的确定性双向转换器 描述，也可以由带*Muller*条件的无复制流式字符串转换器 [3] 描述。然而，

Muller条件能够检验无限输入的正则性质，因此可以描述不可（图灵）计算的函数 [3]。现在，让我们回顾一下由 Carton 和 Douéneau-Tabot 在 [9],中引入的*Büchi*确定性流式字符串转换器 （BSST）模型；它恰好刻画了确定性正则函数这一类。

形式上，Büchi确定性流式字符串转换器由一个单向确定性自动机组成，该自动机具有有限集合 $\mathfrak{R}$ 个寄存器，用于存储来自 Γ^* 的单词。我们使用一个特殊的寄存器 out 来存储读取无限词时产生的输出。读取输入时，通过替换修改寄存器，即映射 $\mathfrak{R} \to (\Gamma \uplus \mathfrak{R})^*$。我们用$\mathcal{S}_{\mathfrak{R}}^{\Gamma}$ 表示这些替换的集合。通过保持 Γ 中的元素不变，可以将它们从 $(\Gamma \uplus \mathfrak{R})^*$ 按态射扩展到 $(\Gamma \uplus \mathfrak{R})^*$ 。

▶ **例 2.4** （替换）。 设 $\mathfrak{R} = \{\mathfrak{r}, \mathfrak{s}\}$ 以及 $\Gamma = \{b\}$。考虑 $\tau_1 := \mathfrak{r} \mapsto b, \mathfrak{s} \mapsto b\mathfrak{r}\mathfrak{s}b$ 和 $\tau_2 := \mathfrak{r} \mapsto \mathfrak{r}b, \mathfrak{s} \mapsto \mathfrak{r}\mathfrak{s}$，则 $\tau_1 \circ \tau_2(\mathfrak{r}) = \tau_1(\mathfrak{r}b) = bb$ 以及 $\tau_1 \circ \tau_2(\mathfrak{s}) = \tau_1(\mathfrak{r}\mathfrak{s}) = bb\mathfrak{r}\mathfrak{s}b$。

▶ 定义 *2.5.* 一个 *Büchi*确定性流式字符串转换器（*BSST*），记作 $T = (\Sigma, \Gamma, Q, F, q_0, \delta, \mathfrak{R}, \mathrm{out}, \lambda)$，由以下部分组成：有限的输入（分别为输出）字母表 Σ （分别为 Γ）；有限状态集 Q，其中 $q_0 \in Q$ 为初始状态，$F \subseteq Q$ 为最终状态；转移函数 $\delta : Q \times \Sigma \rightharpoonup Q$；有限寄存器集 $\mathfrak{R}$，其中有一个指定的输出寄存器 $\mathrm{out} \in \mathfrak{R}$；更新函数 $\lambda : Q \times \Sigma \rightharpoonup \mathcal{S}_{\mathfrak{R}}^{\Gamma}$，满足对于所有 $(q, \sigma) \in \mathit{Dom}(\lambda) = \mathit{Dom}(\delta)$：$\lambda(q, \sigma)(\mathrm{out}) = \mathrm{out} \cdots$；在 $\lambda(q, \sigma)(\mathfrak{r})$（$\mathfrak{r} \in \mathfrak{R}$）中，$\mathrm{out}$ 不会在其他位置出现。

该机器定义了一个偏函数 $f : \Sigma^\omega \rightharpoonup \Gamma^\omega$，如下所示。对于 $i \geq 0$，令 $q_i^u := \delta(q_0, u[1{:}i])$（若有定义）。对于 $i \geq 1$，令 $\lambda_i^u := \lambda(q_{i-1}^u, u[i])$（若有定义），并且对所有 $\mathfrak{r} \in \mathfrak{R}$，$\lambda_0^u(\mathfrak{r}) = \varepsilon$。对于 $i \geq 0$，令 $[\![\cdot]\!]_i^u := \lambda_0^u \circ \cdots \circ \lambda_i^u$。根据构造，$[\![\mathrm{out}]\!]_i^u$ 是$[\![\mathrm{out}]\!]_{i+1}^u$ 的前缀（若有定义）。如果对于所有 $i \geq 0$，$[\![\mathrm{out}]\!]_i^u$ 都有定义，q_i^u是 F 的状态且出现无穷多次，并且 $|[\![\mathrm{out}]\!]_i^u| \to +\infty$，那么令 $f(u) := \bigvee_i [\![\mathrm{out}]\!]_i^u$（符号 $\vee$用于表示唯一的 $v \in \Gamma^\omega$，使得对于所有 $i \geq 0$，$[\![\mathrm{out}]\!]_i^u$ 都是 v 的前缀）。否则，$f(u)$ 无定义。

▶ **例 2.6. </style}例 1.4 中的函数 replace 可以由 BSST** 计算。对于所有 $i \geq 1$，它都会遍历块 0^{n_i}，并在两个寄存器中计算 1^{n_i} 和 2^{n_i}。一旦看到 σ_i，它就会将存储 ${\sigma_i}^{n_i}$ 的寄存器加入 out。

▶ 定义 *2.7*（无复制、有界复制）。如果对于所有 $\mathfrak{r} \in \mathfrak{R}$，$\mathfrak{r}$ 在 $\{\tau(\mathfrak{s}) : \mathfrak{s} \in \mathfrak{R}\}$ 中至多出现一次（分别地，对于所有 $\mathfrak{r}$，$\mathfrak{s} \in \mathfrak{R}$ 在 $\tau(\mathfrak{s})$ 中至多出现 K 次），则称替换 $\tau \in \mathcal{S}_{\mathfrak{R}}^{B}$ 是无复制的（分别地，是 K-有界的）。如果对于所有使 $\lambda_i^u \circ \cdots \circ \lambda_j^u$ 有定义的 $u \in \Sigma^\omega$ 和 $i \leq j$，该替换都是无复制的（分别地，是 K-有界的），则称 *BSST* $T = (\Sigma, \Gamma, Q, q_0, \delta, \mathfrak{R}, \mathrm{out}, \lambda)$ 是无复制的（分别地，是 K-有界的）。

▶ 备注 2.8。两个无复制替换的复合仍然是无复制的，因此，只要对于所有 $q \in Q$ 和 $\sigma \in \Sigma$，$\lambda(q, \sigma)$ 都是无复制的，BSST 就是无复制的。然而，K-有界性在复合下不一定保持。

注意，例 2.6 中描述的BSST 是无复制的。现在，我们回顾 Carton 和 Douéneau-Tabot 的结果，该结果证明了双向转换器、无复制且有界复制的 Büchi确定性流式字符串转换器之间的等价性。

▶ **定理 2.9** ([9, 定理 3。7])。 以下机器计算的是同一类无限词上的偏函数：

1. 确定性双向转换器(2-dT)；

2. K-有界确定性 *Büchi* 流式字符串转换器(K-有界BSST)；3. 无复制确定性 *Büchi* 流式字符串转换器(无复制BSST)。此外，所有转换都是可有效实现的。

▶ 注 2.10。 [9] 将 2-dT 转换为 BSST 的原始证明只考虑所有状态均为终态的机器。不过，该证明很容易推广到具有非终态的转换器。此外，给定一个BSST （可能具有非终态），根据 [9, 引理 D.1] ，可以构造一个与之等价且所有状态均为终态的BSST （Büchi 条件隐含在输出必须是无限的这一事实中）。总而言之，所有这些模型（无论状态是否均为终态）都恰好刻画了确定性正则函数这一类。

最后，我们回顾确定性正则函数的定义域。若一个语言由确定性 Büchi 自动机接受（例如参见 [25]），则称其为*Büchi* 确定性 语言。

▶ **命题 2.11** ([9])。 若 f 是确定性正则函数，则定义域(f) 是 *Büchi* 确定性的。

3 复合与分解定理

本节将证明，确定性正则函数在复合运算下封闭；反过来，它们也可以写成某些“基本”函数的复合。

自 [10]（分别为 [3]）以来，人们已知有限词（分别为无限词）上的正则函数类在复合运算下封闭。我们在定理 3.1 中将这一结果推广到无限词的确定性正则函数。不过，该定理的证明并非正则情形的直接推广，它也揭示了本文的主要难点：由于不允许使用前瞻，2-dT 要检查某个性质是否在其当前位置之后 成立，是一件复杂的事。

▶**定理 3.1**。 确定性正则函数在复合运算下（有效地）封闭。

证明思路。 这种方法是直接复合两个转导器（使用乘积构造）；复合两个计算时的困难在于，一个转导器向前移动，而另一个向后移动。在这种情况下，我们需要将向后移动的转导器的计算回退一个计算步骤。

为了恢复之前的配置，前瞻非常有用。如上文所述，不允许使用（无限）前瞻，但我们采用一种较弱形式的有限前瞻（将在第 5 节中介绍）；这种前瞻不会提高无限词上确定性双向转换器的表达能力（并且可以有效地移除），参见定理 5.2。有限前瞻用于处理非局部但有限的性质。我们定义的前瞻基本上重新追踪双向转换器迄今为止所执行的计算。注意，这确实是一种有限性质，因为双向转换器的计算只访问了输入的一个前缀。◀

作为定理 3.1 的一个直接推论，我们注意到，确定性正则函数通过逆像（有效地）保持 Büchi 确定性语言。对于带有正则语言的有限词（分别为无限词）的正则函数，也有类似的结果。

▶ **命题 3.2**。 如果 $f\colon \Sigma^\omega \multimap \Gamma^\omega$ 是确定性正则的，且 $L \subseteq \Gamma^\omega$ 是 *Büchi* 确定性的，那么 $f^{-1}(L) \subseteq \Sigma^\omega$ 是（有效的）*Büchi* 确定性的。

证明。 函数 $f \circ \mathrm{id}_L$ (其中 $\mathrm{id}_L\colon \Gamma^\omega \multimap \Gamma^\omega$ 是限制在 L上的恒等函数) 是确定性正则函数。其定义域 $f^{-1}(L)$ 根据命题 2.11 是 Büchi 确定性的。◀

现在我们来关注定理 3.1 的逆命题，即证明任何确定性正则函数都可以表示为“基本”函数的复合。正如引言中提到的，有限字的正则函数可以写成映射-复制-反转（见例1.1）和顺序函数（由单向变换器计算）的复合。

▶**定理 3.3** ([6, 定理 13])。 对于有限词，一个函数是正则函数，当且仅当它可以（有效地）写成映射-复制-反转 函数与顺序函数的复合。

为了陈述关于无限词的确定性正则函数的类似结果，我们先正式回顾无限词上的顺序函数的定义。

▶ **定义 3.4** (顺序函数)。 一个确定性单向变换器 是一个2-dT $(\Sigma, \Gamma, Q, q_0, F, \delta, \lambda)$，满足对于所有 $q \in Q$ 和 $\sigma \in (\Sigma \uplus \{\vdash\})$， $\delta(q, \sigma)$ （有定义时）具有形状 $(_, \triangleright)$。由单向确定性变换器计算的无限词上的（部分）函数类称为无限词上的顺序函数。

▶ **例 3.5**。 任何将输入中的某个字母替换为另一个字母的函数都是顺序函数。例 2.3 和例 1.4 中的函数 replace 和映射-复制-反转都不是顺序函数（可用抽引论证证明这一点）。注意，replace 可以写成以下函数的复合：一个顺序函数，它将每个 $\sigma_i \in \{1,2\}$ 替换为 $\sigma_i|$；函数映射-复制-反转；最后，一个顺序函数，它利用每个块的第一个复制来确定 σ_i的值，并相应地变换（镜像后的）第二个复制。

现在，我们陈述分解结果，该结果也用到了例2.3中的映射-复制-反转。其证明在某种程度上较为技术性，并再次揭示了本文的主要难点：确定性正则函数无法检验许多关于“未来”的性质。

▶**定理 3.6**。 一个函数是确定性正则函数，当且仅当它可以（有效地）表示为映射-复制-反转 与无限词上的顺序函数的复合。

证明思路。 对于有限词， [7, 6] 的证明依赖于 Simon 的因子分解森林定理 [27]。它们首先构造一个因子分解森林，然后利用其结构来模拟转导器的运行。此外，在有限词上，这类森林可以通过一个有理函数计算得出，而该函数是顺序函数与映射-复制-反转的复合。对于无限词，我们采用类似的证明思路，但主要问题在于，因子分解森林无法再通过顺序函数与映射-复制-反转 的复合来计算（其结构可能取决于输入的正则性质）。因此，我们改用 Colcombet 提出的弱化版森林，称为前向 *Ramsey* 分裂 [11]。这类分裂可以通过顺序函数计算得出。我们的新方法展示了如何利用前向 Ramsey 分裂来模拟转导器的运行。◀

4 受限 MSO 转导

本节将定义有限词和无限词上的单子二阶逻辑（MSO）、MSO 转导及其受限片段。我们还将陈述确定性正则函数的逻辑刻画（定理 4.8）。

关于无限词的单子二阶逻辑 定义域为 $\mathbb{N}$ 的 Σ 上的无限词被视为结构，其签名为 $\mathcal{W}_\Sigma = \{S(x,y), (\sigma(x))_{\sigma\in\Sigma}\}$，由后继谓词 $S(x,y)$（按 $\mathbb{N}$ 上的后继关系自然解释）以及对所有 $\sigma \in \Sigma$ 的一元谓词 $\sigma(x)$组成，

解释为由 σ 标记的位置集合。给定一个无限词 $u \in \Sigma^\omega$，我们用 G_u 表示它所诱导的结构；当 u 明确表示结构 G_u 时，简记为 u。

单子二阶公式定义为一阶逻辑公式，并且可以额外使用量词 $\exists X$，$\forall X$ 对位置集合进行量化，以及形如 $x \in X$ 的隶属原子公式，其中 x 是一阶变量，而 X 是集合变量。我们用 MSO$[\Sigma, S, \leq]$（或者在谓词可从上下文中明确时简称 MSO）表示在扩展了序谓词$\leq$（其解释为 $\mathbb{N}$ 上的自然序）的词签名 $\mathcal{W}_\Sigma$ 上的单子二阶公式集合。众所周知，谓词 $\leq$ 只是语法上的简写。语义按预期方式定义（例如，详情可参见 [28, 13]）。对于一个公式 ϕ，其自由一阶变量和集合变量的集合为 x，X（我们使用元组记法，其中隐含地假设变量之间存在一个顺序），可以写作 $\phi(x, X)$，以明确指出 ϕ 的自由变量。我们还用 $Free(\phi)$ 表示 ϕ 的自由（一阶和集合）变量。给定一个字 w、一个由 w 中的位置组成的 n 元组 p，以及一个由 w 的位置集合构成的 m 元组 P，我们写作 $w \models \phi(p, P)$，表示由 w 诱导的结构在赋值 p 和 P 下是 ϕ 的模型。

▶ **例 4.1.** 公式 first$(x) = \forall y \cdot \neg S(y, x)$ 对任意词及满足条件的位置x 都成立：x 是左侧的第一个位置。

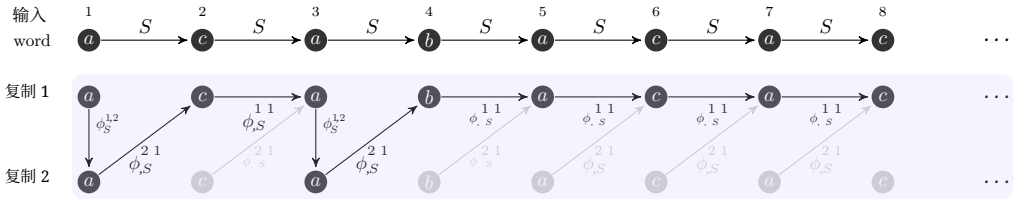
在字母表 Σ 上，任意封闭的 MSO公式 $\phi \in$ 定义 一个正则语言 $L_\phi = \{u \in \Sigma^\omega \mid u \models \phi\}$。根据 Büchi-Elgot-Trakhtenbrot 定理 [29, 8, 20]，已知 MSO 恰好定义了字母表 Σ 上的正则语言类：对于 Σ 上的任意语言 L，L 是正则语言，当且仅当存在某个 $\phi \in$MSO公式使得 $L = L_\phi$。MSO 公式也可以在有限词结构上解释，其定义域是词位置组成的（有限）集合。众所周知，有限词语言是正则语言，当且仅当它可以由 MSO 定义。

无限词的 MSO-**转导** MSO 转导定义了图结构之间的变换。Engelfriet 和 Hoogeboom 曾在有限词的背景下研究过这类转导，见 [21]（关于 MSO 转导的较新介绍另见 [13]）。[21] 的主要结果是一个 Büchi 型定理：有限词上的函数是 MSO 可定义的，当且仅当它是正则的（即可以由确定性双向转换器识别）。这一结果随后在 [3]，被推广到无限词上的函数；然而，要完整体现 MSO 转导的表达能力，确定性双向转换器可能需要无限前瞻。

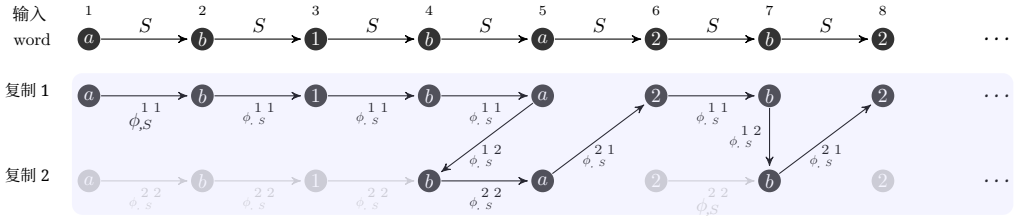
在 MSO 转导中，输出词结构通过对输入词（视为一种结构）的固定数量 k 个副本进行 MSO 解释来定义。因此，输出词的节点是输入词节点的第 1 至 k 个副本。输出节点是对 (i, c)（通常记作 i^c），对每个副本 c 和输入节点 i 均如此。

输出标签和后继谓词分别由含一个和两个自由一阶变量的 MSO 公式定义，并在输入结构上解释。例如，对于输出字母表 $\Gamma = \{a, b\}$，若要将所有输出标签设为 a，只需为所有副本 c 指定公式 $\phi_a^c(x) = \top$ 和 $\phi_b^c(x) = \bot$。输出后继谓词将可能属于不同副本的输入节点关联起来，因此由以下形式的公式定义：$\phi^{c,d}_S(x, y)$，其索引为副本 $c, d \in \{1, \ldots, k\}$。

最后，还有一个特殊的复制 c_0，以及一个公式 $\phi^{c_0}_{\text{fst}}(x)$，该公式最多只能由一个节点 x 满足。直观地说，如果输出结构是一个字，那么这个公式就定义了输出词的第一个节点。输出结构的定义域由所有能够从初始节点 x^{c_0} 出发、沿着多条后继边到达的节点组成。一般而言，输入词 u 经由 MSO 转导$\mathcal{T}$得到的输出结构可能不是无限词结构；在这种情况下，u 不属于由$\mathcal{T}$定义的函数的定义域。



(a) 例 1.2 中函数double的 MSO 转导在输入上的输入结构和输出结构 t
字 $acab(ac)^\omega$



(b) 例 1.3 中函数复制在输入词 $ab1ba2b2\ldots$ 上对应的 MSO 转导的输入和输出结构

形式上，关于输入字母表 Σ 和输出字母表 Γ 的一个 MSO-转导 是一个元组 $\mathcal{T} = (k, (\phi^c_\gamma(x))_{1\le c\le k,\gamma\in\Gamma}, (\phi^{c,d}_S(x,y))_{1\le c,d\le k}, c_0, \phi^{c_0}_{\mathrm{fst}}(x))$，其中 $k \in \mathbb{N}\setminus\{0\}$，并且对于所有输入 $u \in \Sigma^\omega$，至多存在一个位置 i 满足 $u \models \phi^{c_0}_{\mathrm{fst}}(i)$。我们可以省略上述元组中的$c_0$ 。

现在我们正式定义 MSO 转导的语义。令 $u \in \Sigma^\omega$ 和 $N \subseteq \mathbb{N}\times\{1,\ldots,k\}$。我们首先定义从 N经过零步或多步可以到达的输出节点集合。令 $\mathsf{Post}^0_u(N) = N$，并且对于所有 $\ell > 0$,

$$\mathsf{Post}^\ell_u(N) = \{j^d \mid \exists i^c \in \mathsf{Post}^{\ell-1}_u(N) \cdot u \models \phi^{c,d}_S(i,j)\} \text{ and } \mathsf{Post}^*_u(N) = \bigcup_{\ell\ge 0} \mathsf{Post}^\ell_u(N)$$

给定如上所述的 MSO 转导 $\mathcal{T}$以及输入词 $u \in \Sigma^\omega$，其输出结构 e, 记作$\mathcal{T}(u)$，是签名 $\mathcal{W}_\Gamma$ 上由以下解释定义的结构：

- 定义域为 $D = \mathsf{Post}^*_u(\{i^{c_0} \mid u \models \phi^{c_0}_{\mathrm{fst}}(i)\})$（注意，$\mathsf{Post}^*_u$ 的参数要么为空，要么是单元素集合）；若 $u \models \phi^c_\gamma(i)$，则节点 $i^c \in D$ 带有标签 $\gamma \in \Gamma$；若 $u \models \phi^{c,d}_S(i,j)$，则节点 j^d 是节点 i^c 的后继。

输出结构$\mathcal{T}(u)$可能不是词结构。例如，一个节点可能具有多个标签，$\mathcal{T}(u)$可能包含环或分支。因此，我们在语义上将$\mathcal{T}$定义的函数限制在词结构上。形式上，由 $\mathcal{T}$ 定义的函数是函数 $[\![\mathcal{T}]\!] : \Sigma^\omega \rightharpoonup \Gamma^\omega$ 其图为：

$$\{(u,v) \in \Sigma^\omega \times \Gamma^\omega \mid G_v \text{（与 } v \text{ 关联的结构）同构于} \mathcal{T}(u)\}$$

我们用 MSOT 表示单子二阶逻辑转导的集合，并称函数 $f : \Sigma^\omega \rightharpoonup \Gamma^\omega$是 MSOT 可定义的，当且仅当存在某个$\mathcal{T} \in$ 属于 MSOT，使得 $f = [\![\mathcal{T}]\!]$。

▶ **例 4.2.** 我们再次考虑例 1.2 中的函数double（如图 1a 所示），并展示如何用单子二阶逻辑转导来定义它。由于某些 a 必须被复制，因此需要两个副本，即 $k = 2$。标签保持不变：对于所有 $c \in \{1,2\}$和 $\sigma \in \Sigma$，$\phi^c_\sigma(x) = \sigma(x)$。第一个副本 c_0 是 1，而 $\phi^{c_0}_{\mathrm{fst}}(x) = \mathsf{first}(x)$。后继公式会区分未来是否存在一个 b 。首先，从第二个副本到第一个副本，总是存在一条

后继关系，将一个节点连接到副本 1 中它的后继节点：$\phi^{2,1}_S(x,y) = S(x,y)$。如果 $x = y$，x 被标记为 a，并且从 x开始的剩余无限后缀中存在一个 b，那么就存在一条从 x^1 到 y^2 的后继关系：$\phi^{1,2}_S(x,y) = a(x) \wedge (x = y) \wedge \exists z \cdot\ x \leq z \wedge b(z)$。在第一个副本中，这取决于输入的标签：$\phi^{1,1}_S(x,y) = S(x,y) \wedge (a(x) \rightarrow (\forall z \geq x \cdot\ \neg b(z))$。在第二个副本中，从不出现谓词边：$\phi^{2,2}_S = \bot$。图 1a 展示了这些公式的解释；属于输出词的部分以粗体表示，其余部分则以浅灰色表示。可以看出，由第一个节点的所有后代（通过后继关系的传递闭包）诱导出的输出结构，与结构 $G_{aacaab(ac)^\omega}$同构。

例 1.3 中的复制函数（如图 1b 所示）可由具有两个副本的 MSO 转导定义（$k = 2$）。公式 $\phi^{\text{co}}_{\text{fst}}$ 和 ϕ^c_σ 与 double 中的相同。于是：

$$\phi^{1,1}_S(x,y) = \phi^{2,2}_S(x,y) = S(x,y) \wedge \neg 2(y) \qquad \phi^{2,1}_S(x,y) = S(x,y) \wedge 2(y)$$
$$\phi^{1,2}_S(x,y) = \exists \mathsf{g} \cdot y < x \leq \mathsf{g} \wedge 2(\mathsf{g}) \wedge \forall z \leq y \cdot (S(z,y) \rightarrow (1(z) \vee 2(z))) \wedge$$
$$\forall t \cdot (y \leq t \leq x) \rightarrow (a(t) \vee b(t))$$

无限词上的正则函数类在 [3] 中被定义为可由扩展了正则（无限）前瞻的确定性双向转换器识别的函数类：为了执行一次转移，这种转换器可以在当前无限后缀上查询正则预言机（例如，将其表示为一个确定性优先自动机）。等价地，该类对应于可由（确定性）SST识别的函数：它们的工作方式与BSST 相同，但不要求无限次输出某个特殊寄存器的内容。取而代之的是，运行的输出取决于该运行过程中无限次出现的状态集合，并且只有在处理完无限输入后才能“计算”出来（更多细节见 [3]）。下面给出正则函数类的逻辑刻画：

▶ **定理 4.3** ([3])。 函数 $f\colon \Sigma^\omega \rightharpoonup \Gamma^\omega$ 是正则的，当且仅当它可以由MSOT定义。

文献 [3] 中的 MSOT 定义略有不同，但与本文采用的定义等价。

受保护的 单子二阶逻辑**无限词转导** 受限 MSO 公式是对 MSO 公式的一种语法限制。此限制要求所有自由变量和量词都由一阶变量 g 守卫，也就是说，量词只能讨论在 g 之前（即小于 g）的位置。直观而言，受限公式在无限词上的可满足性只取决于截至位置 g 的有限前缀。形式上，给定两个一阶变量 x 和 g，我们令 G(x,g) 表示公式 $x \leq \mathsf{g}$（x 由 g 守卫）；对于集合变量 X，令 G(X,g) 表示公式 $\forall x \in X$,G(x, g)。那么，若 MSO 公式 φ 等于 ψ(g) $\wedge \bigwedge_{\alpha \in Free(\psi)}$ G(α, g)，其中ψ(g) 满足其所有带量词子公式（即形如 $QX \cdot \psi'$ 或 $Qx \cdot \psi'$ 的子公式，其中 $Q \in \{\exists, \forall\}$）均属于以下形式之一，则称该公式由某个变量 g守卫：

$$(1)\ \forall x \cdot \mathsf{G}(x,\mathsf{g}) \rightarrow \zeta \quad (2)\ \exists x \cdot \mathsf{G}(x,\mathsf{g}) \wedge \zeta \quad (3)\ \forall X \cdot \mathsf{G}(X,\mathsf{g}) \rightarrow \zeta \quad (4)\ \exists X \cdot \mathsf{G}(X,\mathsf{g}) \wedge \zeta$$

如果 MSO 公式的形式为 $\exists \mathsf{g} \cdot \varphi$，且 φ 受守卫 g 保护，则称其为受保护的 。我们用 MSO$_g$ 表示受保护的MSO公式集合。为简洁起见，我们可以写作 $\forall x : \mathsf{g} \cdot \zeta$，而不写作 $\forall x \cdot \mathsf{G}(x,$ g) $\rightarrow \zeta$；也可以写作 $\exists x : \mathsf{g} \cdot \zeta$，而不写作 $\exists x \cdot \mathsf{G}(x,\mathsf{g}) \wedge \zeta$（集合变量也类似）。

▶ **例 4.4**。 MSO 转导例 4.2 中定义函数double的所有公式都是受保护的，或者与某个受保护公式平凡等价。例如，公式 first(x) 等价于受保护公式 $\exists \mathsf{g} \cdot\ x \leq \mathsf{g} \wedge \forall y \leq \mathsf{g} \cdot\ \neg S(y,x)$。

序谓词 $x \leq y$ 可由受保护公式 $\exists \mathrm{g}.\ x \leq \mathrm{g} \wedge y \leq \mathrm{g} \wedge y = \mathrm{g}$ 定义。由于 $\neg(x \leq y)$ 等价于 $y \leq x \wedge y \neq x$，我们很容易得到：任意 MSO$_g$- 公式 ϕ 都等价于某个 MSO$_g$- 公式 ψ，其中序谓词仅用于限制量词；具体做法是对一个全局守卫进行存在量词量化，并对用于定义 ϕ 中出现的 $z \leq t$ 形式原子公式（假设其正向出现）的所有局部守卫加以限制。

▶ 注 4.5。单子二阶逻辑$_g$ 公式只涉及前缀，具体来说：若 $\varphi = \exists \mathrm{g}.\psi(\mathrm{g})$ 是一个闭受限公式，且 $w \in \Sigma^\omega$，那么 $w \models \varphi$ 当且仅当存在一个 w 的有限前缀u，使得 $u \models \psi(\ell)$，其中 ℓ 是 u 的最后一个位置。这使我们可以立即得到如下刻画：语言 $L \subseteq \Sigma^\omega$ 是单子二阶逻辑$_g$ 可定义的，当且仅当存在一个正则语言 $F \subseteq \Sigma^*$，使得 $L = F\Sigma^\omega$。

▶ **定义 4.6** (受限 MSO 转导)。 受限的单子二阶逻辑转导 （MSO 转导$_g$）是指其所有公式均受限的单子二阶逻辑转导。

▶ **例 4.7**。 如例 4.4 中所述，例 4.2 中定义 double 的 MSO 转导的所有公式都是受限公式，或与某个受限公式显然等价。

现在我们可以给出确定性正则函数的逻辑刻画：

▶ **定理 4.8** （逻辑刻画）。函数 $f : \Sigma^\omega \rightharpoonup \Gamma^\omega$ 是确定性正则函数，当且仅当它是单子二阶逻辑转导$_g$可定义的。

证明见第 6 节。作为这一结果的一个应用，由于根据定理 3.1，确定性正则函数在复合下（有效地）封闭，我们得到单子二阶逻辑转导$_g$在复合下也（有效地）封闭。对于有限字符串 [21], 、无限字符串 [3] 以及更一般的任何结构 [12], 上的单子二阶逻辑转导，这是一个众所周知的结果；不过，已有证明完全基于逻辑且直接，而我们在这里使用了复杂的基于自动机的论证（移除前瞻）。事实上，单子二阶逻辑转导的复合封闭性是通过公式替换得到的。要复合两个单子二阶逻辑转导$\mathcal{T}_2 \circ \mathcal{T}_1$, ，需将$\mathcal{T}_2$ 中出现的谓词替换为它们在$\mathcal{T}_1$中的定义。这样的直接证明思路不适用于受限片段中的单子二阶逻辑转导$_g$，因为守卫公式在否定下并不封闭。

我们通过讨论 MSO$_g$-转导的一种替代定义来结束本节，该定义记为 MSOT$_g[\leq]$。它不定义输出后继关系，而是要求使用 MSO$_g$ 公式定义输出结构的全序 $\leq$。然而，这允许定义不可计算函数，正如附录中证明的以下命题所述：

▶**命题 4.9**。 存在一个 MSOT$_g[\leq]$，它定义了一个不可计算函数。

在没有受保护限制的情况下，这两种定义都定义了无限词上的正则函数类。

5 具有有限前瞻的双向转导器

我们为确定性双向转换器扩展了有限前瞻。转移还会标记一个有限词的正则语言有限词，称为（有限）前瞻。只有当输入序列的剩余部分存在一个属于 L 的前缀时，才能执行带有前瞻 L 的转移。这样的有限前缀称为 L 的前瞻见证。为确保确定性，如果多个前瞻均成功，则要求存在唯一的最短前瞻见证。转换器执行使见证长度最短的转移。如果所有前瞻均不成功，则计算失败。

▶ 定义 *5.1*（有限前瞻）。带有限前瞻的确定性双向转换器（*2-dT*^{FLA}）是一个元组 $T = (\Sigma, \Gamma, Q, q_0, F, \delta, \lambda)$，其中 $\Sigma, \Gamma, Q, q_0, F, \lambda$ 的定义与不带前瞻的确定性双向转换器相同，δ 是一个转移函数 $Q \times (\Sigma \uplus \{\vdash\}) \times R^*(\Sigma)) \rightharpoonup Q \times \{\triangleright, \triangleleft\}$，其中 $R^*(\Sigma)$ 是所有字母表为 Σ 的有限词正则语言的集合。函数 δ 的定义域必须是有限的。转移 $(q, \sigma, L) \mapsto (q, d)$ 的前瞻为 L。此外，我们要求，如果 $\delta(q, \sigma, L)$ 和 $\delta(q, \sigma, L')$ 均有定义，那么对于所有 $L, L' \in R^*(\Sigma)$，$L \cap L' = \emptyset$，且 $q \in Q$ 和 $\sigma \in \Sigma$。最后，假定前瞻语言由确定性有限自动机表示。

与不带前瞻的模型相比，带有限前瞻的确定性双向转换器的语义保持不变。存在前瞻时，唯一的区别在于转移何时启用：带前瞻 L 的转移只有在输入序列剩余部分的某个前缀属于 L 时才能执行。形式上，在输入 u 上的配置 (q, i) 中，形如 $\delta(q, \sigma, L)$ 的转移（其中 $L \subseteq \Sigma^*$）在 $u[i] = \sigma$ 且存在某个 $i < j$ 使得 $u[i{+}1{:}j] \in L$ 时启用。字 $u[i{+}1{:}j]$ 称为 L 的见证。为确保确定性，当转导器处于配置 (q, i) 时，如果有多个前瞻 L_1、…、L_k 已启用，则触发的转移是见证最短的那个唯一转移（由不相交性要求确保其唯一性）。

移除有限前瞻 我们知道，无限前瞻的表达能力严格强于有限前瞻。自然的问题是：向不带前瞻的确定性双向转换器添加有限前瞻，能够获得多大的表达能力提升？正如引言中已经解释的，任何由这类转换器定义的函数都是（图灵机）可计算的：图灵机可以记住自己在输入中的位置，验证哪个前瞻成功，并从记忆的位置继续计算。双向转换器没有能力记住输入中任意远处的位置。因此，在没有某种前瞻“预言机”的情况下，验证输入剩余部分的某个有限前缀是否是某个前瞻的见证，并返回特定位置，就成为一个需要解决的问题。这个问题并非无限词上的双向转换器所独有；当必须检查有限输入词剩余部分的某个正则性质，随后双向转换器还必须返回检查该性质之前所在的位置时，也会出现同样的问题。在有限词上，可以使用 Hopcroft-Ullman [1] 或改进的树形轮廓构造 [15]来处理这一任务。然而，这些构造依赖于输入词是有限的这一事实。我们证明，使用不同的技术，同样可以在无限词上完成这一任务。

下文将证明，允许有限前瞻并不会提升表达能力。

▶ **定理 5.2** (有限前瞻消除)。 给定一个2-dT^{FLA}，可以有效地构造一个等价的2-dT。

证明概要。证明分为两部分。主要部分是将给定的 2-dT^{FLA} 转换为与之等价且具有有界复制的BSST 。然后，我们利用定理 2.9 得到一个与之等价的 2-dT。对于一个不带前瞻的确定性双向转换器，获得等价SST 的标准方法，是在无限输入已读前缀上模拟该确定性双向转换器的从右到右运行，将其输出存入寄存器，并以适当的方式组合这些寄存器（以及“主”从左到右运行的输出），从而重新生成双向转换器的输出。由于该双向转换器是确定性的，因此对输入的任意前缀，不同从右到右运行的数量都有一个全局上界。 [3, 16, 9] 中介绍的构造都基于这一思路。在

[2], 有限词上的SST 与双向转换器之间的等价性已有证明，但该工作没有给出直接的转换方法。

我们的目标是为具有有限前瞻的确定性双向转换器设计类似的构造。主要难点在于，能够在某个前缀上出现的不同运行数量没有全局上界。原因是，这种转换器在某种意义上是非确定性的：它必须决定哪种前瞻能够成功。看到前瞻见证后，才能事后验证这一选择。

因此，我们扩展了标准构造，将确定性双向转换器转换为 SST，同时将前瞻选择纳入考虑。这种方法用树结构表示可能的运行（类似于非确定性自动机的标准运行树；这里的非确定性来自前瞻选择）。此类树中的一条分支对应一种可能的运行，而节点还包含用于判断前瞻选择何时成功或注定失败的信息。通过共享信息和采用相关的剪枝策略，可以控制树表示的大小。该策略负责移除前瞻选择无法成功的分支，以及前瞻选择已经成功的分支（及其前缀）。将此构造应用于不带前瞻的确定性双向转换器，即可得到标准的转换构造。◀

6 逻辑-转换器对应关系：定理 4.8 的证明

本节概述定理 4.8 的逻辑刻画证明。我们首先证明，任意确定性正则函数都可以由 MSO 转导g定义。该证明是标准的，采用了与有限字的正则函数 [21] 和无限词的正则函数 [3]相同的思路。

▶**引理 6.1**。 如果函数 $f:\ \Sigma^\omega \rightharpoonup \Gamma^\omega$ 是确定性正则的，那么它可以由MSO 转导g定义。

证明。 主要思路是在 MSO 转导g 中定义一个 2-dT的运行。MSO 转导g的每个复制代表 2-dT的一个状态；当且仅当存在一段有限运行，从配置 $(p,\ x)$ 到配置 $(q,\ y)$，且仅在配置 $(p,\ x)$ 和 $(q,\ y)$ 中产生输出符号时，节点 x^p 与节点y^q之间才有一条后继边，其中 $x,\ y$ 是输入位置，$p,\ q$ 是状态。该性质可以用一个 MSOg 公式表达。◀

引理 6.1 的逆命题证明更为复杂。我们首先转向一个带有单子二阶逻辑指令的中间模型，其思想源自 [21], 所谓的跳跃 MSOg-转导器，已证明该模型等价于 2-dT。它是一个有限状态模型，可以 (i) 检验当前位置的单子二阶逻辑g 性质（称为环视），(ii) 检验由 MSO公式定义的安全约束，并且 (iii) 使用二元 MSOg 公式从一个位置跳转到另一个位置。形式上，它具有一个有限状态集（所有状态均为终态），其转移形式为 $p \xrightarrow{\phi_{\mathrm{la}}(x)|w,\phi_{\mathrm{mv}}(x,y)\ \phi_{\mathrm{sf}}(x)} q$，其中 p,q是状态，$\phi_{\mathrm{la}},\phi_{\mathrm{mv}}$ 是 MSOg 公式，ϕ_{sf} 是一个 MSO 公式，而 w 是一个有限字。在具有相同源状态的转移中出现的环视被假定为两两不相交（它们的合取不可满足）。初始配置为 $(q_0, 0)$，其中 q_0 是初始状态。当它处于配置 (q, i) 中、面对一个无限词 $u \in \Sigma^\omega$时，它启用环视 $\phi_{\mathrm{la}}(i)$ 在 u 上成立的转移，并选择见证最短的转移。将 t 称为该转移。仅当存在 j 使得 $\phi_{\mathrm{mv}}(i,j)$ 成立，并且对于所有 $k \geq i$, u[都有 k] $\models \phi_{\mathrm{sf}}(i)$ 时，它才会触发 t（否则计算失败）。随后，它输出 γ，并移动到满足 $\phi_{\mathrm{mv}}(i,j)$ 的某个位置 j。请注意，可能存在多个 j，因此一般而言，同一个输入上可能有多个运行。于是我们作出如下假设，即

其非正式表述如下：对于从初始配置可达的转导器的任意可达配置，总是存在唯一的 j。形式上，对于所有无限配置序列 $(q_0,\ i_0 = 0)(q_1,\ i_1)(q_2,\ i_2)\ldots$，对于所有 $k \geq 0$，对于从配置 $(q_k,\ i_k)$ 触发并转移至 $(q_{k+1},\ i_{k+1})$ 的任意转移 t，如果 $\phi_{\mathrm{mv}}(x,\ y)$ 是 t 的跳跃公式，那么 i_{k+1} 是满足 $\phi_{\mathrm{mv}}(i_k,\ i_{k+1})$ 的唯一位置。与双向转换器一样，配置序列 $(q_0,\ i_0 = 0)(q_1,\ i_1)\ldots$ 是 可接受的，如果 $\lim_{k\to\infty} i_k = \infty$，并且它生成一个无限词。

我们证明，这一模型定义了确定性正则函数：

▶**引理 6.2.** 任意跳跃式单子二阶逻辑$_g$-转导器都定义一个确定性正则函数。

证明思路。 证明分两步进行。首先，证明跳跃式单子二阶逻辑$_g$-转导器等价于行走式单子二阶逻辑$_g$-转导器，也就是说，等价于在相邻位置之间（向前或向后）移动的单子二阶逻辑$_g$-转导器。这一步是标准的（例如，在非守卫设置中，[21] 中就出现过）。接着，证明行走式单子二阶逻辑$_g$-转导器等价于一种扩展的 2-dT ，该扩展具有有限环视和安全约束；随后，通过将环视转换为前瞻，并移除前瞻（基于第 5 节中的技术）和安全约束，证明其等价于 2-dT 。◀

▶ **引理 6.3.** 任意单子二阶逻辑$_g$-转导都等价于一个跳跃式单子二阶逻辑$_g$-转导器。

证明。设 $\mathsf{T} = (k, (\phi^c_\gamma)_{c\in[k],\gamma\in\Gamma}, (\phi^{c,d}_S)_{c,d\in[k]}, \phi^{c_0}\mathsf{fst}(x))$ 是定义 f 的一个 MSOT$_g$。我们构造一个与 T 等价的跳转式 MSO$_g$ 转导器 T′。T′ 的状态集为 $\{0, 1\ldots, k\}$。在状态 0 中，T′ 首先跳转到初始位置，即满足 $\phi^{c_0}\mathsf{fst}(y)$ 的位置 y，并转移到状态 c_0。这通过一条从状态 0 转移到状态 c_0的转移实现，其环视和安全约束分别为平凡条件和 $\top$，移动为 $\phi_{\mathrm{mv}}(x, y)$: $= \mathsf{first}(x) \wedge \phi^{c_0}\mathsf{fst}(y)$。然后，它遵循 T 的后继关系，并使用标签公式确定要输出哪个标签。借助安全约束，T′ 还确保输出图结构是词结构。具体而言，这些约束表示，对于任意可达节点，恰有一个标签，且至多有一个后继。无需检查是否至少有一个后继，因为若没有后继，T′ 的运行就会停止，且输入不被接受；这与 T 的语义一致（在这种情况下，T 也会拒绝该输入）。也无需检查是否存在环，因为若存在环，T′ 将永远无法访问所有输入位置，因此输入会被拒绝，这同样与 T 的语义一致。形式化地，对于所有副本 $c, d \in \{1, \ldots, k\}$ 和输出标签 γ，由于 $\phi^{c,d}_S(x, y)$ 和 $\phi_\gamma(x)$ 是受保护的，它们的形式分别为 $\phi^{c,d}_S(x, y) = \exists \mathsf{g} \cdot \psi_S(x, y, \mathsf{g})$ 以及$\phi_\gamma(x) = \exists \mathsf{g} \cdot \psi_\gamma(x, \mathsf{g})$。然后，我们向 T′ 添加以下转移，从 c 到 d:

$$c \xrightarrow{\phi_{\mathrm{la}}(x):=\exists \mathsf{g}\exists z\leq \mathsf{g}\cdot \psi^{c,d}_S(x,z,\mathsf{g})\wedge\psi^c_\gamma(x,\mathsf{g})\wedge \mathsf{disj}_{c,d,\gamma}(x,\mathsf{g})\mid \gamma, \phi_{\mathrm{mv}}(x,y):=\phi^{c,d}_S(x,y), \phi_{\mathrm{sf}}(x)} d$$

其中，$\mathsf{disj}_{c,d,\gamma}(x, \mathsf{g}) = \forall \mathsf{g}' \leq \mathsf{g} \cdot \bigwedge_{\gamma'\neq\gamma} \neg\psi^c_{\gamma'}(x, \mathsf{g}') \wedge \bigwedge_{d'\neq d} \forall z' \leq \mathsf{g}' \cdot \neg\psi^{c,d'}_S(x, z', \mathsf{g}')$保证环视的不相交性，而 $\phi_{\mathrm{sf}}(x)$等于

$$\begin{array}{ll}
(\bigwedge_{d'\neq d} \forall y \cdot \neg\phi^{c,d'}_S(x, y)) \wedge & \text{no successor of } x \text{ in any copy } d' \neq d \\
(\forall y \forall y' \cdot (\phi^{c,c}_S(x, y) \wedge \phi^{c,c}_S(x, y')) \to y = y') \wedge & \text{at most one successor of } x \text{ in copy } c \\
(\bigwedge_{\gamma'\neq\gamma} \neg\phi^c_{\gamma'}(x) & \text{no other label for } x
\end{array}$$

此处提醒读者，安全约束不必由守卫公式定义，因为它们是有限词的正则性质。不过，环视公式和跳跃公式必须受到保护；上述转移确实满足这一点。

最后请注意，$\mathcal{T}'$ 满足以下要求：对于配置的无限序列（q_0, i_0）…，对于所有 $k \geq 0$，i_{k+1} 都是由跳跃公式确定的 i_k 的唯一后继。确实，如果 $\mathcal{T}'$ 的配置序列是无限的，这意味着所有安全约束都得到满足，而这些约束恰好确保不存在分支。◀

作为引理 6.3 和 6.2 的推论，我们得到定理 4.8 的逆向结论：

▶**推论 6.4**。 任何 MSO 转导$_g$-可定义函数 f 都是确定性正则函数。

7 结论

在本文中，我们证明了确定性正则函数类可由多种计算模型刻画，例如确定性双向转换器、带有限（正则）前瞻的确定性双向转换器、Büchi SST，由受保护的MSO 转导构成的逻辑形式体系，以及顺序函数与映射-复制-反转的有限复合。这些模型之间的转换是有效的。我们还证明了该类在复合下封闭：通过将已知的确定性双向转换器的复合封闭性推广到无限词上，并采用新的证明技术。另一个猜想是，确定性正则函数类等于连续正则函数类（关于康托尔拓扑）。已知它包含连续逐字母有理函数 [22] 以及严格更大的连续有理函数类 [9]。我们相信，本文证明的新刻画将有助于解决这一尚未解决且似乎十分困难的问题。所有这些，再加上确定性正则函数是可计算的，而正则函数并非如此这一事实，都表明了这一类的稳健性。

参考文献

1 Alfred V. Aho、John E. Hopcroft 和 Jeffrey D. Ullman。翻译的一般理论。数学系统理论，3(3):193–221，1969。2 Rajeev Alur 和 Pavol Cerný。流式字符串变换器的表达能力。载于 *IARCS* 软件技术与理论计算机科学基础年会，*FSTTCS 2010*，第 8 卷 *LIPIcs*，第 1–12 页。达格施图尔城堡，2010。3 Rajeev Alur、Emmanuel Filiot 和 Ashutosh Trivedi。无限字符串上的正则变换。载于第 *27* 届 *IEEE* 计算机科学逻辑年会论文集，*LICS 2012*，第 65–74 页。IEEE 计算机学会，2012。4 Rajeev Alur、Adam Freilich 和 Mukund Raghothaman。字符串变换的正则组合子。载于第二十三届 *EACSL* 计算机科学逻辑年会（*CSL*）与第二十九届 *ACM/IEEE* 计算机科学逻辑研讨会（*LICS*）联合会议论文集，第 9 页。ACM，2014。5 Nicolas Baudru 和 Pierre-Alain Reynier。从双向转换器到正则函数表达式。载于语言理论发展国际会议，第 96–108 页。施普林格，2018。6 Mikołaj Bojańczyk 和 Rafał Stefański。无限字母表上的单次使用自动机与变换器。载于第 *47* 届自动机、语言与编程国际学术讨论会（*ICALP 2020*）。达格施图尔城堡-莱布尼茨信息学中心，2020。7 Mikołaj Bojańczyk。多正则函数，2018。网址：https://arxiv.org/abs/1810.08760，doi:10.48550/ARXIV.1810.08760。8 J. R. Büchi。弱二阶算术与有限自动机。数学逻辑与数学基础杂志，6(1–6):66–92，1960。

9 Olivier Carton 和 Gaëtan Douéneau-Tabot。连续有理函数是确定性正则函数。载于第 47 届国际计算机科学数学基础研讨会，MFCS 2022，2022。10 Michal P. Chytil 和 Vojtěch Jákl。双向有限状态转换器的串行复合与字符串上的简单程序。载于第 4 届自动机、语言与编程国际学术讨论会，ICALP 1977，第 135–147 页。施普林格，1977。11 Thomas Colcombet。树的组合定理。载于第 34 届自动机、语言与编程国际学术讨论会，ICALP 2007，2007。12 Bruno Courcelle。单子二阶可定义图转换：综述。《理论计算机科学》，126:53–75，1994。13 Bruno Courcelle 和 Joost Engelfriet。图结构与单子二阶逻辑：一种语言理论方法，第 138 卷。剑桥大学出版社，2012。14 Luc Dartois、Emmanuel Filiot 和 Nathan Lhote。具有综合功能的词转换逻辑。载于第 33 届 ACM/IEEE 计算机科学逻辑研讨会论文集，第 295–304 页。ACM，2018。15 Luc Dartois、Paulin Fournier、Ismaël Jecker 和 Nathan Lhote。论可逆转换器。载于第 44 届自动机、语言与编程国际学术讨论会，ICALP 2017，LIPIcs 第 80 卷，第 113:1–113:12 页。达格施图尔城堡，2017。16 Luc Dartois、Ismaël Jecker 和 Pierre-Alain Reynier。无周期字符串转换器。《国际计算机科学基础杂志》，29(5):801–824，2018。17 Vrunda Dave、Emmanuel Filiot、Shankara Narayanan Krishna 和 Nathan Lhote。无限词上可计算正则函数的综合。载于第 31 届并发理论国际会议（CONCUR 2020）。达格施图尔城堡-莱布尼茨信息学中心，2020。18 Vrunda Dave、Emmanuel Filiot、Shankara Narayanan Krishna 和 Nathan Lhote。无限词上可计算正则函数的综合。《计算机科学逻辑方法》，18(2)，2022。19 Vrunda Dave、Paul Gastin 和 Shankara Narayanan Krishna。正则变换的正则转换器表达式。载于第 33 届 ACM/IEEE 计算机科学逻辑研讨会论文集，第 315–324 页。ACM，2018。20 C. C. Elgot。有限自动机设计及相关算术的判定问题。载于《美国数学会汇刊》，98(1):21–51，1961。21 Joost Engelfriet 和 Hendrik Jan Hoogeboom。单子二阶逻辑可定义的字符串转换与双向有限状态转换器。《ACM计算逻辑汇刊》（TOCL），2(2):216–254，2001。22 Emmanuel Filiot 和 Sarah Winter。从无限词上的有理规格综合可计算函数。载于第 41 届 IARCS 软件技术与理论计算机科学基础年会，FSTTCS 2021，2021 年 12 月 15–17 日，线上会议。达格施图尔城堡-莱布尼茨信息学中心，2021。23 Erich Grädel。论守卫的约束能力。《符号逻辑杂志》，64(4):1719–1742，1999。24 Eitan M Gurari。确定性双向顺序转换器的等价性问题是可判定的。《SIAM计算杂志》，11(3):448–452，1982。25 Dominique Perrin 和 Jean-Éric Pin。无限词：自动机、半群、逻辑与博弈。学术出版社，2004。26 J. C. Shepherdson。将双向自动机归约为单向自动机。《IBM研究与开发杂志》，3(2):198–200，1959。 doi:10.1147/rd.32.0198。27 Imre Simon。有限高度的因子分解森林。《理论计算机科学》，72(1):65–94，1990。doi:10.1016/0304-3975(90)90047-L。28 Wolfgang Thomas。语言、自动机与逻辑。载于《形式语言手册》，第 389–455 页。施普林格，1997。29 Boris Avraamovich Trakhtenbrot。有限自动机与单子谓词逻辑（俄文）。《苏联科学院学报》，140:326–329，1961。

A 分解：定理 3.6 的证明

本节将证明，一个函数是确定性正则函数，当且仅当它可以（有效地）写成顺序函数与映射-复制-反转函数的复合。反向蕴含显然成立，因为确定性正则函数在复合下封闭（定理 3.1），而顺序函数和映射-复制-反转函数都是确定性正则函数。接下来要说明如何利用这些基本函数分解一个确定性正则函数，这正是本附录 A 的目的。

A.1 在复合下封闭的函数类

我们首先引入一个函数类 $\mathcal{C}$，它在复合下封闭，并且既可将有限词映射为有限词，也可将无限词映射为无限词（即类型为 $(\Sigma^* \to \Gamma^*) \cup (\Sigma^\omega \to \Gamma^\omega)$）。我们的目标是证明，任意确定性正则函数都可以表示为函数类 $\mathcal{C}$ 中的某个函数在无限词上的限制。不过，在证明中，有时需要将这些函数应用于有限词或无限词。事实上，我们将构造一种“映射”操作，它作用于给定字的有限或无限因子，而且在运行时无法“判定”当前因子是有限还是无限。

回想一下，定义 3.4 中介绍了无限词上的单向变换器和顺序函数。我们将这一概念推广，以便能够使用同一台机器同时处理有限词和无限词。

▶ **定义 A.1**（有限词和无限词的顺序函数）。有限词和无限词上的确定性单向变换器是一个元组 $(\Sigma, \Gamma, Q, q_0, F_*, F_\omega, \delta, \lambda)$，满足：

- $(\Sigma, \Gamma, Q, q_0, F_\omega, \delta, \lambda)$ 是一个无限词上的单向确定性变换器；$(\Sigma, \Gamma, Q, q_0, F_*, \delta, \lambda)$ 是一个有限词上的单向确定性变换器，也就是说，$F_* : Q \to \Gamma^*$ 是一个最终输出函数。有限输入在有限词上的输出定义为运行过程中产生的输出与 $F_*(q)$ 的串接（若其有定义；否则输出未定义），其中 q 是该次运行的最后一个状态。

这样的机器描述了一个类型为 $(\Sigma^* \to \Gamma^*) \cup (\Sigma^\omega \to \Gamma^\omega)$ 的函数。在有限词和无限词上由单向确定性转导器计算的（部分）函数类称为有限词和无限词的顺序函数。

▶ 注 A.2. 必须使用定义同一个函数的转导器来定义有限词和无限词上的函数。

▶ **定义 A.3**。类型为 $(\Sigma^* \to \Gamma^*) \cup (\Sigma^\omega \to \Gamma^\omega)$ 的函数类 $\mathcal{C}$ 是以下函数在复合下的闭包：

- 有限词和无限词上的顺序函数；对于任意新符号 $| \notin \Sigma$，函数映射-复制-反转 $: ((\Sigma \uplus \{|\})^* \to (\Sigma \uplus \{|\})^*) \cup ((\Sigma \uplus \{|\})^\omega \to (\Sigma \uplus \{|\})^\omega)$ 定义为 $u_i \in \Sigma^* : u_1|\cdots|u_n \mapsto u_1|\widetilde{u_1}|\cdots|u_n|\widetilde{u_n}$ 当输入为有限时；$u_1|\cdots|u_i|\cdots \mapsto u_1|\widetilde{u_1}|\cdots|u_i|\widetilde{u_i}|\cdots$ 如果无限输入中包含无限多个 $|$；$u_1|\cdots|u_n|u \mapsto u_1|\widetilde{u_1}|\cdots|u_n|\widetilde{u_n}|u$ 如果 $u \in \Sigma^\omega$

▶ 备注 A.4. 如果上下文不明确，我们会说映射-复制-反转“有分隔符”，意指使用符号 $|$ 来分隔各因子。

现在我们来描述这个类 $\mathcal{C}$ 的一些性质。

▶ **例 A.5.**（全）函数 map-copy $: (\Sigma \uplus \{|\})^\infty \to (\Sigma \uplus \{|\})^\infty$ 的定义方式与定义 A.3 中的 map-copy-reverse 类似，只是将每个复制镜像因子

$\widetilde{u_i}$ 替换为 u_i 的简单复制。函数 map-copy 属于 $\mathcal{C}$。事实上，我们将 map-copy-reverse 应用两次，得到一个形如 $u_1|\widetilde{u_1}|\widetilde{u_1}|u_1|u_2\cdots$ 的字。使用顺序函数很容易移除无用部分。

▶ 注 A.6. 也可以定义函数 map-reverse：它会反转每个因子，但不会在反转前先复制。不过，该函数无法定义在含有有限个 $|$ 符号的无限词上（甚至无法将其扩展为定义域为全定义域的连续函数）。

最后，我们证明类 $\mathcal{C}$ 在一种“映射”操作下封闭，该操作将给定函数应用于由特定符号分隔的字的各个因子。我们在引理 A.7 中给出一个更精确的表述，以便将不同的函数应用于前几个因子。

▶ **引理 A.7.** 设 $f_1, f_2, \ldots, f_n \colon \Sigma^\infty \to \Gamma^\infty \in \mathcal{C}$，且 $|$ 是一个新符号。可以有效地在 $\mathcal{C}$ 中构造一个函数 $[f_1|f_2|\cdots|f_n|] : (\Sigma \uplus \{|\})^\infty \to (\Gamma \uplus \{|\})^\infty$，使得：

$$[f_1|f_2|\cdots|f_n|](u_1|u_2\cdots) = f_1(u_1)|f_2(u_2)|\cdots|f_n(u_n)|f_n(u_{n+1})|f_n(u_{n+2})\cdots$$

每当 $u_1 \in$ 定义域(f_1) $u_2 \in$ 定义域$(f_2)\ldots$.

▶ 注 A.8. 引理 A.7 的表述中有若干元素未明确说明：若输入中有 $k \le n$ 个因子，则 $[f_1|\cdots|f_n|]$ 仅适用于 $f_1, \ldots, f_k$；最后一个因子（若存在）可以是无限的；若输入是无限的，则必须有 $f_1(u_1)|f_2(u_2)|f_3(u_3)\cdots \in (\Gamma \uplus \{|\})^\omega$ 才能使输出有定义。

证明。我们只处理 $n = 1$ 的情形，它对应于经典的“映射”操作。其他情形也可以用类似的方法处理，即使用顺序函数删除 n 最前面的若干部分中的特定标记。令 $f := f_1$。我们对 f 的构造进行归纳来证明该结论。若 f 是顺序函数，则构造一个顺序函数 $[f|]$，由一个与 f 所用的类似的单向变换器描述；但若读到 $|$，则输出该变换器在当前状态下的（有限）最终输出，并返回初始状态继续计算。若 $f = g \circ h$，则根据归纳假设，结论显然成立。若 f 是分隔符为 # 的映射 - 复制 - 反转（必然有 # $\neq |$），我们先应用一个顺序函数，将每个 $|$ 转换为 $|\#$。然后在整个输入上应用分隔符为 # 的映射 - 复制 - 反转。最后应用一个顺序函数，每次遇到 $||$ 因子时将其删除，并在这种情况下将下一个 # 替换为 $|$。◀

A.2 前向 Ramseyan 分割

现在，我们展示如何将确定性正则函数分解为关于$\mathcal{C}$的函数。继对有限词上的正则 [7] 或多正则 [6] 函数进行分解的结果之后，我们的首要目标是利用一种类似于 Simon 因子分解森林 [27]的结构来分割输入。事实上，双向转换器在这类分割上的运行方式很简单。我们在定义 A.9 中给出一种弱化版的因子分解森林，称为前向 *Ramsey* 分裂，由 Colcombet 在 [11] 中提出。我们将在定理 A.12 中回顾 Colcombet 的主要结果：这类分割可以由顺序函数计算。

若 Σ 是一个字母表，我们考虑无限多个新符号 $|_1$，$|_2$，$\cdots \notin \Sigma$。对于所有 $k \ge 0$，我们定义字母表 $\Delta_k := \{|_1, \ldots\ldots, |_k\}$。

▶ **定义 A.9**（前向 Ramseyan 划分 [11]）。令 $\mu \colon \Sigma^* \to M$ 为到有限幺半群的一个态射，且 $k \ge 0$。令 $u \in \Sigma^\infty$。如果一个字 $\mathfrak{s} \in (\Sigma \uplus \Delta_k)^\infty$ 是关于 u、高度为 $k \ge 0$ 的前向 Ramseyanμ-划分，则称其可按以下方式通过归纳法定义：

- 要么 $k=0$ 且 $\mathfrak{s}=u=a\in\Sigma$，要么 $\mathfrak{s}=u=\varepsilon$；或者 $\mathfrak{s}=\mathfrak{s}_1|_k\cdots|_k\mathfrak{s}_n$，其中 $u=u_1\cdots u_n$，
- 对于所有 $1\le i\le n$，$\mathfrak{s}_i$ 都是 μ、高度为 $k-1$ 的$u_i\in\Sigma^\infty\setminus\{\varepsilon\}$的划分，并且对于所有 $1\le i,$
- $j<n$ $\mu(u_i)\ \mu(u_j)\ =\mu(u_i)$（注意 $u_i,\ u_j\in\Sigma^+$）；或者 $\mathfrak{s}=\mathfrak{s}_1|_k\cdots|_k\mathfrak{s}_i|_k\cdots$，其中
- $u=u_1\cdots u_i\cdots\in\Sigma^\omega$，对于所有 $1\le i,\ \mathfrak{s}_i$ 都是 μ、高度为$k-1$ 的 $u_i\in\Sigma^+$的划分，并且对于所有 $1\le i,\ j,\ \mu(u_i)\ \mu(u_j)\ =\mu(u_i)$。

▶ 注 A.10。特别地，条件 $\mu(u_i)\mu(u_j)=\mu(u_i)$ 意味着 $\mu(u_i)$ 是幂等元。此外，这意味着 $\mu(u_i)$ 在 Greene 关系的意义下是 $\mathcal{L}$-等价的（例如参见 [11]）。在 Simon 的常规因子分解森林（也称为 Ramseyμ-分裂）中，条件更强：$\mu(u_i)=\mu(u_i)^2=\mu(u_j)$（即幂等元相等）。

▶ 注 A.11。在第二种情况下，对值 $\mu(u_n)$ 没有任何假设，即使$u_n\in A^+$。特别是，如果 $n=2$，则对 $\mu(u_2)$ 没有任何假设。

从现在起，态射 μ 将始终被隐式固定，我们用 $\mathrm{FRS}_k(u)$ 表示字 $u\in\Sigma^\infty$ 上高度至多为 $k\ge0$ 的 μ-分裂集合。

▶ **定理 A.12** ([11, 定理 1])。设 $\mu:\Sigma^*\to M$ 是到某个有限幺半群的一个态射。存在一个界 N 和一个顺序函数 $\mathsf{split}:\ \Sigma^\omega\to(\Sigma\cup\Delta_k)^\omega$，使得对所有$u\in\Sigma^\omega$，$\mathsf{split}(u)\in\mathrm{FRS}_N(u)$。该构造是有效的。

A.3 双向转导器的转移幺半群

本节旨在回顾双向转导器的著名概念——转移态射 μ（例如参见 [19]），并将其与前向 Ramseyan μ-分割联系起来。在附录 A 的其余部分中，我们考虑一个固定的确定性双向转换器$\mathcal{T}=(\Sigma,\Gamma,Q,q_0,F,\delta,\lambda)$。令 $\overleftarrow{Q}:=\{\overleftarrow{q}:\ q\in Q\}$ 和 $\overrightarrow{Q}:=\{\overrightarrow{q}:\ q\in Q\}$ 为 Q 的两个不相交副本。为简化记号，我们假设不失一般性地有 $\vdash\in\Sigma$。

回想一下，转移函数 δ的类型为 $Q\times\Sigma\rightharpoonup Q\times\{\triangleleft,\triangleright\}$。我们将转移函数扩展到 $\delta:\ \overrightarrow{Q}\cup\overleftarrow{Q}\times\Sigma^*\rightharpoonup\overrightarrow{Q}\cup\overleftarrow{Q}$，并类似地扩展 $\overrightarrow{Q}\cup\overleftarrow{Q}\times\Sigma^*\rightharpoonup\Gamma^*$。直观地说，$\delta(\overleftarrow{q},u)=\overrightarrow{p}$且 $\lambda(\overleftarrow{q},u)=v$表示：从 u 最右侧位置出发，在 u上的最长运行最终会在状态 p下“从右侧”离开 u，且其输出的拼接为 v。我们假设 $\delta(_,\varepsilon)$ 是恒等函数，并且 $\lambda(_,\varepsilon)=\varepsilon$。最后，我们还通过定义 $\delta(\overrightarrow{q},u):=\overleftarrow{p}$（若最长运行“从左侧”离开 u），以及 $\delta(\overrightarrow{q},u):=\omega$（若该运行是无限的且会访问任意大的位置），将 δ 扩展到无限词 $u\in\Sigma^\omega$。注意，$\delta(\overleftarrow{q},u)$ 没有意义。相应地扩展函数 λ。

以下结果是关于双向转导器的一个众所周知的结论（针对有限词，不过由于这里讨论的是有限词上的转移，因此这不会造成任何影响）。

▶**命题 A.13**（转移态射，参见例如 [19]）。可以有效地构造一个有限幺半群 M和一个态射 $\mu:\Sigma^*\to M$，称为 T 的转移态射，使得对于所有 $u\in\Sigma^*$ 和 $q\in\overrightarrow{Q}\cup\overleftarrow{Q}$，$\delta$（$q$，$u$）仅取决于$q$和 $\mu(u)$。

从现在起，假设 $\mu:\ \Sigma^*\to M$ 是$\mathcal{T}$的转移态射。现在，我们证明，对于所构造的一个前向 Ramseyan μ-分割，$\mathcal{T}$的运行可以用一种相当简单的方式描述。此结果受到有限词上的因子分解森林相关结果的启发（参见例如 [7, 引理 6.8]），但由于我们的 μ-分割弱于因子分解森林（参见注 A.10），当前结果也较弱。引理 A.14 中提到的运行 ρ_1，ρ_2，..... 见图 2。

▶ **引理 A.14** (运行与前向 Ramseyan 分割)。设 $k\ge1$，$u=u_1u_2\cdots\in\Sigma^\infty$ 和 $\mathfrak{s}=\mathfrak{s}_1|_k\mathfrak{s}_2\cdots\in\mathrm{FRS}_k(u)$ 是遵循因子分解 $u_1,u_2,\ldots$ 的一个 μ-分割。假设

$\delta(\overrightarrow{q}, u_1) = \overrightarrow{p}$. 设 ρ 是 T 沿 u 从配置 $(q,1)$ 开始的最长（有限或无限）运行。于是 ρ 可以分解为 $\rho = \rho_1 \to \rho_2 \to \cdots$ 的形式，其中：
1. 对于所有 $i \geq 1$ 使得 u_i 有定义，ρ_i 从 ρ 的第一个配置开始，并访问 u_i；2. 对于所有 $i \geq 2$，ρ_i 以如下形式的配置开始 $(p, _)$（也就是说，它从 $\overrightarrow{p}$ 开始）；3. 对于所有 $i \geq 2$ 使得 u_{i+1} 有定义，ρ_i 只访问 u_i 的位置，并且 u_{i-1}（它不能返回到 u_{i-2}）。

证明。我们有 $\delta(\overrightarrow{q}, u_1) = \overrightarrow{p}$。对于所有使 u_i 有定义的 $i \geq 2$，必定存在有限词 u_1、……、$u_{i-1} \in \Sigma^+$，以及 $\mu(u_1)\mu(u_2)\cdots\mu(u_{i-1}) = \mu(u_1)$，因此 $\delta(\overrightarrow{q}, u_1u_2\cdots u_{i-1}) = \delta(\overrightarrow{q}, u_1) = \overrightarrow{p}$。这意味着因子 u_i 被 ρ 访问，而且这次访问从状态 p 开始。通过相应地定义 ρ_i，即可得到第 1 项和第 2 项。对于第 3 项，设 $i \geq 3$（对于 $i = 2$，结论显然）满足 u_{i+1} 有定义，我们证明 ρ_i 只访问 u_i 和 u_{i-1}。首先注意，根据 ρ_{i+1} 的构造，这次运行不会访问 u_{i+1}。考虑 u_{i-1} 在 ρ_{i-1} 中第一次访问的位置时最后经过的状态 r。由于 $\mu(u_{i-1}u_i) = \mu(u_{i-1})$，我们有 $\delta(\overrightarrow{r}, u_{i-1}u_i) = \delta(\overrightarrow{r}, u_{i-1}) = \overrightarrow{p}$（最后一个等式由第 2 项推出，因为它描述了 ρ_i 的开头）。这意味着，从 u_{i-1} 的第一个位置上的 r 开始时，T 将执行完 ρ_{i-1} 的末尾，接着执行 ρ_i，最终“向右”离开 $u_{i-1}u_i$。因此，运行 ρ_i 会一直停留在 $u_{i-1}u_i$ 中，直到它以状态 p 转到 u_{i+1}（按照构造，这正是 ρ_{i+1} 的开头）。◀

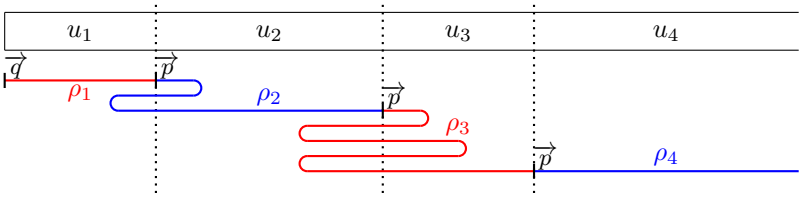


图 2 引理 A.14 中运行的形状（即对于所有 i, j，当 $\mu(u_i)\mu(u_j) = \mu(u_i)$ 时）。

▶注 A.15。如果未定义 u_{n+1}（即分割的形状为 $\mathfrak{s}_1|_k \cdots |_k \mathfrak{s}_n$），且有 $u_n \in \Sigma^\infty$，那么没有理由认为 ρ_n 只应访问 u_{n-1} 和 u_n。事实上，我们不知道 $\mu(u_n)$ 的信息（即使它有定义），因此无法描述 $\mathcal{T}$ 在 u_n 中的行为。consequently，最后一个 ρ_n（如果存在）无法受到控制，并且引理 A.14 并不意味着：如果 $u \in A^\omega$，则 $\delta(\overrightarrow{q}, u) = \omega$；如果 $u \in A^+$，则 $\delta(\overrightarrow{q}, u) = \overrightarrow{p}$。

▶注 A.16。注意，在图 2 中，运行 ρ_2 访问 u_2 的第一个位置 3 次，而 ρ_3 访问 u_3 的第一个位置 5 次。对于 Simon 的常规因子分解森林（即 $\mu(u_1) = \mu(u_2) = \cdots$），由于所有因子都具有相同的幂等像，这种差异（$3 \neq 5$）不可能出现。

回顾一下，定理 3.3 表明，任意定义在有限词上的正则函数，都可以表示为某个 $\mathcal{C}$ 函数在有限词上的限制。利用这一结果，我们在引理 A.17 中断言，从有限词右侧开始的 $\mathcal{T}$ 运行可以由某个 $\mathcal{C}$ 函数模拟。我们甚至将输入词 u 替换为一个 μ-分割的 $\mathfrak{s} \in \mathrm{FRS}_k(u)$（这里并非必要，但这样更容易在证明的其余部分处理中更精确的 $\mathfrak{s}$）。

▶引理 *A.17*。对于所有 $k \geq 0$，都可以有效地构造一个函数 $\overleftarrow{run}_k : (\Sigma \cup \Delta_k \cup \overleftarrow{Q})^* \to (\Sigma \cup \Delta_k \cup \Gamma \cup \overleftarrow{Q})^* \in \mathcal{C}$，使得对于所有 $u \in \Sigma^+$，$\mathfrak{s} \in FRS_k(u)$ 和 $q \in Q$：如果 $\delta(\overleftarrow{q}, u) = \overrightarrow{p}$ 且 $\lambda(u, \overleftarrow{q}) = v$，那么 $\overleftarrow{run}_k(\mathfrak{s}\overleftarrow{q}) = v\mathfrak{s}\overrightarrow{p}$；如果 $\delta(\overleftarrow{q}, u) = \overrightarrow{p}$ 且 $\lambda(\overleftarrow{q}, u) = v$，那么 $\overleftarrow{run}_k(\overleftarrow{q}) = v\overleftarrow{p}\mathfrak{s}$。

证明。这样的函数（从有限词映射到有限词）可以由一个忽略 Δ_k 中符号的双向转换器计算。结论由定理 3.3 得出。◀

A.4 主要归纳证明

证明的核心是对 $k \geq 0$ 使用归纳法证明引理 A.18。它大致相当于引理 A.17 的一个类比，但针对的是从有限词或无限词左侧开始的运行。由于字可以“向右”无限延伸，却不能“向左”无限延伸，上述两个引理之间存在根本的不对称性。

▶ **引理 A.18**。对于所有 $k \geq 0$，都可以有效地构造一个函数 $\overrightarrow{\text{运行}}_k \in \mathcal{C}$，使得对于所有 $q \in Q$, $u \in \Sigma^\infty$ 和 $\mathfrak{s} \in \text{FRS}_k(u)$：

- 若 $\delta(\overrightarrow{q}, u) = \overleftarrow{q}$ 且 $\lambda(\overrightarrow{q}, u) = v \in \Gamma^*$，则 $\overrightarrow{\text{运行}}_k(\overrightarrow{q}\mathfrak{s}) = v\overleftarrow{q}\mathfrak{s}$；若 δ
- $(\overrightarrow{q}, u) = \omega$ 且 $\lambda(\overrightarrow{q}, u) = v \in \Gamma^\omega$，则 $\overrightarrow{\text{运行}}_k(\overrightarrow{q}\mathfrak{s}) = v$；若 $u \in \Sigma^+$,
- $\delta(\overrightarrow{q}, u) = \overrightarrow{q}$ 且 $\lambda(\overrightarrow{q}u) = v \in \Gamma^*$，则 $\overrightarrow{\text{运行}}_k(\overrightarrow{q}\mathfrak{s}) = v\mathfrak{s}\overrightarrow{q}$。

▶ 注 A.19. 如果我们不属于三种情况中的任何一种（例如 $\delta(\overrightarrow{q}, u) = \omega$ 但是 $\lambda(\overrightarrow{q}u) \in \Gamma^*$），那么 $\overrightarrow{\text{运行}}_k$ 的值无关紧要（它甚至可以未定义）。

现在，我们来说明为什么定理 3.6 可以由这个结果直接推出。

定理 3.6 的证明。根据定理 A.12，可以有效地构造一个函数split：$\Sigma^\omega \to (\Sigma \cup \Delta_k)^\omega$ 使得对所有 $u \in \Sigma^\omega$，$\text{split}(u) \in \text{FRS}_N(u)$。我们将split 与一个在输入开头添加字母 $\overrightarrow{q_0}$ 的函数复合，然后再与 $\overrightarrow{\text{run}}_N$复合。只要输入存在一个输出为无限词的接受运行，这个复合就会产生与$\mathcal{T}$相同的输出。然而，它也可能在不属于$\mathcal{T}$定义域的输入上有定义。但请注意，根据命题 2.11，该定义域 L 是一个 Büchi 确定性语言。因此，只需将我们的构造与恒等函数在 L 上的限制进行前复合（这是一个定义在无限词上的顺序函数）。◀

附录 A.4 的其余部分用于证明引理 A.18。我们通过对 $k \geq 0$ 进行归纳法来证明这一结果。对于 $k = 0$，必然是 $u = \mathfrak{s} = a$ 或 $u = \mathfrak{s} = \varepsilon$，因此只需构造一个将函数 δ 和 λ硬编码其中的顺序函数，便可得到该结果。

现在，假设函数 $\overrightarrow{\text{run}}_k$ 已针对某个 $k \geq 0$ 在 $\mathcal{C}$ 中构造完成。我们来说明如何构造函数 $\overrightarrow{\text{run}}_{k+1}$，使其在 $\mathcal{C}$ 中运行。令 $u \in A^\infty$ 和 $\mathfrak{s} \in \text{FRS}_{k+1}(u)$。不失一般性（先应用一个将 $|_{k+1}$ 转换为 $|$的顺序函数），我们假设 $\overrightarrow{q}\mathfrak{s}$ 具有如下形式：$\overrightarrow{q}\mathfrak{s}_1|\mathfrak{s}_2|\cdots$，其中 $\mathfrak{s}_i \in \text{FRS}_k(u_i)$、$u_i \in A^\infty$ 和 $u = u_1u_2\cdots$。

1. 我们首先应用 $[\overrightarrow{\text{run}}_k| \,\text{id}\,|]$ 函数（其中 id 表示恒等函数），该函数由引理 A.7 构造，输出如下（其中 $v_1 := \lambda(\overrightarrow{q}u_1)$）：

- 若 $\delta(\overrightarrow{q}, u_1) = \omega$（必然 $u = u_1 \in \Sigma^\omega$）且 $v \in \Gamma^\omega$，则 v_1（无事可做）；若 $\delta(\overrightarrow{q}, u_1)$
- $= \overleftarrow{p_1}$，则 $v_1\overleftarrow{p_1}\mathfrak{s}_1|\mathfrak{s}_2\cdots$（无事可做）；若 $\delta(\overrightarrow{q}, u_1) = \overrightarrow{p}$（必然 $u_1 \in \Sigma^+$），则 $v_1\mathfrak{s}_1\overrightarrow{p}|\mathfrak{s}_2\cdot$
- $\cdots$。

2. 接着，我们应用一个顺序函数，检查是否出现 $\overrightarrow{p}|$，移除 $\overrightarrow{p}$，并将每个字母 $|$ 替换为 $|\overrightarrow{p}$，由此得到：

- 若 $\delta(\overrightarrow{q}, u_1) = \omega$ 且 $v_1 \in \Gamma^\omega$，则 v（无事可做）；若 $\delta(\overrightarrow{q}, u_1) = \overleftarrow{p_1}$，则
- $v_1\overleftarrow{p_1}\mathfrak{s}_1|\mathfrak{s}_2\cdots$（无事可做）；若 $\delta(\overrightarrow{q}, u_1) = \overrightarrow{p}$，则 $v_1\mathfrak{s}_1|\overrightarrow{p}\mathfrak{s}_2|\overrightarrow{p}\mathfrak{s}_3|\overrightarrow{p}\cdots$。在这种
- 情况下，按照引理 A.14 的记号，v_1 是沿着 ρ_1产生的输出，并且对于所有 $i \geq 2$ ρ_i，都以状态 p开始。现在，我们的思路是利用一个“映射”并行模拟（这正是本证明的关键思路）所有针对 $i \geq 2$ 的运行 ρ_i，因为它们都从状态 p开始。

3. 因此，我们想要将 $\overrightarrow{\text{运行}}_k(\overrightarrow{p}\mathfrak{s}_i)$ 应用于所有适当的因子。为此，我们先在每个 $\#$ 符号前写入一个 $\overrightarrow{p}$（此操作可以让我们保留不含 $\overrightarrow{p}$ 的因子不变）。我们构造 $\overrightarrow{\text{try-run}} := [\text{id}\#\overrightarrow{\text{运行}}_k\#]$（回想一下，如果它没有遇到 $\#$，这个

函数就只会表现为恒等函数 id）。然后，我们将函数 $[\overrightarrow{\text{try-run}}|]$ 应用于整个输入：

- 如果没有字母 $\overrightarrow{p}$，则该字不会发生变化（前面的几种情况）；如果输入为
- $v_1\mathfrak{s}_1|\overrightarrow{p}\mathfrak{s}_2|\overrightarrow{p}\mathfrak{s}_3|\overrightarrow{p}\cdots$，则得到一个形状为 $v_1\mathfrak{s}_1|w_2|w_3|\cdots$ 的字。下面根据引理 A.14 的运行 ρ_i，正式描述 w_i for $i \geq 2$：如果定义了 u_{i+1}，则会出现两种情况：
  - 要么 ρ_i 从未访问 u_{i-1} 的最后一个位置，这意味着它“直接从右侧”离开 u_i，且根据引理 A.14，处于状态 p。此时我们有 $w_i = v_i\mathfrak{s}_i\overrightarrow{p}$，其中 v_i 是沿整个 ρ_i 生成的输出；
  - 要么 ρ_i 访问了 u_{i-1} 的最后一个位置，且首次访问该位置时处于状态 $p_i \in Q$，于是 $w_i = v_i\overleftarrow{p_i}\mathfrak{s}_i$，其中 $v_i \in \Gamma^*$ 是 $\mathcal{T}$ 从起始处到首次访问该位置期间沿 ρ_i 生成的输出；否则，如果 $u_n \in \Sigma^\infty$ 是最后一个因子，则可能出现三种情况，分别产生 $w_n = v_n\mathfrak{s}_n\overrightarrow{r}$（类似于上述第一种情况）、$w_n = v_n\overleftarrow{p_n}\mathfrak{s}_i$（类似于第二种情况）或 $w_n = v_n \in \Gamma^\omega$（$\rho_n$ 访问 $u_n \in \Sigma^\omega$ 中任意大的位置）

4. 然后，我们应用一个顺序函数，将每个子词 $\overrightarrow{s}|$ 替换为 $|$（以便移除因子中的 $\overrightarrow{p}|$，因为相应的 ρ_i 已经被完全模拟）。
5. 现在，我们的目标是模拟运行的下一部分 ρ_i，即处理形如 $w_i = v_i\overleftarrow{p_i}\mathfrak{s}_i$的因子，这些运行尚未完成。在这种情况下，我们需要计算 $\overleftarrow{\text{运行}}_k(\mathfrak{s}_{i-1}\overleftarrow{p_i})$。我们首先构造一个函数：如果下一个因子包含某个 $\mathfrak{s}_{i-1}$，则复制前一个因子中的 $\overleftarrow{s}$。

▷ 命题 A.20。设 # 为一个新符号。我们可以有效地构造一个 $\mathcal{C}$ 中的函数，该函数以目前构造出的字为输入，将形如 $v_i\overleftarrow{p_i}\mathfrak{s}_i$ 的每个因子 w_i（其中 $i \geq 2$）转换为 $v_ip_i\#\mathfrak{s}_{i-1}\overleftarrow{p_i}\#\mathfrak{s}_i$，并保持其他因子不变。

证明。由于所有因子的形状均为 $v_i\overleftarrow{p_i}\mathfrak{s}_i$、$v_i\mathfrak{s}_i$ 或 $v_n\mathfrak{s}_n\overrightarrow{r}$ 或 $v_n \in \Gamma^\omega$，我们可以先应用一个顺序函数，在每个 $\mathfrak{s}_i$之前添加一个 # 符号。然后，我们应用带有分隔符 # 的映射复制，得到形如 $\#\mathfrak{s}_{i-1}|v_i(\overleftarrow{p_i}?)\#\mathfrak{s}_{i-1}|v_i(\overleftarrow{p_i}?)\#$ 的因子。接着，我们应用一个顺序函数：先使用前一个因子的第一个 $\mathfrak{s}_{i-1}$，然后输出 v_i；如果存在 $\overleftarrow{p_i}$，则输出 $p_i\#\mathfrak{s}_{i_1}$；忽略下一个 v_i，最后以 $\overleftarrow{p_i}$结束。容易看出，当 v_i 或 $\mathfrak{s}_{i-1}$ 为无限时，该函数的行为符合预期。◀

最后，我们将命题 A.20 中描述的函数应用于整个输入。

6. 剩下的就是在适当的因子上应用 $\overleftarrow{\text{运行}}_k(\mathfrak{s}_{i-1}\overleftarrow{p_i})$。为此，我们构造 $\overleftarrow{\text{try-run}} := [\text{恒等函数}\ \#\ \overleftarrow{\text{运行}}_k\ \#\ \text{恒等函数}\ \#]$（回想一下，如果它不遇到 #，此函数就只会表现为恒等函数 id），然后我们将函数 $[\overleftarrow{\text{try-run}}|]$ 应用于整个字：
   - 不含 # 的因子不会被修改（也就是说，之前没有 $\overleftarrow{s} \in \overleftarrow{Q}$ 的因子，以及可能含有一个 $\overleftarrow{s}$ 的第一个因子）；否则，每个形如 $v_ip_i\#\mathfrak{s}_{i-1}\overleftarrow{p_i}\#\mathfrak{s}_i$ for $i \geq 2$ 的因子现在都会变成：
   - 要么是 $v_ip_i\#v'_i\mathfrak{s}_{i-1}\overrightarrow{p'_i}\#\mathfrak{s}_i$。这意味着，在 u_{i-1} 中停留之后，ρ_i 会回到 u_i，并处于状态 p'_i；而 $\mathcal{T}$ 在 ρ_i 中这段路径上的输出为 v'_i。在这种情况下，我们希望将 v_i 和 v'_i 拼接起来；要么是 $v_np_n\#v'_n\overleftarrow{p'_n}\mathfrak{s}_{n-1}\#\mathfrak{s}_n$。在这种情况下，这意味着 ρ_n 将访问 u_{n-2}。根据引理 A.14，只有当 $u_n \in \Sigma^\omega$ 是最后一个因子时，这种情况才可能发生。在这种情况下，μ-分割无法帮助我们控制 ρ_n，但这种非常特殊的情况在整个过程中只会发生一次，并将在第 10 项中处理。

7. 现在，让我们去掉# 以及无用的因子副本。

▷ 命题 A.21. 可以在$\mathcal{C}$中有效地构造一个定义在字母表$\Sigma \cup \Gamma \cup Q \cup \overleftarrow{Q} \cup \overrightarrow{Q} \cup \{\#\}$上的函数 normalize，其行为如下：

- 如果其输入不包含 #，它的行为就是恒等函数；如果其输入的形式
- 为 $v_i p_i \# v'_i \mathfrak{s}_{i-1} \overrightarrow{p'_i} \# \mathfrak{s}_i$，则将其映射为 $v_i v'_i \overrightarrow{p'_i} \mathfrak{s}_i$；如果其输入的形式为
- $v_n p_n \# v'_n \overleftarrow{p'_n} \mathfrak{s}_{n-1} \# \mathfrak{s}_n$，则将其映射为 $v_n p_n \mathfrak{s}_n$

证明。我们首先将第二个 # （如果存在）替换为 |，然后应用一个映射复制，其分隔符为 |。因此，如果输入原本具有第二种（分别为第三种）形式，现在就变为 $v_i p_i \# v'_i \mathfrak{s}_{i-1} \overrightarrow{p'_i} | v_i p_i \# v'_i \mathfrak{s}_{i-1} \overrightarrow{p'_i} | \mathfrak{s}_i \cdots$（分别为 $v_n p_n \# v'_n \overleftarrow{p'_n} \mathfrak{s}_{n-1} | v_n p_n \# v'_n \overleftarrow{p'_n} \mathfrak{s}_{n-1} | \mathfrak{s}_n \cdots$）。最后，我们应用一个顺序函数，输出它所看到的内容，直到遇到一个因子 $s\#$，其中 $s \in Q$。在这种情况下，它读取 # 与 | 之间的下一个因子（不进行写入），以确定输入具有第二种还是第三种形式，然后相应地处理最后一部分。对于不含 # 的（有限或无限）词，这一过程不会改变它们。◀

然后，我们将函数 [normalize |] 应用于整个输入。注意，在命题 A.21 的最后一种情况下，我们撤销了 $\overleftarrow{\mathsf{run}_k}(\mathfrak{s}_{i-1} \overleftarrow{p_i})$ 的计算，而状态 p_i 不再带有向上箭头：它某种程度上被冻结了（但请记住，这种情况只可能出现在最后一个因子中），其余部分 ρ_i 将在第 10 项中模拟。

8. 将第 3–7 项中描述的步骤迭代 $|Q|$ 次，即可模拟运行 ρ_i，其中 $i \geq 2$（可能最后一次运行除外，见上文）。事实上，这些运行访问各自的 $|Q|$ 首个位置的次数至多为 u_i。因此，我们得到的输出具有以下某种形式（其中 $v_1 := \lambda(\overrightarrow{q} u_1)$）：

- 如果 $\delta(\overrightarrow{q}, u_1) = \omega$ 且 $v_1 \in \Gamma^\omega$，那么 v_1（无需再做任何事）；如果 $\delta(\overrightarrow{q}, u_1) = \overleftarrow{p_1}$，那
- 么 $v_1 \overleftarrow{p_1} \mathfrak{s}_1 | \mathfrak{s}_2 \cdots$（无需再做任何事）；如果 $\delta(\overrightarrow{q}, u_1) = \overrightarrow{p}$（必然 $u_1 \in A^+$），那么会出
- 现两种主要情况：没有状态被冻结（也就是说，每个 ρ_i 都已被完全模拟），于是我们得到一个形如 $v_1 \mathfrak{s}_1 | v_2 \mathfrak{s}_2 | \cdots$ 的字（可能以 $v_i \in \Gamma^\omega$ 结尾；若输入是有限的，也可能以 $\overrightarrow{r}$ 结尾），其中每个 v_i 恰好是 T 沿着引理 A.14 的 ρ_i 得到的输出。在这种情况下，接下来只需重新组合 v_i（若输入有限，还需组合 $\mathfrak{s}_i$）；或者某个状态 $s \in Q$ 在某个时刻（最后一个因子中）被冻结，于是我们得到一个形如 $v_1 \mathfrak{s}_1 | \cdots | v_n s \mathfrak{s}_n$ 的字，其中 $i < n$ 对应的 v_i 是 T 沿着 ρ_i 得到的输出，而 v_n 是 T 沿着 ρ_n 得到的输出，直到它在 $u_{n-1} \in \Sigma^+$的最后一个位置被访问时为止，此时它处于某个状态。在这种情况下，我们仍需生成一些输出。

9. 现在，我们声称可以应用一个 $\mathcal{C}$ 函数，使结果如下：

- 如果 $\delta(\overrightarrow{q}, u_1) = \omega$ 且 $v_1 := \lambda(\overrightarrow{q} u_1) \in \Gamma^\omega$，则 v_1（无需处理）；如果 $\delta(\overrightarrow{q}, u_1)$
- $= \overleftarrow{p_1}$ 且 $v_1 := \lambda(\overrightarrow{q} u_1) \in \Gamma^*$，则 $v_1 \overleftarrow{p_1} \mathfrak{s}_1 | \mathfrak{s}_2 \cdots$（无需处理）；否则，如果 $\delta(\overrightarrow{q}, u_1)$
- $= \overrightarrow{p}$：要么在此过程中没有冻结任何状态，并且我们有 $v \in \Gamma^\omega$（无限输入）或 $v \mathfrak{s}_1 | \cdots | \mathfrak{s}_n \overrightarrow{s}$（有限输入），其中 $v := \lambda(\overrightarrow{q} u)$（无需处理）；要么冻结了一个状态，并且我们得到 $v \# \mathfrak{s}_1 | \cdots | \mathfrak{s}_{n-1} | \overleftarrow{s} \mathfrak{s}_n$，其中 $u = u_1 \cdots u_n$，且 $u_n \in \Sigma^\infty$ 和 v 是引理 A.14 的运行 ρ 过程中产生的输出，直到在 u_{n-1}的最后一个位置访问某处为止；此次访问是在状态 $s \in Q$下完成的。

为了构造这样的 $\mathcal{C}$ 函数，我们先在任何 $s, \overleftarrow{s}$ 或 $\overrightarrow{s}$（如果存在）之前加上一个 #。然后应用 [map-copy # id #]，并用第一个副本生成 $v_1 v_2 \cdots v_n$（必要时末尾加上一个 #），用第二个副本生成 $\mathfrak{s}_1 | \cdots | \mathfrak{s}_{n-1}$。最后，我们加上 $\overleftarrow{s} \mathfrak{s}_n$ 或$\mathfrak{s}_n \overrightarrow{s}$。注意，如果没有$s, \overleftarrow{s}$也没有 $\overrightarrow{s}$，这一构造仍可正确扩展：输入是无限的，而我们只生成 v_i。

10. 最后，简要说明如何处理条目 9 的最后一种情况，也就是我们有 $v\#\mathfrak{s}_1|\cdots|\mathfrak{s}_{n-1}|\overleftarrow{s}\ \mathfrak{s}_n$ 且 $u_n \in \omega^\infty$ 的情形。我们先将 $|\overleftarrow{s}$ 转换为 $\overleftarrow{s}\#$。然后应用函数 $[\mathrm{id}\ \#\overleftarrow{\mathrm{run}}^{k+1}\#\ \mathrm{id}\ \#]$，以模拟在前缀上的一次运行（注意，此处需要 $\overleftarrow{\mathrm{run}}^{k+1}$，而不是 $\overleftarrow{\mathrm{run}}_k$）。结果要么在左侧给出一个输出和一个状态 $\overleftarrow{t}$，要么在右侧给出它们 $\overrightarrow{t}$。如果存在 $\overrightarrow{t}\#$，我们就将其替换为 $\#\overrightarrow{t}$，然后应用 $[\mathrm{id}\ \#\ \mathrm{id}\ \#\overrightarrow{\mathrm{run}}_k\#]$，以模拟在后缀上的一次运行。接着，以类似于条目 9 的方式重新组合输出。由于 u_n 的第一个位置至多被访问 $|Q|$ 次，我们可以通过迭代这些操作 $|Q|$ 次来模拟整个运行 ρ。

B 受限 MSO 转导

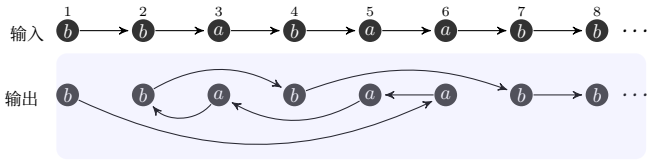
B.1 MSO 转导$_g$ 带全序，以及命题 4.9 的证明

在第 4 节对 MSO 转导和 MSO 转导$_g$ 的定义中，并不要求定义输出结构的全序 $\leq$。这里我们考虑这样的 MSO 转导：要求定义全序，而不是后继。虽然这不会改变 MSO 转导的表达能力，因为后继可以由序表达，反之亦然，但对于受限片段而言情况有所不同。我们证明，这一修改使得我们能够定义一些并非确定性正则的函数。

在本节中，我们写作 MSO 转导$_g[S]$ 而不是 MSO 转导$_g$，以强调这里只定义了后继。我们将带序的守卫 MSO 转导记作 MSO 转导$_g[\leq]$，其定义与 MSO 转导$_g[S]$ 完全相同，只是将后继公式替换为序公式 ϕ^c、$\overset{d}{\leq}(x、\ y)$，以及任意副本 $c、\ d$。其语义如下。设 $T \in$ MSO 转导$_g[\leq]$的输入字母表为 Σ，输出字母表为 Γ。对于输入 $u \in \Sigma^\omega$，T 定义一个输出结构，记作 $T(u)$，其签名为 $\{\leq(x、\ y)、(\gamma(x))_{\gamma\in\Gamma}\}$。随后，我们将从 $T(u)$ 得到的、签名为 $\{S(x、\ y)、(\gamma(x))_{\gamma\in\Gamma}\}$ 的结构记作 $T(u)|_S$，其中将 S 解释为 $\leq$ 在 $T(u)$ 中的解释的单步后继。注意，此时结构 $T(u)|_S$ 可能并不对应一个词结构，它只是 $\{S(x、\ y)、(\gamma(x))_{\gamma\in\Gamma}\}$ 上的某个结构。至于 MSO 转导 $_g[S]$，我们仅针对词结构定义 T 的语义：

$$[\![T]\!] = \{(u, v) \in \Sigma^\omega \times \Gamma^\omega \mid T(u)|_S \text{ is isomorphic to } G_v\}$$

▶ 例 B.1. 设 $\Sigma = \Gamma = \{a, b\}$，且 $\mathrm{er}_b : \Sigma^* \to \Sigma^*$ 为如下（擦除）态射：$\mathrm{er}_b(a) = a$，且 $\mathrm{er}_b(b) = \varepsilon$。我们定义函数 $h : \Sigma^\omega \rightharpoonup \Gamma^\omega$，其输入为形如$bub^\omega$的字符串，其中 $u \in \{a, b\}^*$，定义为 $h(bub^\omega) = b\mathrm{er}_b(u)b^\omega$。函数 h 不连续（因此不是确定性正则函数）：$\lim_{n\infty} h(b^n ab^\omega) = bab^\omega \neq h(\lim_{n\infty} b^n ab^\omega) = h(b^\omega) = b^\omega$。我们证明它可由一个 1-复制 MSO 转导$_g[\leq]$定义。输入$bbabaab^\omega$ 上的一个输出结构示例如下（我们只画出后继谓词，不画出序）：



为定义这些公式，我们使用受限公式 $\mathrm{first}(x) = \exists \mathrm{g} \forall y \leq x \leq \mathrm{g} \cdot y = x$，当 x 是第一个输入位置时，该公式成立。注意，它的否定等价于受限公式 $\exists y \leq x \cdot y \neq x$。然后，通过 $\phi_{\mathrm{fst}}(x) = \mathrm{first}(x) \wedge b(x)$ 定义第一个输出元素；当 x 是第一个输入元素且其标签为 b 时，该公式成立。如果第一个元素是

未标记为 b，那么后一个公式不成立；根据 MSOT$_g$ 的语义定义，输出结构为空。因此，它与任何无限词结构都不同构，输入词也就不属于定义域。标签公式保留标签：对所有 $\sigma \in \{a, b\}$，$\phi_\sigma(x) = \sigma(x)$。

定义顺序公式 $\phi_\leq(x, y)$。如果 x 是第一个输入节点，那么它小于任意 y。如果 x 是第二个节点，那么它小于之后出现的任意 b。如果 x 标记为 a，那么它小于除第一个之外的任意 b，也小于任何出现在它左侧的其他 a：

$$\begin{aligned}\phi_\leq(x,y) \quad = \quad & (b(x) \wedge b(y)) \to x \leq y \wedge \\ & (b(x) \wedge a(y)) \to \mathsf{first}(x)) \wedge \\ & a(x) \to (b(y) \wedge \neg \mathsf{first}(x)) \vee (a(y) \wedge y \leq x)\end{aligned}$$

注意，上述所有公式都受保护。我们没有显式地将它们写成受保护的形式，但只需使用 $max(x, y)$ 作为守卫即可。

这证明了命题 4.9。

C 有限前瞻的消除

在本节余下部分，令 $\mathcal{T} = (\Sigma, \Gamma, Q, q_0, F, \delta, \lambda)$ 为一个 2-dT^{FLA}。不失一般性，我们假设只有向右移动的转移使用非平凡前瞻语言，而向左移动的转移使用平凡前瞻语言 Σ^*（即无前瞻）。

这可以通过将每个借助非平凡前瞻向左移动的转移，替换为一个带有该前瞻、向右移动的转移和两个不带前瞻、向左移动的转移来实现。这种构造还有一个方便的副作用：对于任一状态，要么只允许向右转移，要么只允许向左转移。

设 $L_1, \ldots, L_n$ 是 δ 中出现的正则前瞻语言。不失一般性，我们假设这些前瞻语言可由同一个确定有限自动机接受，只是终态不同。形式化地，设 $\mathsf{A} = (\Sigma, Q_\mathcal{A}, q_0^\mathcal{A}, F_\mathcal{A}, Q_\mathcal{A}, \delta_\mathcal{A})$ 是一个确定有限自动机，且 $c\colon Q_\mathcal{A} \to \mathrm{P}(\{1, \cdots n\})$ 是其状态的一个（多重）着色。对于所有 $1 \leq i \leq n$，令 $F_i = \{q \in Q_\mathcal{A} \mid c(q) = i\}$，并令 $\mathsf{A}_i = (A, Q_\mathcal{A}, q_0^\mathcal{A}, F_\mathcal{A}, F_i, \delta_\mathcal{A})$。我们假设 $L(\mathsf{A}_i) = L_i$。我们将 A 称为前瞻自动机。我们为其状态集添加一个指定的汇状态 q_{Σ^*}，其颜色为 0，以表示平凡前瞻语言 Σ^*（即无前瞻）。令 C 为颜色集合 $\{0, 1, \ldots, n\}$。示例见图 3。

C.1 前瞻树

我们引入一些记号。

▶ **定义 C.1**（前瞻信息）。前瞻信息 是形如 (p, c) 的元组，其中 $p \in Q_\mathcal{A}$，且 $c \in C$。此类元组的预期含义是：前瞻自动机当前处于状态 p，而我们希望到达一个颜色为 c 的状态。

如果 $c \in c(p)$，我们就称前瞻信息 (p, c) 已成功。

给定前瞻信息 (p, c) 和一个字母 $\sigma \in \Sigma$，若 (p, c) 尚未成功，我们定义向上 $((p, c)\ \sigma) = (\delta_\mathcal{A}(p, \sigma)\ c)$；否则定义向上 $((p, c)\ \sigma) = (p, c)$（因为一旦我们知道某个前瞻信息是否已成功，就希望将其状态固定下来）。

令 I 表示所有前瞻信息的集合。给定子集 $M \subseteq I$ 和字母$\sigma \in \Sigma$，我们定义 向上$(M, \sigma) = \{$向上$(i, \sigma) \mid i \in M\}$。

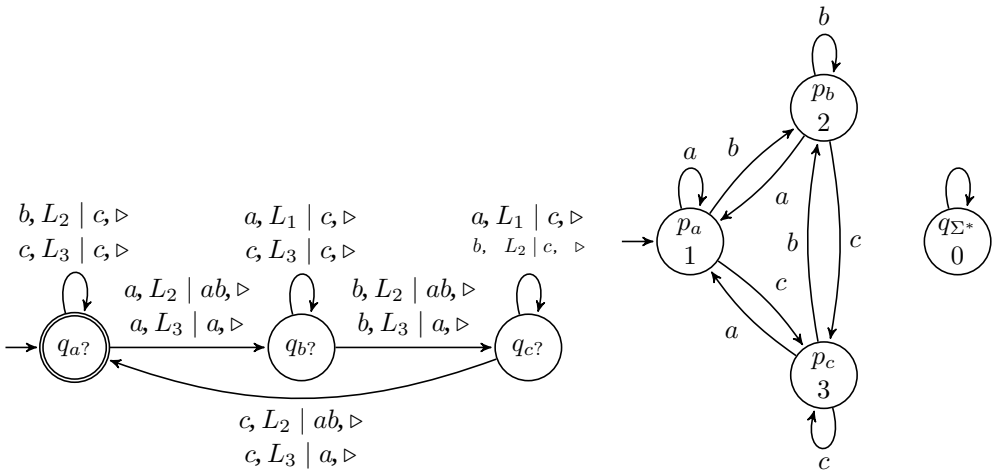


图 3 左侧是一个具有有限前瞻的确定性双向转换器，右侧是相应的前瞻自动机。

我们引入前瞻树的概念，并对其进行归纳定义。前瞻树是节点带有来自 $Q \times I$ 的标签的树，也就是说，它们存储双向转换器的一个状态和一个前瞻信息集合。图 4 和图 6 展示了其示意图。

▶ **定义 C.2** (前瞻树)。这组单节点树是前瞻树：$\{(q, \{(q_{\Sigma^*}, 0)\}) \mid q \in Q\}$。

给定一棵前瞻树 t 和一个字母 $\sigma \in \Sigma$，依次应用下文定义的以下操作（$t_1 =$向上(t, σ)，然后是 $t_2 = \mathsf{del}_{\mathsf{success}}(t_1)$，再然后是 $t_3 = \mathsf{del}_{\mathsf{fail}}(t_2)$，最后是 $t' =$ 合并(t_3)），即可得到一棵新的前瞻树 t'。

更新

更新过程的第一步是根据字母 σ更新所有前瞻信息。因此，我们重新为每个节点标注标签。形如 (q, M) 的标签被替换为 $(q,$ 向上$(M, \sigma))$。

如果一个标签为 (q, M) 的节点，其存储在 M 中的所有前瞻信息都已成功，则称该节点成功 。树中的一条路径称为成功 路径，当且仅当其所有节点都已成功。

更新过程的第二步是向树中添加新节点。对于一个状态q，我们收集在字母 σ 下可能启用的所有 $\triangleright$-转移。令$S_q = \{i \mid \delta(q, \sigma, L_i) \neq \bot\}$。

对于标签为 (q, M) 的叶节点，如果满足 $S_q \neq \emptyset$，则将以下节点添加为其子节点。对于所有$i \in S_q$，如果满足 $\delta(q, \sigma, L_i) = (q', \triangleright)$，则添加一个标签为 $(q', \{(q_0^A, i)\}$ 的子节点。

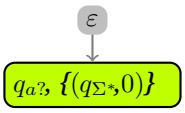
否则（如果 $S_q = \emptyset$），我们添加一个标签为*"fail"*的子节点。

我们将此过程称为向上$(\cdot\ \sigma)$。

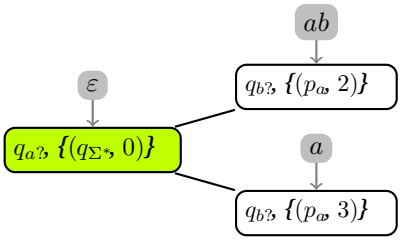
删除

删除步骤的目的有两个。

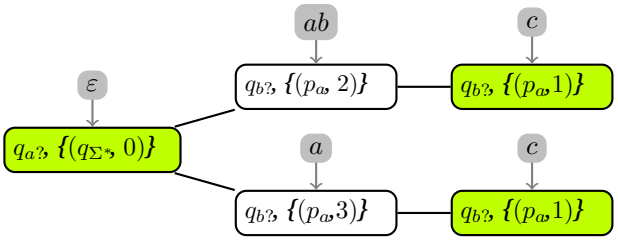
一方面，如果一条从根节点到叶节点的路径是成功的，我们就希望将其移除，因为这表明该路径存储了有关接受运行的信息（我们已经看到这条路径上的前瞻的见证）。另一方面，我们希望移除那些已知不存储有关接受运行信息的路径。



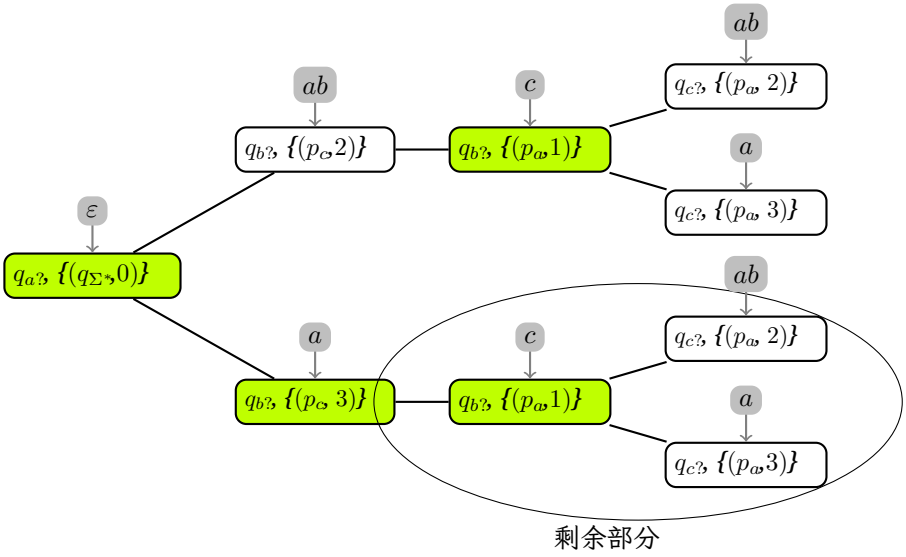
(a) 初始前瞻树



(b) 应用 up(· a) 后；应用 $\text{del}_{\text{success}}(\cdot)$、$\text{del}_{\text{fail}}(\cdot)$ 和 merge(·) 后不再有变化

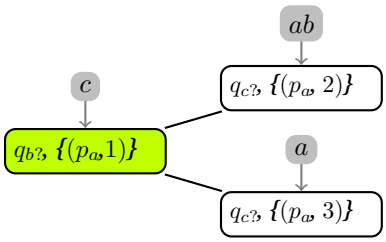


（**c**）应用 up(· a) 后；应用 $\text{del}_{\text{success}}(\cdot)$、$\text{del}_{\text{fail}}(\cdot)$ 和 merge(·) 后不再发生变化

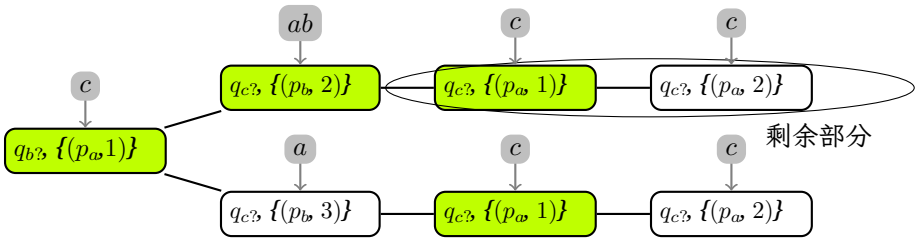


(d) 应用 up(· b) 后；应用 $\text{del}_{\text{success}}(\cdot)$ 后的结果以标记表示；来自待删除部分的输出 $\varepsilon \cdot a$ 被追加到 out

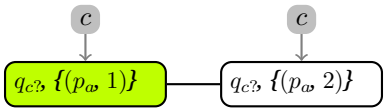
图4 根据图 3 所示的转导器构造前瞻树的概述。图 4a–4c、4e、4g、4i 和 4j 展示了前瞻树；图 4d、4f 和 4h 展示了中间构造步骤。以石灰色背景绘制的节点表示已成功。以灰色背景绘制的节点不属于这棵树；它们用于展示与树中节点关联的寄存器。此图的第二部分见第 29 页。



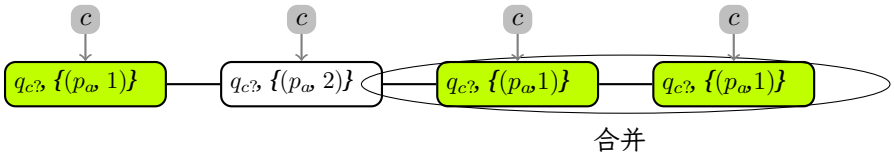
(e) 应用 $\mathrm{del}_{\mathrm{success}}(\cdot)$ 后；应用 $\mathrm{del}_{\mathrm{fail}}(\cdot)$ 和 merge(·) 后不再发生变化



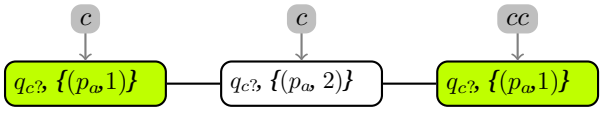
(f) 应用 up(· a)、$\mathrm{del}_{\mathrm{success}}(\cdot)$、$\mathrm{del}_{\mathrm{fail}}(\cdot)$、merge(·) 和 up(· b) 后；应用 o 后的结果 f $\mathrm{del}_{\mathrm{success}}(\cdot)$ 被标记；待删除部分的输出 $c \cdot ab$ 被追加到 out



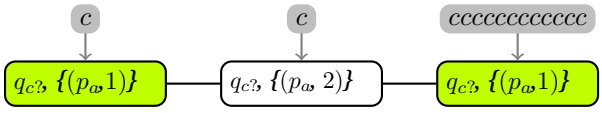
(g) 应用 $\mathrm{del}_{\mathrm{success}}(\cdot)$ 后；应用 $\mathrm{del}_{\mathrm{fail}}(\cdot)$ 和 merge(·) 后不再发生变化



(h) 应用 up(· a)、$\mathrm{del}_{\mathrm{success}}(\cdot)$、$\mathrm{del}_{\mathrm{fail}}(\cdot)$、merge(·)、up(· a)、$\mathrm{del}_{\mathrm{success}}(\cdot)$ 和 $\mathrm{del}_{\mathrm{fail}}(\cdot)$ 后，将受 merge(·) 影响的节点标记出来



(i) 应用 merge(·) 后



(j) 在应用向上操作 up(· a)、$\mathrm{del}_{\mathrm{success}}(\cdot)$、$\mathrm{del}_{\mathrm{fail}}(\cdot)$ 和合并操作 merge(·) $10\times$ 之后

图4 基于图 3 所示的转导器构造前瞻树的概览。图 4a–4c、4e、4g、4i 和 4j 展示了前瞻树，图 4d、4f 和 4h 展示了中间构造步骤。以石灰色背景绘制的节点表示已成功。以灰色背景绘制的节点不属于树；它们表示与树中节点关联的寄存器。该图的第一部分见第28页。

删除步骤之后，前瞻树中只包含存储以下运行信息的节点：其中某些前瞻仍有待验证。

第一步，如果树中存在一个节点 n，它不是根节点 r，且从 r 到 n 的路径是成功的，并且 n 没有任何成功的子节点，那么 n 就成为新的根节点。请注意，前瞻的不相交性假设保证了至多存在这样一条路径。

我们将此过程称为 删除$_{\text{success}}(\cdot)$。

第二步，我们移除那些不包含接受运行信息的节点。因此 ，
执行以下步骤：

- 如果某个叶节点标记为*"fail"*，则删除该叶节点及其所有祖先节点，直到某个祖先节点至少有两个子节点。这表示双向转换器的这次运行失败了。如果从根节点到叶节点的两条
- 路径描述了相同的情形（稍后解释），我们可以将它们同时删除。更具体地说，给定一条从根节点到叶节点的路径 $\pi = n_1 n_2 \cdots n_i$，其中对所有 $1 \leq j \leq i$，节点 n_j 的标签都是 (q_j, M_j) 。我们用摘要(π) 表示集合 $\bigcup_{j=1}^{n} M_j$。它概括了这条路径上必须成立的前瞻状态。设 π_1 和 π_2 是从根节点到两个叶节点的路径，叶节点标签分别为 (q, M) 和 (q, M')，且 摘要(π_1) $=$摘要(π_2)。根据构造，这些路径描述了双向转换器的不同 运行，但它们的前瞻状态相同。这意味着这两次运行必定都失败或都成功。两次运行不可能都成功，因为这与双向转换器的确定性相矛盾。因此，这两次运行都会失败，我们可以从前瞻树中移除它们的表示。为此，每当存在两条路径，其叶节点存储了双向转换器的相同状态，且它们的前瞻摘要相等时，就删除这些叶节点及其所有祖先节点，直到某个祖先节点至少有两个子节点。

我们将此过程称为删除$_{\text{fail}}(\cdot)$。

合并

最后，如果树包含一个长度至少为 *3* 的线性分支段，也就是说，存在节点 n_1、n_2、……、n_j，使得 $j > 3$，且 n_{i+1} 是 n_i 的子节点，并且对于 $1 \leq i < j$，n_i 没有其他子节点，那么将按如下方式合并该分支：删除节点 n_3、……、n_{j-1}，并将已删除节点中的前瞻信息推送至 n_j。设 M_3、……、$M_{j_1} \subseteq I$ 是它们的前瞻信息集合，那么我们用 $\bigcup_{i=3}^{j} M_i$ 替换节点 n_j 的前瞻信息 M_j 。注意，执行合并后，可能会产生另一个线性分支段，因此必须重复合并操作，直到不再发生变化。

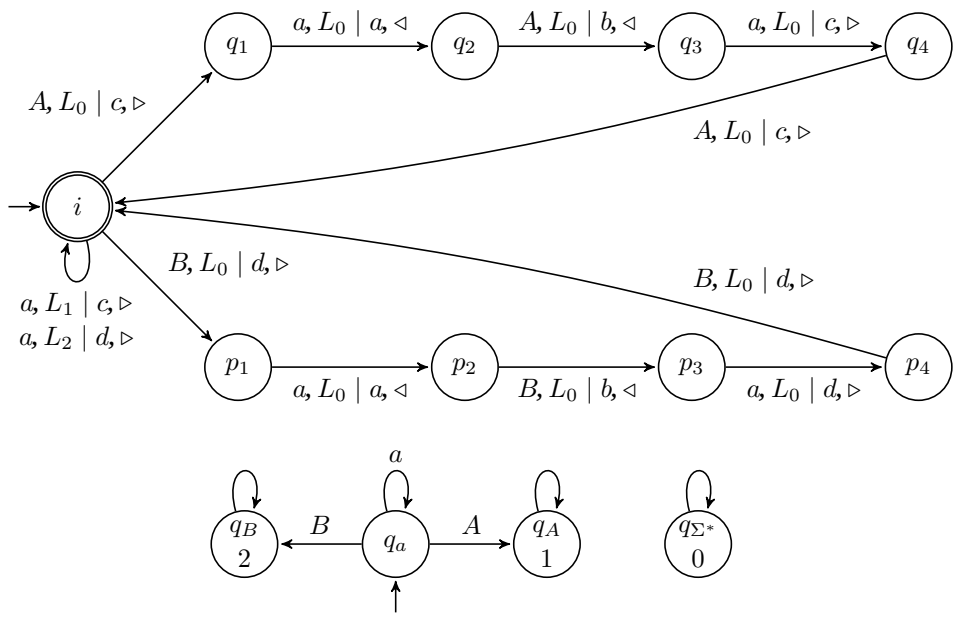
我们将此过程称为合并$(\cdot)$。

▶**引理 C.3**。 前瞻树的集合是有限的。

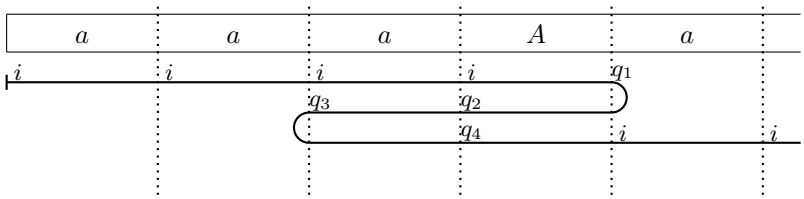
证明。 这可直接由归纳定义得出。前瞻摘要的集合是有限的，且线性分支段的高度有界。因此，前瞻树的集合是有限的。◀

C.2 Büchi 确定性串流串转换器的构造

现在我们已经定义了前瞻树，接下来设计一个 SST（不含前瞻），用来模拟具有有限前瞻的确定性双向转换器的运行。



(a) 上：带有限前瞻的确定性双向转换器。下：对应的前瞻自动机。



(b) 转换器在输入词 $aaaAa\cdots \in \Sigma^\omega$ 上的运行。请注意，即使转换器向左移动，它也始终读取右侧的符号。

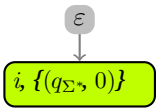
图5 带有限前瞻的确定性双向转换器及其对应的运行 n.

主要困难在于，为了模拟双向转换器的运行，流式字符串变换器无法预先获知哪种前瞻会成功。因此，SST 必须跟踪所有可能的运行，并验证所假设的前瞻是否确实可用。对于转导器的每次模拟转移，这可能需要无界的时间。因此，在确定哪条运行才是正确的之前，不可能将跟踪的运行存储在 SST 的状态空间中。

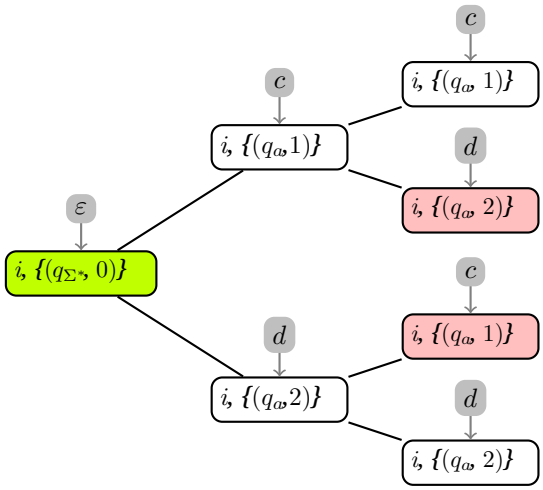
在上一节中，我们介绍了前瞻树。接下来，我们将给出一种构造方法，使我们能够使用前瞻树存储所跟踪运行的抽象表示。

我们的目标是设计一个BSST $\mathcal{S}$：给定一个输入词 $u \in \Sigma^\omega$，它在 u 的一个前缀 u' 上维护可能在 u' 上发生的从左到右运行和从右到右运行的（抽象）信息。此外，这些运行的输出必须存储在适当的寄存器中，以便一旦确定这些（部分）运行中哪些属于转换器所执行的（整体）运行，就能用它来构造 SST 的输出。

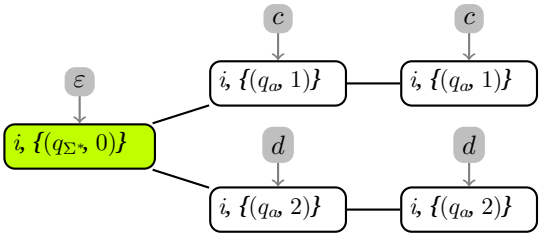
现在，我们描述 SST 的状态、如何用一个状态表示一次运行、SST 的哪些寄存器包含该运行的输出，以及如何更新指定输出寄存器 $\mathfrak{out}$。



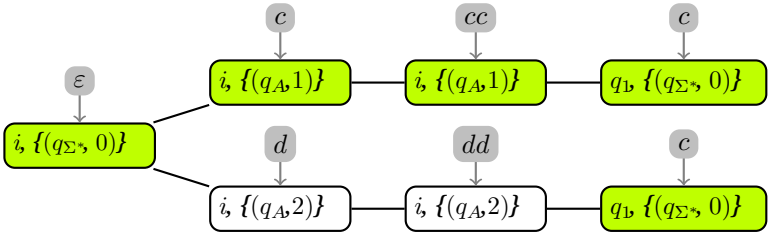
(a) 初始前瞻树



(b) 应用 up(· a)、$\text{del}_{\text{success}}(\cdot)$、$\text{del}_{\text{fail}}(\cdot)$、merge(·)、up(· a) 和 $\text{del}_{\text{success}}(\cdot)$ 后；红色节点表示前瞻信息摘要相同的分支，因为这些节点中的状态也相同，所以这些分支必须被删除

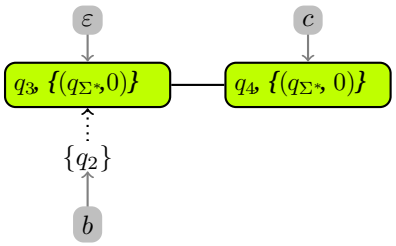


(c) 应用 $\text{del}_{\text{fail}}(\cdot)$ 和 merge(·) 之后

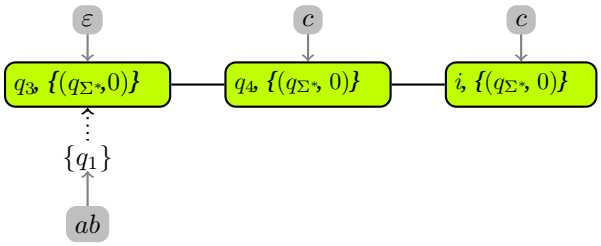


(d) 应用 up(· a)、merge(·)、$\text{del}_{\text{success}}(\cdot)$、$\text{del}_{\text{fail}}(\cdot)$、merge(·) 和 up(· A) 后；该树表示图5中的转导器在 $aaaAa$ 的前缀 $aaaA\cdots$ 上的（部分）从左到右运行；未表示的从左到右运行已在先前步骤中被确定为注定失败（例如，参见图6b）

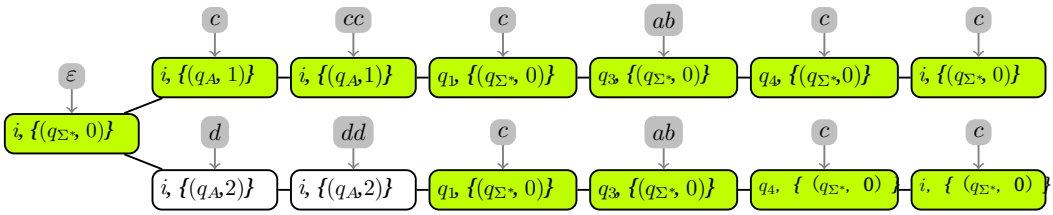
图6 概述如何根据图5所示转导器在输入 $aaaAa\cdots \in \Sigma^\omega$ 上，使用前瞻树表示从左到右运行和从右到右运行。图 6a、6c、6e、6f 和 6h 表示为图5所示转导器构造的 SST 状态空间中存储的对象。图 6b 、 6d 和 6g 表示确定 SST 新状态的中间构造步骤。这些图中的所有灰色节点都表示寄存器及其内容。该图的第二部分见第33页。



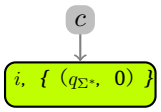
(e) 该前瞻树及其关联集合表示图5所示转导器在 $aaaAa\cdots \in \Sigma^\omega$ 的前缀 aaa 上从 q_2 到 q_3 的从右到右运行。



(f) 该前瞻树及其相关集合表示了图5中转导器在 $aaaAa\cdots \in \Sigma^\omega$的前缀 $aaaA$ 上从 q_1 到 q_4 的从右到右转换。



（**g**）图 6d 所表示的从左到右运行和图 6f 所表示的从右到右运行可以连接起来。这展示了所得的表示。



(h)（对图 6g 中的树应用操作 $\mathrm{del}_{\mathrm{success}}(\cdot)$、$\mathrm{del}_{\mathrm{fail}}(\cdot)$ 和 merge(·) 的结果。
o输出 $\varepsilon \cdot c \cdot cc \cdot c \cdot ab \cdot c$ 来自被移除部分的内容已添加到寄存器 out 中。

图6 概述如何根据图5所示的转导器，在输入 $aaaAa\cdots \in \Sigma^\omega$ 上使用前瞻树表示从左到右运行和从右到右运行。图 6a、6c、6e、6f 和 6h 表示为图5所示转导器构造的 SST 状态空间中存储的对象。图 6b、6d 和 6g 表示确定 SST 新状态的中间构造步骤。这些图中的所有灰色节点都表示寄存器及其内容。该图的第一部分见第 32 页。

这些步骤的可视化结果如图6所示。

基本上，对于 $u \in \Sigma^\omega$ 的一个前缀 $u' \in \Sigma^*$，我们希望表示它所有从左到右和从左到左的运行，以及它们在 u' 上存储的输出。

我们更精确地说明什么是从左到右运行和从右到右运行。回想一下，根据假设，在 u 上 $\mathcal{T}$ 的配置 $(q,\ i)$ 中，要么只有向右移动的转移是启用的，要么只有一个向左移动的转移是启用的（因为向左移动时没有需要验证的前瞻）。

▶ **定义 C.4** (从左到右运行，从右到右运行)。在 $u \in \Sigma^\omega$ 的一个前缀 $u' \in \Sigma^*$ 上，从 p 到 q 的从左到右（分别为 从右到右）运行，是指从 $(p,0)$ 开始（分别从 $(p,|u'|)$ 开始）并在 $(q,|u'|)$ 结束的运行，且在 $(q,|u'|)$ 中，只有 $\triangleright$-转移是启用的，并且运行中不会访问位置 $> |u'|$ 。请注意，该定义要求从右到右运行至少执行一次转移。

令 s_0 为$\mathcal{S}$的初始状态。我们首先描述$\mathcal{S}$ 读取来自 s_0 的字母 $\sigma \in \Sigma$ 后到达的状态，以及其在前缀 σ 上的相关运行和寄存器内容。

然后，我们描述如何根据在 $u' \in A^*$ 处到达的状态，确定对于某个 $\sigma \in \Sigma$ 在 $u'\sigma$ 处到达的状态，以及其相关运行和寄存器内容。

读取 a 后，令寄存器 $\mathfrak{out}$ 保存 ε。在前缀 $\sigma \in \Sigma$上，我们可能有多个 al 从左到右运行，形式为 $(q_0,0)(q',1)$，其中 $\delta(q_0^{\mathcal{A}},\sigma,L) = (q',\triangleright)$。

容易看出，这些运行在前瞻树 $\mathsf{up}((q_0,\{(q_{\Sigma^*},0\})\ \sigma)$ 中的表示方式是沿着从根节点到相应叶节点的路径。我们为 t 的根节点关联一个寄存器（例如 $\mathfrak{r}$），其内容为 ε。如果 τ 是使这一步得以执行的转移，则为叶节点关联一个寄存器，其内容为 $\lambda(\tau)$。

那么，运行 $(q_0,\ 0)(q',\ 1)$ 的输出为 $[\![\mathfrak{out}]\!][\![\mathfrak{r}]\!][\![\mathfrak{l}]\!]$，其中 $\mathfrak{l}$ 是为导向 $(q',\ 1)$ 的转移新增的寄存器。

形式化地说，为了确定某个前缀上的从右到右运行，我们还需要知道一个输入字母，因为从位置 i 向 $i-1$ 移动取决于输入字母 i。因此，为便于表述，不失一般性，我们假设始终提供这一信息。

对于前缀 $\sigma \in \Sigma$（假设下一个字母是 σ'），我们可能会有如下形式的从右到右运行：$(q_1,1)(q_2,0)\cdots(q_n,1)$，其中 $n \geq 3$ 是奇数，必须在 $(q_n,1)$ 中进行一次向右移动，并且对于所有奇数 $i \leq n$，$\delta(q_i,\sigma') = (q_{i+1},\triangleleft)$；对于所有偶数 $i \leq n$，$\delta(q_i,\sigma,L_{j_i}) = (q_{i+1},\triangleright)$。这样的运行由一个前瞻树和一个状态集组成的对表示如下：前瞻树为 $(q_n,\{\bigcup_{i=2}^{n-2},_i \mathsf{even}(q_0^{\mathcal{A}},j_i\})$，其对应的状态集为 $\{q_1\}$。我们将一个寄存器（例如 $\mathfrak{r}$）关联到该树的根节点，其内容为 ε；再将一个寄存器（例如 $\mathfrak{s}$）关联到 q_1，用于存储 $\lambda(\tau_1)\lambda(\tau_2)\cdots\lambda(\tau_{n-1})$，其中 $\tau_1\cdots\tau_{n-1}$ 是运行 $(q_1,1)(q_2,0)\cdots(q_n,1)$ 中采取的转移序列。于是，其输出为 $[\![\mathfrak{s}]\!][\![\mathfrak{r}]\!]$。

现在描述一般步骤。令 $u' \in \Sigma^*$ 为 $u \in \Sigma^\omega$的一个非空前缀，令 s为读取 u'后到达的 $\mathcal{S}$ 的状态，令下一个输入字母为 a，再下一个输入字母为 σ'。

状态 s 存储一棵前瞻树 t，以及可能的若干对 $(t_1,\ M_1)\ldots,\ (t_n,\ M_n)$ $\in$前瞻信息树集 $\otimes$ $\mathcal{P}(Q)$，其中前瞻信息树集是前瞻树的集合，$\mathcal{P}(Q)$ 是 Q 的幂集（双向转换器 $\mathcal{T}$的状态集）。

我们假设 $t,\ t_1,\ \ldots,\ t_n$ 满足以下不变量。为便于表述，若 π是树中的一条路径，则令 $\pi_{\mathfrak{R}}$ 表示与之关联的寄存器序列 $\mathfrak{r}_1\cdots\mathfrak{r}_j$。接着用 $[\![\pi_{\mathfrak{R}}]\!]$ 表示 $[\![\mathfrak{r}_1]\!]\cdots[\![\mathfrak{r}_j]\!]$。

▷ 断言 C.5. 在 t 中，设 π 是一条从根节点到叶节点的路径，其标签形式为 $(q,\ X)$。那么，存在一个从左到右的 $\mathcal{T}$，它在 u' 上从 q_0 到 q，输出为 $[\![\mathfrak{out}]\!][\![\pi_{\mathfrak{R}}]\!]$。

在 $(t_i,\ M_i)$ 中，设 π 是一条从根节点到叶节点的路径，其标签形式为 $(q,\ X)$，并设 $p \in M_i$ 和 $\mathfrak{r}$ 是与 $p \in M_i$ 关联的寄存器。那么，存在一个从右到右的 $\mathcal{T}$，它在 u' 上从 p 到 q，输出为 $[\![\mathfrak{r}]\!][\![\pi_{\mathfrak{R}}]\!]$。

▷ 断言 C.6。若 $u \in$ 定义域$(\mathcal{T})$，则输出 $\mathcal{T}(v)$ 可写为 $[\![\mathfrak{out}]\!]v$，其中存在 $v \in \Gamma^\omega$，且存在一条路径 π，从根节点通向 t 中的一个叶节点，使得 $[\![\pi_{\mathfrak{R}}]\!] \sqsubseteq v$。

注意，断言 C.5 和 C.6 对前缀 $\sigma \in \Sigma$ 上到达的状态同样成立。

为确定 $\mathcal{S}$ 的下一个状态，我们构造

$$\tilde{t} = \mathsf{up}(t, \sigma),\ \text{and}\ \tilde{t}_i = \mathsf{up}(t_i, \sigma)$$

对所有 $1 \le i \le n$ 均如此。我们为每个新插入的叶节点关联一个寄存器，用于存放构造该叶节点时所用转移的输出。

显然，若 π 是一条从根节点通向 t 中某个叶节点的路径，且 π' 是一条从根节点通向 $\tilde{t}$ 中某个叶节点的路径，并且 $\pi \sqsubseteq \pi'$：若 π 表示 $\mathcal{T}$ 在 u' 上从 q_0 到 q 的一次从左到右运行，其输出为 $[\![\mathfrak{out}]\!][\![\pi_{\mathfrak{R}}]\!]$，那么 π' 表示 $\mathcal{T}$ 在 $u'a$ 上从 q_0 到 q' 的一次从左到右运行，其输出为 $[\![\mathfrak{out}]\!][\![\pi'_{\mathfrak{R}}]\!] = [\![\mathfrak{out}]\!][\![\pi_{\mathfrak{R}}]\!][\![\mathfrak{r}]\!]$，其中 $\mathfrak{r}$ 是与为转移 $\delta(q, \sigma, L) = (q', \triangleright)$ 引入的叶节点相关联的寄存器。

~ ~ ~对于这些树 $t_1, \dots, t_n$，我们更新关联集合 $M_1, \dots, M_n$。设 M_i 为一个集合：若存在状态 $p \in M_i$ 和转移 $\delta(p, \sigma') = (q, \triangleleft)$，则该集合包含 q。此外，设 $\mathfrak{r}$ 为与 $p \in M_i$ 关联的寄存器，我们为 $q \in M_i$ 关联一个寄存器 $\mathfrak{s}$，并令 $\mathfrak{s} \mapsto v\mathfrak{r}$，其中 v 是转移 $\delta(p, \sigma') = (q, \triangleleft)$ 的输出。

如果集合 $\tilde{M}_i$ 为空，那么我们移除对 $(\tilde{t}_i, \tilde{M}_i)$，因为这表明无法将 $\mathcal{T}$ 在 u' 上由 (t_i, M_i) 表示的从右到右运行扩展为 $\mathcal{T}$ 在 $u'\sigma$ 上的从右到右运行。

~容易看出，若 $p \in M_i$（其关联寄存器为 $\mathfrak{r}$）和 $q \in M_i$（其关联寄存器为 $\tilde{\mathfrak{r}}$），且 $\delta(p, \sigma') = (q, \triangleleft)$，其输出为 o；并且 π 是从根节点到叶节点的一条路径，其标签的形式为 (p', X)，且该路径位于 t_i 中；π' 是从根节点到叶节点的一条路径，其标签的形式为 (q', X')，且该路径位于 $\tilde{t}_i$ 中；此外，$\pi' \sqsubseteq \pi$ 和 $\delta(p', \sigma, L) = (q', \triangleright)$，其输出为 o'。那么，若 π 表示 T 在 u' 上从 p 到 q、输出为 $[\![r]\!][\![\pi_{\mathfrak{R}}]\!]$ 的从右到右运行，则 π' 表示 T 在 $u'\sigma$ 上从 p' 到 q'、输出为 $[\![\tilde{\mathfrak{r}}]\!][\![\pi'_{\mathfrak{R}}]\!] = o[\![r]\!][\![\pi_{\mathfrak{R}}]\!]o'$ 的从右到右运行。

现在我们检查是否存在 $(\tilde{t}_i,\ \tilde{M}_i)$ 和 $(\tilde{t}_j,\ \tilde{M}_j)$，需要将它们组合起来，以便（最终）表示恰当的从右到右运行。

若出现以下情况，则满足这一条件：$\tilde{t}_i^{\sim}$ 中有一个标签为 $(q,\ X)$ 的叶节点，并且在配置 $(q,\ |u'\sigma|)$ 中没有可执行的右移，且存在 $q \in M_j$。我们进行如下更改：

- 我们对 $(\tilde{t}_i,\ \tilde{M}_i)$ 进行更改，使得在 $\tilde{t}_i$ 中，标签为 $(q,\ X)$ 的叶节点以 $\tilde{t}_j$（的副本）作为子树（即，将该叶节点连接到 $\tilde{t}_j$（的副本）的根节点）。与 $\tilde{t}_j$（的副本）关联的寄存器内容（设为 $\mathfrak{r}$）将更新为 $\mathfrak{r} \mapsto \mathfrak{s}\mathfrak{r}$，其中 $\mathfrak{s}$ 是与 $q \in M_j$ 关联的寄存器~。由于我们已将从 q 开始的运行连接到在 q 结束的运行，因此从集合 M'_j 中删除条目 q。
- 如果移除某个状态后集合 $\tilde{M}_j$ 为空，则对 $(\tilde{t}_j,\ \tilde{M}_j)$ 就不再需要，可以将其删除。

这些组合步骤必须重复进行，直到再也没有可以合并的对为止。设 $(\hat{t}_1, \hat{M}_1) \dots, (\hat{t}_m, \hat{M}_m)$ 为经过这些步骤后剩下的对。不难看出，这些对表示了 $u'\sigma$ 上 T 的从右到右运行。此外请注意，T 在 $u'\sigma$ 上可能存在一些只访问最后一个字母 σ 的从右到右运行。这些运行并未由 $(\hat{t}_1, \hat{M}_1) \dots, (\hat{t}_m, \hat{M}_m)$ 表示，因为这些表示的是已经存在且可以延伸的运行

。我们已经说明了，对于前缀 σ ，如何得到只访问一个字母的从右到右运行的表示。

˜最后，可以将由 t 表示的运行与由这些对表示的运行组合起来。

如果在 $\tilde{t}$ 中存在一个标签为 $(q,\ X)$ 的叶节点，并且在配置 $(q,\ |u'\sigma|)$ 中没有可用的右移，且对于某个对 $(\bar{t},\ M)$，存在 $q \in M$，那么就属于这种情况。组合方式与上文所述相同。同样，必须重复这些组合步骤，直到无法再扩展 $\tilde{t}$ 。将最终结果记为 $\hat{}t$ 。不难看出，$\hat{}t$ 表示 $\mathcal{T}$ 在 $u'a$ 上的从左到右运行。

还剩下三个操作要做。

从所有树中删除表示成功运行前缀的部分。

我们区分两种情况：处理作为从左到右运行表示的树 $\hat{}t$ ，或处理某个对 $(\bar{t},\ M)$ 中作为从右到右运行表示的树 $\bar{t}$。

我们考虑前一种情况。我们构造 $\mathsf{del}_{\mathsf{success}}(\hat{}t)$。如果在此操作中移除了 $\hat{}t$ 的路径片段 π（也就是选择了一个新的根节点），那么很容易看出 $[\![\mathfrak{out}]\!][\![\pi_{\mathfrak{R}}]\!] \sqsubseteq \mathcal{T}(u)$ if $u \in$ 定义域$(\mathcal{T})$。我们相应地更新 $\mathfrak{out}$，令 $\mathfrak{out} \mapsto \mathfrak{out}\,\mathfrak{r}_1 \cdots \mathfrak{r}_j$，其中 $\mathfrak{r}_1 \cdots \mathfrak{r}_j$ 是与被移除路径 π 相关联的寄存器序列。

我们考虑后一种情况。对于某个 $(\bar{t}, M)$，我们也构造 $\mathsf{del}_{\mathsf{success}}(\bar{t})$。$\bar{t}$ 中被移除的部分表示一次部分运行，在此运行中，所有前瞻都已得到验证（也就是说，已经看到见证）。因此，不再需要前瞻信息，我们只需记住这次部分运行的输出。这次部分运行的输出由与被移除路径相关联的寄存器的输出串接而成。我们将被移除路径的输出推送到新的根节点。在下文中，我们会执行类似的操作，并给出更形式化一些的细节。

我们从所有树中移除表示失败运行的部分。这通过应用操作 $\mathsf{del}_{\mathsf{fail}}(\cdot)$ 来完成。

最后，为了获得适当的前瞻树，我们对每棵树应用操作 $\mathsf{merge}(\cdot)$。回顾一下，线性分支段通过将所有前瞻信息向上推送到最后一个节点来合并。对于关联的输出，我们也需要进行相同的操作。因此，设 $n_1 \cdots n_j$ 为被合并的节点序列，其中保留 n_1 和 n_j，并设 $\mathfrak{r}_1 \cdots \mathfrak{r}_j$ 为关联的寄存器序列，那么我们进行更新 $\mathfrak{r}_j \mapsto \mathfrak{r}_2\mathfrak{r}_3 \cdots \mathfrak{r}_{j-1}\mathfrak{r}_j$。

现在，我们已经定义了一个前瞻树 t'，以及一个由前瞻树和状态集组成的对的集合 $\{(t'_1, M'_1) \ldots, (t'_k, M_k)\}$，并定义了与每个树节点（分别为集合元素）关联的寄存器以及如何更新它们。由 $\mathcal{S}$ 执行的下一次转移会按照所述方式更新寄存器，而到达的状态存储 t' 和 $\{(t'_1, M'_1) \ldots, (t'_k, M_k)\}$。注意，正如构造中所述，不变式 C.5 和 C.6 仍然有效。

C.3 正确性证明

在证明关于该构造的主要技术引理之前，我们先说明，状态$\mathcal{S}$ 中需要存储的信息数量以及所用寄存器的数量都有界。

回顾一下，前瞻信息树集是由前瞻信息树组成的有限集合。状态中存储的信息是 $\mathrm{LAT} \times \mathcal{P}(\mathrm{LAT} \times \mathcal{P}(Q_{\mathcal{A}}))$ 的一个元素。设 $(t, \{(t_1, M_1) \ldots, (t_n, M_n)\})$ 为一个状态。存储与该状态相关信息所需的寄存器数量有界于 $b_1 + b_2$，其中 b_1 是树 $t, t_1, \ldots, t_n$ 中的节点数，而 b_2 是 $|M_1| + \cdots + |M_n|$。

▶ **引理 C.7.** 令 $u \in \Sigma^\omega$，对于 $i \geq 1$，令 u_i 表示 u 长度为 i的前缀。令 LR_i表示 T 在 u_i上的从左到右运行集合，令 RR_i 表示 T 在

u_i上的从右到右运行集合。

若 $u \in$ 定义域(T)，令 ρ_u 表示 T 在 u上的接受运行。序列 ρ_u 可以分解为若干因子，其中一些因子描述在 u_i 上的从左到右运行，另一些因子（可能不存在）描述在 u_i上的从右到右运行。令 $LR_i^{\text{acc}} \subseteq LR_i$ 和 $RR_i^{\text{acc}} \subseteq RR_i$ 分别表示这两组运行。前一组是单元素集。

令 c_i 为 S 读取 u_i后的配置（当前状态和寄存器内容），令s_i 为 S 的状态。以下性质成立。
1. 状态 s_i 表示 LR_i 和 RR_i的一个子集。**2.** 若 $\rho \in LR_i \cup RR_i$ 被表示，则对于其对应的寄存器序列 π，有$[\![\pi]\!] =$ 输出(ρ) 属于 c_i。**3.** 若 $u \in$ 定义域(T)，则 s_i 表示 LR_i^{acc}和 RR_i^{acc}的一个超集。**4.** 若 $u \in$ 定义域(T)，则 $[\![\text{out}]\!] \sqsubseteq T(u)$ 属于 c_i。

证明。 不难看出，条目 1 和 2 直接由断言 C.5 得出，而条目 3 和 4 直接由断言 C.6 得出。

我们通过对所考虑前缀的长度进行归纳法，证明断言 C.5 和 C.6。

对于归纳基础，所考虑的前缀是一个字母 $\sigma \in \Sigma$。我们已明确给出了在 $\sigma \in \Sigma$ 上到达的状态以及寄存器内容 $[\![\text{out}]\!]$。显然，断言 C.5 和 C.6 成立。
对于归纳步骤，设这些断言对长度为 $i \geq 1$ 的前缀成立，我们证明它们对长度为 $i+1$ 的前缀仍然成立。在上述构造中，我们已经证明，在构造后继状态和寄存器更新时，这些断言仍然有效。◀

▶ **引理 C.8。** 设 $u \in \Sigma^\omega$，若 $u \in$ 定义域(T)，则在 S 于 u 上的运行中，寄存器 out 会被无限次更新（通过追加一个非空的字）。

证明。 首先，我们注意到，如果 $u \in$ 定义域($\mathcal{T}$)，那么 $\mathcal{T}(u)$ 是无限的。
为了证明所需的结论，我们重新考察$\mathcal{S}$的构造。回想一下，每当操作 $\text{del}_{\text{success}}(\cdot)$ 从表示 $\mathcal{T}$ 在当前输入词前缀上的从左到右运行的树中移除一条成功的非空路径 π 时，就会更新 out 的寄存器内容（通过追加输出）。只有当$\mathcal{T}$ 在 u 上的接受运行得到表示时，这样一条成功路径才可能反复出现。引理 C.7 的第 3 项保证了这一点。追加的输出是与 π关联的寄存器序列的内容。根据引理 C.7 的第 4 项，我们知道通过这种方式构造出$\mathcal{T}(u)$。由于$\mathcal{T}(u)$是无限的，这些被移除的路径中必定有无穷多条，其对应的寄存器序列内容构成非空字。

剩下要证明的是，$\text{del}_{\text{success}}(\cdot)$会无穷多次移除一条成功路径。令 t_i 表示处理长度为 $i \geq 1$的中缀后，表示可能的从左到右运行的前瞻树。注意，从 t_i计算树 t_{i+1} 时，唯一能够改变节点中前瞻信息数量的操作是 $\text{merge}(\cdot)$。但根据定义，该操作绝不会改变根节点及其子节点中存储的前瞻信息数量。此外，注意，属于接受运行的所有前瞻信息最终都会成功（这是运行被称为接受运行的定义）。因此，由于接受运行得到了表示，对根节点以及与接受运行对应的子节点中的前瞻信息进行更新，最终会使这些节点成功。随后，它们会被操作 $\text{del}_{\text{success}}(\cdot)$移除。◀

接下来需要定义$\mathcal{S}$的 Büchi 接受条件。从本质上说，$\mathcal{S}$ 会跟踪$\mathcal{T}$ 在输入上的运行。每当寄存器 out 更新时，$\mathcal{S}$ 接受运行中的另一个片段就已确定。此运行片段随后不再存储在

$\mathcal{S}$的状态空间中。我们需要修改 SST，使得每当接受运行中的一个运行片段被移除时，当且仅当该运行片段经过了一个终态，SST 才访问终态。请注意，无法直接从存储的抽象中恢复这一信息，但很容易将其添加到抽象中。

▶ **引理 C.9**。 设 $u \in \Sigma^\omega$。BSST S 在 u 上存在接受运行，当且仅当 $u \in$ 定义域(T)。

证明。 这直接由上述 Büchi 接受条件的定义得出，该定义模拟了双向转换器的 Büchi 接受条件。◀

▶**引理 C.10**。 BSST S 是有界复制的。

证明。设 S $= (\Sigma, \Gamma, Q_{\mathcal{S}}, q_0^{\mathcal{S}}, \delta_{\mathcal{S}}, \mathfrak{R}, \mathfrak{out}, \lambda')$。我们的目标是证明 S 是 1-有界的，即对于所有满足 $\lambda_i'^u \circ \cdots \circ \lambda_j'^u$ 有定义的 $u \in \Sigma^\omega$ 和 $i \leq j$，该替换是 1- 有界的。如果对于所有 $\mathfrak{r}$、 $\mathfrak{s} \in \mathfrak{R}$， $\mathfrak{r}$ 在 $\tau(\mathfrak{s})$ 中至多出现一次，则替换 τ是 1-有界的。我们不提供形式化证明，而是对这一结论给出直观的说明。

首先，我们注意到，所构造的 BSST 之所以不是无复制的，唯一原因是：对于 M 中与形如 (t, M) 的对相关联、用于表示前缀上的从右到右运行的元素，进行了寄存器更新。

回想一下，来自一对 (t, M) 的树 t 存储了有关向右移动的从右到右运行的转移的信息，而集合 M 仅存储可以进行从右到右运行的状态。还要回想一下，向左移动的转移不使用前瞻。这就是我们没有树结构来表示后续一系列向左移动的原因，因为无需验证前瞻。因此，无需保留中间信息。

因此，回想一下，如果 $\mathcal{S}$ 在输入 $u \in \Sigma^\omega$ 的前缀 $u[1{:}i] \in \Sigma^*$ 上到达状态 s_i，且该前缀包含一对 (t, M)，使得 $p \in M$，那么这意味着该转导器在 $u[1{:}i]$ 上存在一次从 p开始的从右到右运行。

如果在下一步中，$\mathcal{S}$ 在包含一对 $(t', M\prime)$ 的前缀 $u[1{:}i{+}1] \in \Sigma^*$ 上到达状态 s_{i+1}，且该对是通过更新对 (t, M) 得到的，并且存在 $p_1, p_2 \in M\prime$，那么，由于存在转移 $(p_1, \sigma) \mapsto (p, \triangleleft)$ 和 $(p_2, \sigma) \mapsto (p, \triangleleft)$，这意味着转导器在 $u[1{:}i{+}1]$ 上存在一次从右到右运行，分别始于 p_1 和 p_2。

回想一下，用于存储与 $p \in M$ 相关的输出的寄存器对应于对 (t, M)，记作 $\mathfrak{out}_p^{(t,M)}$；它用于定义存储与 $p_1, p_2 \in M\prime$相关输出的寄存器，对应于对 (t', M')，记作 $\mathfrak{out}^{(t',M')p_1}$ 和 $\mathfrak{out}^{(t',M')p_2}$。

证明每个替换都是 1-有界的关键在于，最终，转导器在 $u[1{:}i{+}1]$ 上的从右到右运行中，要么只有从或 从 p_2 开始的运行会影响转导器的接受运行，因为该转导器是确定性的。因此，接下来，寄存器 $\mathfrak{out}^{(t',M')p_1}$， $\mathfrak{out}^{(t',M')p_2}$ 永远不会对同一个寄存器产生贡献。◀

在证明定理 5.2 之前，我们还需要最后一个辅助引理。也就是说，我们不需要 Büchi 接受条件。

▶**引理 C.11.** 对于每个BSST T_1，都存在一个等价的BSST T_2，使得它的所有 s 状态都是终态。

证明。 BSST $\mathcal{T}_1$ 可以很容易地进行修改，使其所有状态都是终态。因此，在$\mathcal{T}_2$, 中，每当$\mathcal{T}_1$ 要更新指定输出寄存器 $\mathfrak{out}$时，我们需要引入一个新寄存器，例如 $\mathfrak{r}$，并将输出追加到 $\mathfrak{r}$ 中。每当$\mathcal{T}_1$ 要访问一个终态时，就将 $\mathfrak{r}$ 中的寄存器内容追加到 $\mathfrak{out}$ 中，并清空 $\mathfrak{r}$。

要看出 $[\![\mathcal{T}_1]\!] = [\![\mathcal{T}_2]\!]$，只需认识到以下一点：如果输入词 $u \in \Sigma^\omega$ 属于$\mathcal{T}_1$]]的定义域，那么在$\mathcal{T}_1$ 对 u的无限运行中

- $\mathcal{T}_1$ 会无限次访问终态，并且
- out 中内容的长度趋于无穷。

因此，在$\mathcal{T}_2$ 对 u 的无限运行中，寄存器 out 会被无限次更新，且内容长度趋于无穷。如果输入词 $u \in \Sigma^\omega$ 不属于$\mathcal{T}_1$]]的定义域，那么运行要么只有限次访问终态，要么 out中内容的长度不会趋于无穷。对于第一种情况，寄存器 out 在$\mathcal{T}_2$ 对 u的运行中只被更新有限次，因此 out 中内容的长度不会趋于无穷（也就是说，该运行是拒绝的）。对于第二种情况， out 中内容的长度无论如何都不会趋于无穷。◀

最后，我们可以给出前瞻消除定理的证明了。

定理 5.2 的证明。由引理 C.7 的第 4 项以及引理 C.8 和 C.9 可直接推出：$[\![\mathcal{T}]\!] = [\![\mathcal{S}]\!]$，也就是说，带有限前瞻的确定性双向转换器与确定性 Büchi SST 识别相同的函数。

如引理 C.10 所示，确定性 SST 具有有界复制，因此可以按照定理 3.7] 中所示的方式构造一个等价的确定性双向转换器（回想一下，我们在定理 2.9 中重述了这一点）。正如注 2.10 中已经指出的，被引用的结果假设有界复制 SST 只有终态（这一点可由引理 C.10 保证）。根据构造，所得确定性双向转换器的所有状态都是终态。◀

C.4 带安全约束的扩展

我们为带有限前瞻的确定性双向转换器加入安全约束。它是一个 2-dT^{FLA} $\mathcal{T}$，并附加一个映射 S，将每个转移 t 映射到有限词的一个正则语言 $S(t)$。如果 $\mathcal{T}$ 在字 $u \in \Sigma^\omega$的输入位置 i 触发转移 t，那么它还必须确保对所有 $j \geq i$， $u[i{:}j] \in S(t)$ 成立；否则运行将终止。我们证明，尽管安全约束是关于无限后缀的性质，但它们并不会为 2-dT^{FLA}增加额外的计算能力。

带有限前瞻和安全约束的确定性双向转换器（2-dT$^{\mathsf{FLA}}_{\mathsf{S}}$）是一个元组 $\mathrm{T}' = (\Sigma, \Gamma, Q, q_0, F, \delta, \lambda, S)$，其中 $\mathrm{T} = (\Sigma, \Gamma, Q, q_0, F, \delta, \lambda)$ 是一个 2-dT^{FLA}，称为 T′的底层 2-dT^{FLA}，并且 $S : \delta \to \mathrm{R}^*(\Sigma)$ 将任意转移映射到有限词的一个正则语言，称为安全约束。其语义定义如下：给定一个输入$u \in \Sigma^\omega$，若 (i) T 通过转移 t从 (q, i) 移至 (q, j)，且对所有 $k \geq i$，(ii) $u[i{:}k] \in S(t)$，则 T′ 按照转移 t从配置 (q, i) 移至配置 (q, j)。显然，由于这一额外约束，T 和 T′ 分别识别的函数$f_{\mathcal{T}}$和 $f_{\mathcal{T}'}$具有如下性质：$\mathrm{dom}(f_{\mathcal{T}'}) \subseteq \mathrm{dom}(f_{\mathcal{T}})$，且它们在 $\mathrm{dom}(f_{\mathcal{T}'})$ 上一致。只要 $t = (p, \sigma, L, q) \in \delta$且 $S(t) = C$，转移便记作 $p \xrightarrow{\sigma,\ L|w\ C} q$，。另外请注意，如果对所有转移 t， $S(t) = \Sigma^*$，那么 T′ 显然等价于其底层 2-dT^{FLA} T。

▶**引理 C.12.** 一个函数可由 2-dT$^{\mathsf{FLA}}_{\mathsf{S}}$计算，当且仅当它可由 2-dT计算。

证明。 该证明是对定理 5.2 证明的稍作扩展。因此，我们只描述其中的改动。我们将安全约束的验证整合到从转导器到等价 SST 的转换过程中。因此，我们扩展前瞻树的节点，使其也能存储有关安全约束的信息。假设安全语言

由确定性有限自动机给出，例如，安全语言 C_i 由确定有限自动机 $\mathcal{C}_i$给出。每当根据带有安全约束 $S(t) = C_i$ （其中 i）的转移 t 创建前瞻树中的新节点时，该节点就会存储 $\mathcal{C}_i$ 的初始状态。每当根据字母 $\sigma \in \Sigma$ （记作 up($\cdot$ σ）更新前瞻树时，所有存储的安全自动机状态也会按照 σ 更新。如果某个状态表明安全约束已被违反，则该节点被视为失败节点，包含该失败节点的分支可以从树中删除。合并树的节点时，也会合并其中的安全信息。◀

C.5 扩展：带有限环视

我们在此考虑能够从当前位置检测有限前缀（前瞻），以及检测截至当前位置的前缀（后行断言）的双向转换器。二者结合为一种环视，它是标记字的正则语言，其中只有一个位置带有标记，对应于当前位置。形式化地，对于字母表 Σ，我们令 $\Sigma = \{\dot{\sigma} \mid \sigma \in \Sigma\}$。带标记字 是 $\Sigma^*\Sigma\Sigma^*$ 中的一个有限词。有限环视是标记字的正则语言。

带有限环视的确定性双向转换器是如下形式的元组 T $=(\Sigma, \Gamma, Q, q_0, F, \delta, \lambda)$，其定义方式与 2-dT^{FLA} 相同，但 δ 的类型为 $Q \times (\Sigma \uplus \{\vdash\}) \times \mathrm{LAR}^*(\Sigma) \rightharpoonup Q \times \{\triangleright, \triangleleft\}$，其中 $\mathrm{LAR}^*(\Sigma)$ 是 Σ 上有限环视的集合。与 2-dT^{FLA} 一样，要求满足：（i） dom(δ) 有限；（ii）同一源状态和同一输入符号上的转移中出现的所有环视两两不相交。与 2-dT^{FLA} 一样，为确保确定性，当多个环视均成功时，只选取见证长度最短的环视所对应的转移（由不相交性可知该转移唯一）。如果没有环视成功，则计算失败。更准确地说，在输入词u 上的配置 (q, i) 中，若 $u[i] = \sigma$，则运行沿着转移 $\delta(q, \sigma, L)$ 进行，其中 L 在集合 $\{j \mid \exists L' \cdot \delta(q, \sigma, L')$ 有定义且 $u[{:}i{-}1]\dot{\sigma}u[i{+}1{:}j] \in L'\}$ 中具有最小的 j。

同样，我们进一步将该模型扩展到带有限环视和安全约束的确定性双向转换器（2-dT$^{\mathsf{FLAR}}_{\mathsf{S}}$）。现在，我们的目标是证明 2-dT 与 2-dT$^{\mathsf{FLAR}}_{\mathsf{S}}$具有同等的表达能力。

►**引理 C.13**。 一个函数可由 2-dT$^{\mathsf{FLAR}}_{\mathsf{S}}$计算，当且仅当它可由 2-dT计算。

证明。 直观而言，任意环视都可以分解为有限个后行断言和前瞻的并。我们利用这一观察证明，任意 $\mathcal{T}$ $\in$ 2-dT$^{\mathsf{FLAR}}_{\mathsf{S}}$都可以分解为：

- 一个 1-dT，用后行断言信息（环视自动机在迄今为止的前缀上到达的状态集合）标注输入；一个 2-dT$^{\mathsf{FLA}}_{\mathsf{S}}$，在带标注的输入上模拟 $\mathcal{T}$
-

更形式化地说，设 $\mathcal{T}$ $\in$ 2-dT$^{\mathsf{FLAR}}_{\mathsf{S}}$的输入字母表为 Σ。假设任意环视 L 都由一个 DFA$\mathcal{A}_L$ 给出，其状态集合为 Q_L ，终态集合为 F_L。给定一个状态 $q \in Q_L$，我们用 Right^q_L 表示 Σ^*中由 A_L 接受的字集合，其中初始状态被替换为 q。对称地，令 Left^q_L 表示 Σ^*中由 A_L 接受的字集合，其中终态集合被替换为 $\{q\}$。显然：

$$L = \bigcup_{q \in Q_L, \sigma \in \Sigma} \bigcup_{p \in Q_L \cdot \delta_L(q, \dot{\sigma}) = p} \mathsf{Left}^q_L \dot{\sigma} \mathsf{Right}^p_L$$

现在我们描述第一个函数 t，它用后行断言信息标记输入。令$\mathcal{L}$ 为 T 的转移上出现的所有环视的集合。令 $Q = \biguplus_{L \in \mathcal{L}} Q_L$。该

函数定义为 t：$\Sigma^\omega \to (\Sigma \times 2^Q)^\omega$：给定一个字 $u \in \Sigma^\omega$ 和一个位置i，将 $u[i]$ 替换为 $(u[i], P)$，其中 P 是环视自动机在输入 $u[:i-1]$ 上到达的所有状态。特别地，第一个字母会被标记为环视自动机的初始状态集合。

第二个函数将与 t 复合，由一个 2-dT$^{\mathsf{FLA}}_{\mathsf{S}}$ M 实现，它通过对输入进行注释来模拟$\mathcal{T}$，但只使用前瞻。具体而言，考虑$\mathcal{T}$的一条形如 $x \xrightarrow{\sigma,\ L|v\ L_{safe}} y,(y,d)$ 的转移，其中 x, y 是$\mathcal{T}$的状态，$\sigma \in \Sigma$, L 是环视，L_{safe} 是安全约束，$v \in \Gamma^*$ 是输出词，d 是一个方向。对于任意 $P \subseteq Q$，我们在 M 中创建转移 $x \xrightarrow{(\sigma\ P,\ \ q\ L_{safe}|v\ ,)\mathsf{Right},(y,d)} y$，其中对所有满足存在 $p \in P$ 使得 $\delta_L(\dot{\sigma}, p) = q$ 的 $q \in Q_L$ 均成立。我们还必须确保不相交性。考虑 M 中两条形如

$$p \xrightarrow{(\sigma,P),\mathsf{Right}^{q_1}|v_1,L_{\mathrm{sf}_1}} p_1$$
$$p \xrightarrow{(\sigma,P),\mathsf{Right}^{q_2}|v_2,L_{\mathrm{sf}_2}} p_2$$

可能存在 Right^{q_1} 和 Right^{q_2} 不相交的情况。在这种情况下，我们按如下方式替换这两个转移（在 q_1 和 q_2 之间任取一个序）：

$$p \xrightarrow{(\sigma,P),\mathsf{Right}^{q_1}\setminus\mathsf{Right}^{q_2}|v_1,L_{\mathrm{sf}_1}} p_1$$
$$p \xrightarrow{(\sigma,P),\mathsf{Right}^{q_2}|v_2,L_{\mathrm{sf}_2}} p_2$$

我们对所有转移反复应用这一重写，直到满足不相交性条件。其正确性理由如下。假设 M 在一个由函数 t 标注的字上运行，那么由于 $\mathcal{T}$ 的环视彼此不相交，当 q_1 和 q_2 是两个不同环视自动机的状态时，只有一个前瞻 Right^{q_1} 或 Right^{q_2} 能够成功，否则就会与不相交性矛盾。如果 q_1 和 q_2 是同一个环视自动机的状态，那么由于该环视自动机的确定性，两个前瞻中至多有一个能够成功（否则，在同一个带标记字上会存在该环视自动机的两个不同运行：一个经过 q_1，另一个经过 q_2）。

最后，注意到标记函数 t 可以由一个所有状态都是终态的 1-dT 实现。因此，我们证明了 $\mathcal{T}$ 可以定义为一个 2-dT$^{\mathsf{FLA}}_{\mathsf{S}}$ 与这样一个 1-dT 的复合。根据引理 C.12，2-dT$^{\mathsf{FLA}}_{\mathsf{S}}$= 2-dT，并且根据定理 3.11，2-dT 在与所有状态都是终态的 1-dT 进行前复合下保持封闭，因此我们得到所需结果。◀

C.6 复合封闭性：定理 3.1 的证明

定理 3.1 的证明。 设 $\mathcal{T}_1$,$\mathcal{T}_2$ 表示两个 2-dT。我们的目标是构造一个带环视的确定性双向转换器，使其识别 $\mathcal{T}_2 \circ \mathcal{T}_1$。然后，我们使用引理 C.13 得到一个确定性双向转换器。

其思路是进行乘积构造。$\mathcal{T}$ 的行为由$\mathcal{T}_1$ 和$\mathcal{T}_2$的组合行为决定。在每一步中，给定一个输入字母 $\sigma \in \Sigma$，$\mathcal{T}$ 先模拟一个匹配的转移 $t_1 \in \delta_1$，然后模拟$\mathcal{T}_2$ 在产出$\lambda_1(t_1) \in \Gamma^*$上的一次端到端运行 ρ_2，接着产生相应的输出 out(ρ_2)。

我们假设$\mathcal{T}$处于这样一个状态：$\mathcal{T}_1$ 的分量处于状态 $p \in Q_1$，而$\mathcal{T}_2$ 的分量处于状态 $q \in Q_2$。

如果$\mathcal{T}_2$ 正在向前移动，那么我们就需要推进$\mathcal{T}_1$的运行。我们可以利用$\mathcal{T}_1$ 的确定性，根据 t_1 更新$\mathcal{T}_1$ 的分量：$= (p,\ \sigma) \mapsto (p',\ d) \in \delta_1$，并模拟$\mathcal{T}_2$ 从左侧进入 $\lambda_1(t_1) \in \Sigma^*$。因此，$\mathcal{T}$ 包含一个转移，其

形式为$((q,\ p)\ \sigma) \mapsto ((q',\ p')\ d)$，其中 q' 是从 q左侧开始、在输出$\lambda_1(t_1) \in \Sigma^*$ 上的端到端运行 ρ_2 的目标状态。该转移的输出为 $\mathrm{out}(\rho_2)$。

如果$\mathcal{T}_2$ 正在向后移动，那么我们需要在$\mathcal{T}_1$的运行中回退。然而，$\mathcal{T}_1$ 并非余确定性的，因此此时我们需要使用环视特性。

假设对于每个转移 $t := (p', \sigma) \mapsto (p, d) \in \delta_1$，我们都有一个前瞻信息树集 L_t：如果配置 (p, i) 可通过应用转移 t 从 (p', i') 到达，则该前瞻信息树集在$\mathcal{T}_1$ 的配置 (p, i) 中成立。下面我们将说明如何构造这样的前瞻信息树集。

因此，如果之前使用的转移是某个 $t_1 :=(p', \sigma) \mapsto (p, d) \in \delta_1$，则前瞻信息树集 L_{t_1} 处于启用状态。2-dT $\mathcal{T}$模拟$\mathcal{T}_2$ 从右侧进入 $\lambda_1(t_1) \in \Gamma^*$。因此，$\mathcal{T}$包含一个形如 $((q, p)\ \sigma) \mapsto ((q', p')\ d)$ 的转移，其中 q' 是从右侧开始、在 q中的产出 $\lambda_1(t_1) \in \Gamma^*$ 上进行的端到端运行 ρ_2 的目标状态。该转移的输出为 $\mathrm{out}(\rho_2)$。

关于$\mathcal{T}$的接受分量，只需结合$\mathcal{T}_1$ 和$\mathcal{T}_2$的 Büchi 条件即可。

现在我们解释如何使用确定性双向自动机获得所需的环视。随后，我们利用这样一个事实：确定性双向自动机与确定性单向自动机具有相同的表达能力（即，这两种模型都识别正则语言），并且这种转换是有效的 [26]。

首先回顾一下，环视是有限带标记位置的词构成的正则语言；当在带环视的确定性双向转导器的计算中评估环视时，带标记的字母位于双向转导器当前读取的位置。因此，如果双向转导器$\mathcal{T}_1$ 已经到达某个配置（p，i），并且我们想要使用一个双向自动机在输入上确定$\mathcal{T}_1$ 的前驱配置，其中位置 i 上是带标记的字母，那么该自动机只需先移动到字的开头，然后模拟$\mathcal{T}_1$ 的运行，直到它以状态 p 到达带标记的字母处，并记住为到达那里所采取的转移。

注意，这样的双向自动机只需访问双向转换器迄今访问过的输入有限前缀。因此，很明显，我们确实定义了一个有限环视。此外，容易看出，当转导器$\mathcal{T}_1$ 处于位置 i 上的状态 p 时（为了确定(p, i)的后继配置），需要验证的前瞻两两不相交；否则，就意味着两条不同的运行会导向同一个配置(p, i)，这与$\mathcal{T}_1$ 是确定性的相矛盾。◀

D 逻辑-转换器对应关系：定理 4.8 的证明

D.1 证明的结构

证明经过若干中间模型。这里我们概述用于证明逻辑刻画的（有效）模型转换链。我们从一个 MSOT$_\mathcal{G}$ $\mathcal{T}_0$开始。

1. 首先，根据引理 6.3，将 T_0 转换为跳跃式单子二阶逻辑$_\mathcal{G}$-转导器 T_1, 。2. 根据引理 D.1，将 T_1 转换为步行式单子二阶逻辑$_\mathcal{G}$-转导器 T_2, 。3. 将 T_2 转换为复合 $\mathrm{T}_3 \circ \mathrm{U}_1$，其中 T_3 是一个 2-dT$^{\mathsf{FLAR}}_{\mathsf{S}}$（扩展了环视和安全约束的 2-dT，定义见 C.5节），U_1 是一个 1-dT，且其所有状态都是终态（见引理 D.2）。4. 接着，将 T_3 转换为复合 $\mathrm{T}_4 \circ \mathrm{U}_2$，其中 T_4 是一个 2-dT$^{\mathsf{FLA}}_{\mathsf{S}}$，$\mathrm{U}_2$ 是一个 1-dT，且其所有状态都是终态（见引理 C.13）。这展示了如何将前瞻转换为前瞻。

5. 根据引理 C.12，将 T_4 转换为一个 2-dT T_5, ；其证明是对定理 5.2 的前瞻移除构造进行安全约束扩展。6. 由此可得，T_0 等价于 $T_5 \circ U_2 \circ U_1$，其中 T_4 是一个 2-dT，而 U_1,和 U_2 是 1-dT，且它们的所有状态都是终态。文献 [9]中的定理 3.11 表明，确定性正则函数在与所有状态均为终态的 1-dT 所定义的函数进行前复合时有效封闭（实际上，结果稍微更一般）。因此，T_0 等价于一个 2-dT T_6。7. 最后，根据引理 6.1，将 T_6 转换为一个与 T_0, 等价的单子二阶逻辑转导器 T_7 ，从而完成这一系列变换。

D.2 引理 6.1 的证明

证明。 设 $\mathcal{T} = (\Sigma, \Gamma, Q, q_0, F, \delta, \lambda)$ 是一个识别 f 的 2-dT。根据注 2.10，我们可以假设 $Q = F$。不失一般性，我们还假设，对于某个 $n \in \mathbb{N}$，有 $Q = \{1, \ldots, n\}$，且 $q_0 = 1$。我们还假设，对于所有 $q \in Q$ 和 $\sigma \in \Sigma$，$|\lambda(q, \sigma)| \leq 1$，并且 $|\lambda(q, \vdash)| = 0$。如果 $\mathcal{T}$一次产生多个符号，那么可以通过一系列 U 形转向分多步产生它们。同样，可以通过强制转导器进行一些 U 形转向来满足第二个假设，并使其在第二个符号上产生原本应该在 $\vdash$ 上产生的内容。

我们定义一个 MSOT$_g$ $\mathcal{M} = (k, (\phi^{c,\ d}_{S})_{c,\ d}, (\phi_\gamma)_\gamma, \phi^{c_0}_{\mathrm{fst}})$，使得 $[\![M]\!] = f$。它恰好使用输入的n 个副本，即 $k = n$。直观地说，每个副本对应于$\mathcal{T}$的一个状态。从副本 p 到副本 q 的后继关系，总结了从状态 p 开始的非输出转移序列，以及随后恰好进行一次转移到 q 并产生某个符号的过程。标签谓词正是通过考察这最后一个符号来定义的。

形式化地，设 $q = \delta(\vdash, q_0)$。于是，我们令 $c_0 = q$ 和 $\phi^{c_0}\mathrm{fst}(x) = \mathrm{first}(x)$。然后，给定两个状态 p、q，我们令 $\mathrm{Run}^{p,q}(x, y)$ 表示如下二元谓词：存在一条 T 的有限运行，它从位置 x 的状态 p 开始（读取 x的标签），在读取位置 y后于状态 q 结束，并且除最后一次转移外不产生任何输出符号。根据有限词的正则语言与 MSO 之间的对应关系，可以标准地证明，该谓词可由有限词上的 MSO 定义。然后，我们令 $\Phi^{p,\ q}_{S}(x, y) = \exists \mathsf{g} \cdot x \leq \mathsf{g} \wedge y \leq \mathsf{g} \wedge \mathrm{Run}^{p,q,\mathsf{g}}(x, y)$，其中 $\mathrm{Run}^{p,q,\mathsf{g}}(x, y)$ 正是 $\mathrm{Run}^{p,q}(x, y)$，只是所有量词都由 g 守卫（因此，$\mathrm{Run}^{p,q,\mathsf{g}}(x, y)$ 是在输入的有限前缀上解释的）。

最后，标签谓词的定义也类似：$\phi^p_\gamma(x)$ 表示存在一条有限运行，从处于状态 q_0 的初始位置到达位置 x 和状态 q，且最后一次转移产生符号 γ。与之前一样，这可以用一个受限公式表示。◀

D.3 引理 6.2 的证明

给定一个单子二阶逻辑$_g$-转导器 $\mathcal{T}$，我们的目标是构造一个与之等价的带环视和安全约束的确定性双向转导器 $\mathcal{T}'$（这是确定性双向转换器（2-dT$^{\mathsf{FLAR}}_{\mathsf{S}}$，定义见附录 C）的扩展），并由引理 C.13 得出结论，该引理指出此扩展与 2-dT等价。与 2-dT$^{\mathsf{FLAR}}_{\mathsf{S}}$相比，两者有两个主要区别：$(i)$ 跳跃型单子二阶逻辑$_g$-转导器会跳跃，而 2-dT$^{\mathsf{FLAR}}_{\mathsf{S}}$会移动；$(ii)$ 跳跃型单子二阶逻辑$_g$-转导器中的安全约束是关于所有从第一个输入位置开始且包含当前位置的有限前缀的性质，而 2-dT$^{\mathsf{FLAR}}_{\mathsf{S}}$中的安全约束则是关于所有从当前位置开始的有限前缀的性质。

我们将证明分解为两个引理，分别处理这两个难点。下文的引理 D.1 指出，任何跳跃型单子二阶逻辑$_g$-转导器都等价于一个移动型 单子二阶逻辑$_g$-转导器，即满足以下条件的单子二阶逻辑$_g$-转导器：对于其转移中出现的任意跳跃公式 $\phi_{\mathrm{mv}}(x,y)$，$\phi_{\mathrm{mv}}(x,y) \Rightarrow S(x,y) \vee S(y,x)$ 成立。接着，下文的引理 D.2 指出，任何移动型单子二阶逻辑$_g$-转导器都定义一个确定性正则函数。结合这两个引理即可证明引理 6.2。

▶ **引理 D.1**。任意跳跃式单子二阶逻辑$_g$-转导器都等价于一个行走式单子二阶逻辑$_g$-转导器。

证明。设 T 是一个跳跃式 MSO$_g$-转导器。将 T 转换为行走式转导器 T′ 的方法是标准的，思路与 [21, 13] 相同。根据 Büchi–Elgot–Trakhtenbrot 定理，T 的任意跳跃公式 $\phi_{\mathrm{mv}}(x,y)$ 都可转换为一个有限词上的确定有限自动机 $A_{\phi_{\mathrm{mv}}}$，其带有两个指定位置 x 和 y；也就是说，该自动机处理字母表 $\Sigma \times 2^{\{x,y\}}$ 上的词，其中 x 和 y 各恰好出现一次。特别地，我们有 $L(A_{\phi_{\mathrm{mv}}}) = \{(\sigma_1, s_1) \dots (\sigma_n, s_n) \mid \exists 1 \le i,j \le n \cdot\ \sigma_1 \dots \sigma_n \models \phi_{\mathrm{mv}}(i,j) \text{ 和 } \forall 1 \le k \le n,\ i = k \leftrightarrow x \in s_k, j = k \leftrightarrow y \in s_k\}$。当 T 触发一个转移时

$$p \xrightarrow{\phi_{\mathrm{la}}(x) \mid w, \phi_{\mathrm{mv}}(x,y), \phi_{\mathrm{sf}}(x)} q$$

相反，T′ 会转到一个子转换器 T$_{\phi_{\mathrm{mv}}}$，该转换器模拟确定有限自动机 $A_{\phi_{\mathrm{mv}}}$ 的转移（并始终产生 ε）。我们在 $x < y$ 的情况下解释对 $A_{\phi_{\mathrm{mv}}}$ 的模拟。T$_{\phi_{\mathrm{mv}}}$ 从当前位置 x 开始，并使用一些环视（实际上是后行断言）来确定 $A_{\phi_{\mathrm{mv}}}$ 在目前为止的前缀上到达的状态，其最后一个位置由 x 指出；也就是说，$A_{\phi_{\mathrm{mv}}}$ 在输入词 $(\sigma_1, \varnothing) \dots (\sigma_{k-1}, \varnothing)(\sigma_k, \{x\})$ 上到达的状态，其中 $\sigma_1 \dots \sigma_k$ 是截至读头位置的输入前缀。随后，它从环视确定的状态开始模拟 $A_{\phi_{\mathrm{mv}}}$。对于之后的任意转移，T$_{\phi_{\mathrm{mv}}}$ 都会使用有限前瞻来判断当前位置（称为 y）是否满足 $\phi_{\mathrm{mv}}(x,y)$，或者是否应继续模拟 $A_{\phi_{\mathrm{mv}}}$。为此，它会通过前瞻检查是否存在有限字 $v = (\sigma_1, \{y\})(\sigma_2, \varnothing) \dots (\sigma_n, \varnothing)$，使得 $\sigma_1 \dots \sigma_n$ 是当前无限后缀（即从当前位置开始的后缀）的前缀，并且 $A_{\phi_{\mathrm{mv}}}$ 的运行可以在 v 上延续为一个接受运行。如果有限前瞻成功，T$_{\phi_{\mathrm{mv}}}$ 便知道 $\phi_{\mathrm{mv}}(x,y)$ 成立，且当前位置就是 y，于是将控制权交还给 T′。注意，T$_{\phi_{mv}}$ 无需检查 y 是否唯一。事实上，跳转式单子二阶逻辑$_g$-转导器的定义中所作的假设保证：如果某一时刻有两个位置 y 满足 $\phi_{\mathrm{mv}}(x,y)$，那么 T 在该输入词上就不可能有无限运行，因此 T′ 也不可能有无限运行。◀

▶**引理 D.2.** 任意行走型 单子二阶逻辑$_g$-转导器 T 都定义一个确定性正则函数。

证明。设 $\mathcal{T}$ 是一个跳跃型单子二阶逻辑$_g$-转导器。我们将 $\mathcal{T}$ 表示为两个函数的复合：一个是由 1-dT $\mathcal{T}_{\mathrm{sf}}$ 定义的确定性正则函数 f_{sf}，它为输入添加用于检查安全约束的信息；另一个是行走型单子二阶逻辑$_g$-转导器 $\mathcal{T}'$，其安全约束是受限于大于 x 的位置的 MSO 公式 $\psi_{\mathrm{sf}}(x)$。因此，除了上界保护条件之外，量化变量的取值范围也仅限于大于x的位置。注意，$\mathcal{T}'$ 不过是一个 2-dT$_{\mathsf{S}}^{\mathsf{FLAR}}$，其环视和安全约束由 MSO$_g$ 公式定义。

现在我们继续定义 f_{sf} 并构造$\mathcal{T}'$。直观来说，所有公式 $\phi_{\mathrm{sf}}(x)$都会被转换为确定有限自动机，而函数 f_{sf} 会用这些确定有限自动机在当前前缀上到达的某些状态来标注输入。随后便可利用这些信息，构造只涉及从当前位置开始的前缀的安全约束。

更准确地说，出现在$\mathcal{T}$某个转移上的任意公式 $\phi_{\mathrm{sf}}(x)$都会被转换为一个作用于有限带标记位置的词上的确定有限自动机 $A_{\phi_{\mathrm{sf}}}$。我们假设所有确定有限自动机的状态集两两不相交，并令 Q 为它们的并集。于是，函数 $f_{\mathrm{sf}}\colon \Sigma^\omega \to (\Sigma \times 2^Q)^\omega$ 按如下方式标注输入词：对于输入 $u \in \Sigma^\omega$，对任意位置 i，$f_{\mathrm{sf}}(u)[i] = (u[i], P)$，其中 $P \subseteq Q$是所有确定有限自动机在前缀 $u[:i]$ 上到达的状态集，该前缀中标记的位置为 $u[i]$。因此，$f_{\mathrm{sf}}(u)[i]$ 包含确定有限自动机在“选定”位置 i之后立即到达的状态。注意，每个集合 P 中每个确定有限自动机 $A_{\phi_{\mathrm{sf}}}$至多有一个状态，因为它们是确定性的。显然，函数 f_{sf} 可由一个 1-dT实现，该装置并行模拟所有确定有限自动机。

然后，$\mathcal{T}'$ 在字母表 $\Sigma \times 2^Q$ 上的输入词上运行，并利用 f_{sf} 提供的信息如下：对于 $\mathcal{T}$ 的每个转移 $t = p \xrightarrow{\phi_{\mathrm{la}}(x) \mid w,\ \phi_{\mathrm{mv}}(x,\ y)\ \phi(x)} q$ 以及每个子集 $P \subseteq Q$，我们创建 $\mathcal{T}'$ 的转移 t_P

$$t_P = p \xrightarrow{\phi^P_{\mathrm{la}}(x) \mid w, \phi^P_{\mathrm{mv}}(x,y), \phi^P_{\mathrm{sf}}(x)} q$$

其中：

- $\phi^P_{\mathrm{la}}(x)$ 是 $\bigvee_{\sigma \in \Sigma}(\sigma,\ P)(x)$ 与公式 $\phi_{\mathrm{la}}(x)$ 的合取（用于检查第二个输入标签分量是 P），其中对于所有 $\sigma \in \Sigma$，谓词 $\sigma(y)$ 的任何出现都会被替换为 $\bigvee_{R \subseteq Q}(a,\ R)(y)$。$\phi^P_{\mathrm{mv}}(x,\ y)$ 是 $\phi_{\mathrm{mv}}(x,\ y)$，其中标签谓词按照之前的方式进行修改。$\phi^P_{\mathrm{sf}}(x)$ 是 $\bot$，如果 P 包含 $A_{\phi_{\mathrm{sf}}}$ 的多个状态或不包含任何状态。如果 P 恰好包含 $A_{\phi_{\mathrm{sf}}}$ 的一个状态 s，则 $\phi^P_{\mathrm{sf}}(x)$ 表示存在一个有限前缀，在该前缀上存在一个从位置 $x+1$、状态为 s 开始的 $A_{\phi_{\mathrm{sf}}}$ 的接受运行。

请注意，此处的安全约束现在只涉及从当前位置开始的前缀。

我们还需要确保具有相同源状态的转移上的环视彼此不相交，即它们不能同时满足。考虑两个转移：

$$p \xrightarrow{\phi^P_{\mathrm{la}}(x) \mid w_1, \phi^P_{\mathrm{mv}}(x,y), \phi^P_{\mathrm{sf}}(x)} q$$
$$p \xrightarrow{\psi^R_{\mathrm{la}}(x) \mid w', \psi^R_{\mathrm{mv}}(x,y), \psi^R_{\mathrm{sf}}(x)} q'$$

根据 $\mathcal{T}'$ 的定义，我们知道 $\phi_{\mathrm{la}}(x)$ 和 $\psi_{\mathrm{la}}(x)$ 不相交。对于任意集合 $P, R \subseteq Q$，这种不相交性会传递到修改后的公式 $\phi^P_{\mathrm{la}}(x)$ 和 $\psi^R_{\mathrm{la}}(x)$，因为将 $\phi^P_{\mathrm{la}}(x)$（分别为 $\psi^R_{\mathrm{la}}(x)$）的模型投影到字母表 Σ 后，恰好就是 $\phi_{\mathrm{la}}(x)$（分别为 $\psi_{\mathrm{la}}(x)$）的模型。

最后，根据引理 C.13，我们知道 $\mathcal{T}'$ 与某个 2-dT 有效等价，因此它定义了一个确定性正则函数。请注意，$\mathcal{T}_{\mathrm{sf}}$ 是一个 1-dT，其所有状态都是终态。根据 [9] 的定理 3.11，已知将此类转导器与确定性正则函数复合，只会得到确定性正则函数。◀

引理 6.2 证明的最后论证引理 6.2 的结论可直接由引理 D.1 得出。引理 D.1 将任意跳跃式单子二阶逻辑$_g$-转导器转换为行走式单子二阶逻辑$_g$-转导器；而引理 D.2 则证明，后者定义了确定性正则函数。