arXiv:2301.06237v1 [cs.LO] 16 Jan 2023

# 指针程序中序列的分离逻辑及其可判定性

曹天月*、张博文*、金昭‡、曹永志* 和王汉品*†

* 高可信软件技术教育部重点实验室、北京大学计算机科学学院，中国北京

† 广州大学计算机科学与网络工程学院，中国广州

‡ 郑州大学计算机与人工智能学院，中国郑州

tycao@stu.pku.edu.cn, zhangbowen@pku.edu.cn,
jinzhao@zzu.edu.cn, caoyz@pku.edu.cn,
whpxhy@pku.edu.cn

摘要—**分离逻辑及其变体能够描述指针程序的各种性质。然而，要描述序列相关的性质，往往难以将其形式化。为处理变长序列和多级数据结构的性质，我们提出了序列堆分离逻辑，将序列融入对堆操作程序的逻辑推理中。序列变量量词以及存储序列的单例堆（序列单例堆）是该逻辑中的新元素。此外，我们研究了两个片段的可满足性问题。序列堆分离逻辑的命题片段是可判定的，而对程序变量有 2 次交替、对序列变量有 1 次交替的片段是不可判定的。此外，我们还探讨了具有前束范式的该逻辑中可判定片段与不可判定片段之间的边界。**

索引词—**分离逻辑、谓词逻辑、高阶、序列、可判定性**

## I. 引言

Reynolds 提出的经典分离逻辑 [1]被广泛用于描述和验证各种堆操作程序的性质。它发展出了许多版本：面向对象分离逻辑 [2], 定量分离逻辑 [3], 高阶分离逻辑 [4], 带归纳谓词的分离逻辑 [5], 等。它的分离合取 $*$ 和分离蕴含 $-\!\!*$能够更简洁地表达可变数据结构。

序列在程序中也得到广泛应用。许多数据结构都可以抽象为序列，包括 C 中的向量++以及 Java 中的 ArrayList。针对序列已有许多字符串求解器，例如 CVC4 [6] 和 Z3str 3 [7]。序列谓词逻辑 [8] 引入了序列变量和序列函数，并允许对序列变量使用量词。借助序列变量，可以优雅地定义排序函数。

然而，据我们所知，目前尚无一套完整的形式系统，能够对同时涉及堆和序列的程序进行逻辑推理。 [1]中提出的示例提供了一种形式化验证程序是否实现了链表反转的方法。该程序旨在反转 $x$ 中的序列 $\alpha_0$ ，并将其放入 $y$。该程序的不变式如下所示：

$$\exists\alpha\exists\beta.\ \big((\mathtt{ls}(x,\alpha) * \mathtt{ls}(y,\beta)) \wedge \alpha_0^\dagger == \alpha^\dagger \circ \beta\big), \quad (1)$$

其中， $\alpha^\dagger$ 表示序列 $\alpha$的反转，而项$\alpha \circ \beta$ 表示序列 $\alpha$ 和$\beta$ 的连接。谓词 ls($x$, $\alpha$) 表示一个从 $x$开始的单向链表，其定义如下：

$$\begin{aligned} \mathtt{ls}(x,\varepsilon) &\triangleq x = \mathbf{nil}, \\ \mathtt{ls}(x, n\circ\alpha) &\triangleq \exists y.\ x \mapsto y \circ n * \mathtt{ls}(y,\alpha). \end{aligned} \quad (2)$$

方程（1）中定义的性质可归结为指针程序中变长序列的性质。这些性质常见于许多常用的可变数据结构中，例如栈、队列和图。此外，变长序列也出现在多级数据结构中，例如操作系统中的分页和文件系统。我们有必要引入一种逻辑来描述和验证这类性质。

本文旨在为指针程序中的**变长序列**和**多级数据结构**建立逻辑基础，重点关注其定义、表达能力和可判定性。我们提出序列堆分离逻辑，将**序列纳入指针程序的逻辑推理**。该逻辑扩展了 [1], 提出的经典分离逻辑和 [8] 提出的序列谓词逻辑。

序列堆分离逻辑可以看作 [4] 提出的高阶分离逻辑的一个片段。序列量词可以约化为谓词（或集合）上的量词。序列堆分离逻辑的表达能力弱于高阶分离逻辑，但可用于指针程序中涉及序列的许多场景，而且具有更好的可判定性结果。

在序列堆分离逻辑的定义方面，我们将堆模型定义为从位置到序列的有限偏映射，这与 [9] 中定义的块模型 Heap$_B$ 相似。该模型可写作 $h$ : Loc $_{\text{有限}}\!\rightharpoonup$(Loc $\cup$值)*, 而经典分离逻辑中定义的模型 $h$ : Loc $_{\text{有限}}\!\rightharpoonup$ Loc $\cup$值与之相对。

我们还**将序列单例堆定义为**项 $x \mapsto \alpha$ ，以适应更多变长序列场景。该

项可用于表示栈、队列、可变参数、可变长度分支树、出度无界图等。我们以栈为例。

在经典分离逻辑中，必须使用式（2）中定义的归纳谓词 ls($x$, $\alpha$) 来描述栈，其中 $x$ 和 $\alpha$ 分别表示栈顶指针和栈的内容。然而，借助序列堆分离逻辑中定义的序列单例堆 $x \mapsto \alpha$，可以使用该公式描述栈，而无需归纳谓词。

序列单例堆的定义有助于得出可判定性结果。我们证明，带有序列单例堆的 $\Sigma_1$ 片段是可判定的；相比之下，$\Sigma_1$ 片段的不可判定性结果见 [10]，其中定义了列表谓词 ls($x$, $y$)、分离合取和分离蕴含。

此外，恰当定义 $\alpha$ 的形式，有助于该逻辑描述 **多级数据结构**。例如，若 $\alpha$ 的形式为 $\alpha\#\beta$（其中 $\alpha$ 表示位置序列，$\beta$ 表示内容序列，# 用于分隔这两个序列），则可以通过以下方式描述基于块的云存储系统中的双层数据结构 [9]。

$$(x \mapsto \alpha\#\varepsilon) * \forall y \exists \beta.\ \big(y \in \alpha \Rightarrow (y \hookrightarrow \varepsilon\#\beta)\big), \tag{3}$$

其中，$x$ 表示块位置，$\alpha$ 表示内容位置序列，$\beta$ 表示 $\alpha$ 中每个位置的内容。项$y \in \alpha$ 表示 $y$，它出现在 $\alpha$中，定义如下：

$$y \in \alpha \triangleq \exists \alpha_1 \exists \alpha_2.\ \alpha == \alpha_1 \circ y \circ \alpha_2.$$

公式（3）可由图（1）说明。

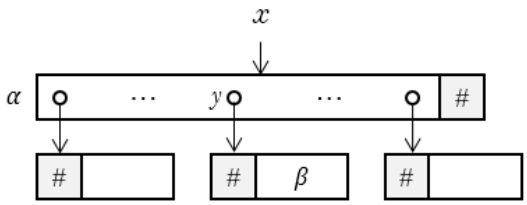


图 1：双层数据结构的表示

我们注意到，[8]中提出的序列谓词逻辑也能够描述变长序列的各种性质。然而，该逻辑不能直接用于描述指针程序中的序列。序列堆分离逻辑不仅仅是序列谓词逻辑与分离逻辑的简单组合。该逻辑丰富了二者的组合方式。

我们还注意到，序列堆分离逻辑的模型与 [9]中提出的基于块的分离逻辑相似。然而，序列堆分离逻辑中的序列单例堆和序列量词提供了更强的抽象能力，可用于更广泛的应用场景，而不仅限于基于块的云存储系统。

例如，组合逻辑可以简化对栈的描述表达式。此外，单独的序列谓词逻辑和分离逻辑无法描述方程（3）所示的多级数据结构，而组合逻辑可以做到这一点。

我们还研究了序列-堆分离逻辑片段的可判定性问题。这有助于我们为可判定片段寻找算法，也能深入理解该逻辑的表达能力。针对可满足性问题，我们得到以下两个主要片段。一个是可判定片段，其范围介于经典分离逻辑的 $\Sigma_1$ 片段与包含列表谓词的逻辑之间。另一个是不可判定片段，其中程序变量量词有两次交替，序列变量量词有一次交替。

本文组织如下。第二节介绍分离逻辑和序列谓词逻辑的一些基本概念。第三节定义序列堆分离逻辑，并探讨其表达能力。第四节证明，$\Sigma_1$ 片段的可满足性问题是可判定的。第五节证明，具有前束范式 $\forall_x^*\forall_\alpha^* \cap \forall_x^*\exists_x^*\exists_\alpha^*$ 的片段的可满足性问题是不可判定的。此外，我们还探讨可判定片段与不可判定片段之间的界限。第六节总结本文，并提出关于序列堆分离逻辑的未来研究方向。

**相关工作**。分离逻辑能够表达堆上的性质，例如悬空指针、无环性和内存泄漏。[11] 然而，已证明该逻辑本身是不可判定的 [12]。我们研究以下两类片段：不含归纳谓词的基本片段，以及$\Sigma_1$ 含有归纳谓词的片段。

在基本片段中，不存在用户定义谓词或硬编码谓词。已知，$\Pi_1$ 片段 SL($*$, $-\!\!*$) 的有效性问题是可判定的 [13]，，而带有量词且不含分离蕴含的基本片段 SL($\forall$, $\hookrightarrow$) 的有效性问题是不可判定的 [13]。如果将 $\varphi_1$ 公式 $\varphi_1 -\!\!* \varphi_2$ 中满足条件的堆大小限制为有界（即 $n$），那么就可以得到一个可判定的基本片段 SL($*$, $-\!\!*^n$, $\exists$) [12]。要得到可判定片段，可以限制前束范式公式中的量词交替次数，或限制量化变量的数量。对于量词交替，具有 2 次量词交替的片段 SL($\exists^*\forall^*$) 是可判定的 [14]，，而具有 3 次量词交替的片段 SL($\exists^*\forall^*\exists^*$) 是不可判定的 [14]。对于量化变量，含有 1 个量化变量的 1SL1 是可判定的 [15]，，而含有 2 个量化变量的 1SL2 是不可判定的 [16]。

在 $\Sigma_1$ 片段中，包含分离蕴含（或其变体）以及用于描述数据结构的归纳谓词。一项基本结果是，带有链表的 $\Sigma_1$ 片段 SL($*$, $-\!\!*$, ls) 的可满足性问题是不可判定的 [10]。然而，若将链表谓词限制为长度至多为 2 的列表，则可得到一个可判定片段 SL($*$, $-\!\!*$, $\text{reach}_2$) [10]。此外，若进一步限制分离模型中的堆并集，则可得到强分离逻辑 [17]。已证明，带有链表谓词的强分离片段 SSL($*$, $-\!\!\!\!\!\!*$, ls) 是可判定的。

上述所有片段均列于表 I。

序列谓词逻辑是对字方程的扩展。具体而言，字方程是序列谓词逻辑的 $\Sigma_1$ 片段或 $\Pi_1$ 片段，其中仅包含序列变量量词

表 I：分离逻辑片段的可判定性

| 片段 | 模型 | 关键联结词和谓词 | 可判定性 |
|---|---|---|---|
| 分离逻辑$(*, -\!\!*)$ | 分离模型 | $\neg, \mapsto, *, -\!\!*$ | 可判定 |
| 分离逻辑$(\forall, \hookrightarrow)$ | | $\neg, \forall, \hookrightarrow$ | 不可判定 |
| SL$(*, -\!\!*^n, \exists)$ | | $*\neg, , -\!\!*^n, \exists$ | 可判定 |
| 分离逻辑$(\exists^*\forall^*)$ | | 前束 $\exists^*\forall^*$ | 可判定 |
| 分离逻辑$(\exists^*\forall^*\exists^*)$ | | 前束 $\exists^*\forall^*\exists^*$ | 不可判定 |
| 1SL1 | | 1 个量化变量 | 可判定 |
| 1SL2 | | 2 个量化变量 | 不可判定 |
| 分离逻辑$(*, -\!\!*, \mathtt{ls})$ | 分离模型 | $\neg, *, -\!\!*, \mathtt{ls}(x, y)$ | 不可判定 |
| 分离逻辑$(*, -\!\!*, \text{可达性}_2)$ | 分离模型 | $\neg, *, -\!\!*, \text{可达}_2(x, y)$ | 可判定 |
| SSL$(*, \overline{-\!\!*}, \mathtt{ls})$ | 强分离模型 | $\neg, *, \overline{-\!\!*}, \mathtt{ls}$ | 可判定 |

。它可以表达序列上的许多性质，例如共轭词，以及词上的字典序 [18]。另一方面，它无法表达诸如“本原性”和“长度相等”这样的性质。研究表明，自由半群上字方程的任意布尔组合都可以化简为单个字方程 [19]。

与经典谓词逻辑不同，序列谓词逻辑中定义了序列变量、序列谓词和序列函数 [8]。字方程是否存在解的问题是可判定的 [19]。也就是说，序列谓词逻辑的 $\Sigma_1$ 片段的真值是可判定的。进一步的研究表明，形式为 $\forall^*\exists^*$ [20]、$\exists^*\forall^*$ [21] 或 $\exists^*\forall^*\exists^*$ [22] 的正公式（其中不含否定或不等式）的真值是不可判定的。为了得到一个可判定片段，可以将形式为 $\exists^*\forall^*$ 的公式限制为正公式 [22]。

有一些研究结合了分离逻辑与序列。在对云存储系统进行形式化时，可以构建一种建模语言**IMDSS** [23]，以及一种基于分离逻辑的证明系统 [9]。这些论文将堆的值定义为有限序列，并在不使用序列单例堆的情况下描述块的属性。基于 Coq 的证明助手 [24] 是根据 [9]中定义的逻辑实现的。

## II. 预备知识

本节介绍分离逻辑和序列谓词逻辑的一些基本概念。

### A. 分离逻辑

本小节主要关注分离逻辑的定义和表达能力。

定义 *2.1*（分离逻辑的定义）：含有 1 个记录的分离逻辑语法 [25]定义如下：

$t ::= \mathbf{空值} \mid n \mid x \mid \mathtt{f}^n(t, \ldots, t) p ::= \quad t = t \mid \mathbf{空堆} \mid t \mapsto t \mid \mathrm{P}^n(t, \ldots, t) \varphi ::$
$= \quad p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \Rightarrow \varphi \mid \varphi * \varphi \mid \varphi -\!\!* \varphi \mid \exists x.\varphi,$

其中，$n$ 表示常量，$x$ 表示变量，$\mathtt{f}^n$ 表示将 $\mathbb{N}^n$ 映射到 $\mathbb{N}$ 的$n$元函数，而$\mathrm{P}^n$ 表示将 $\mathbb{N}^n$ 映射到 $\{0,1\}$的 $n$元谓词。

分离逻辑的基础模型 $\sigma = (s, h)$ 由一个栈 $s$ 和一个堆 $h$组成。集合 Loc 包含除 **空值**之外的位置。集合 Var 包含变量符号，包括 $x, y, z, \ldots$。集合 Val 包含整数值。模型定义如下：

$$\mathrm{Loc}, \mathrm{Val} \subseteq \mathbb{N} \qquad \mathbf{nil} \in \mathrm{Atoms}$$
$$s : \mathrm{Var} \to \mathrm{Loc} \cup \mathrm{Val} \qquad h : \mathrm{Loc} \xrightarrow{\text{fin}} (\mathrm{Loc} \cup \mathrm{Val})^*.$$

项 $\mathrm{f}^n(t_1, \ldots, t_n)$ 的语义可定义如下，其中 $\mathtt{f}^n_\sigma$ 是 $\mathrm{f}^n$ 的解释。

$$[\![\mathtt{f}^n(t_1, \ldots, t_n)]\!]\sigma = \mathtt{f}^n_\sigma([\![t_1]\!]\sigma, \ldots, [\![t_n]\!]\sigma).$$

公式的语义归纳定义如下：

$\sigma \vDash t_1 = t_2$ 当且仅当 $[\![t_1]\!]\sigma = [\![t_2]\!]\sigma\sigma \vDash$ 空堆当且仅当 $\mathrm{dom}(h) = \varnothing\sigma \vDash t_1 \mapsto t_2$ 当且仅当 $[\![t_1]\!]\sigma \neq$ 为空值且 $\mathrm{dom}(h) = \{[\![t_1]\!]\sigma\}$且 $h([\![t_1]\!]\sigma) = [\![t_2]\!]\sigma\sigma \vDash \mathrm{P}^n(t_1, \ldots, t_n)$ 当且仅当 $\mathrm{P}^n_\sigma([\![t_1]\!]\sigma, \ldots, [\![t_n]\!]\sigma)$ 成立。$\sigma \vDash \neg\varphi$ 当且仅当 $\sigma \nvDash \varphi\sigma \vDash \varphi_1 \wedge \varphi_2$ 当且仅当 $\sigma \vDash \varphi_1$ 且 $\sigma \vDash \varphi_2\sigma \vDash \varphi_1 \vee \varphi_2$ 当且仅当 $\sigma \vDash \varphi_1$ 或 $\sigma \vDash \varphi_2\sigma \vDash \varphi_1 \Rightarrow \varphi_2$ 当且仅当如果 $\sigma \vDash \varphi_1$，那么 $\sigma \vDash \varphi_2\sigma \vDash \varphi_1 * \varphi_2$ 当且仅当存在堆 $h_1$ 和 $h_2$，使得 $\mathrm{dom}(h_1) \cap \mathrm{dom}(h_2) = \varnothing$，且 $h = h_1 \uplus h_2$，并且 $(s, h_1) \vDash \varphi_1$，且 $(s, h_2) \vDash \varphi_2$.

$\sigma \vDash \varphi_1 -\!\!* \varphi_2$ 当且仅当对于所有堆 $h_1$，如果 $\mathrm{dom}(h_1) \cap \mathrm{dom}(h) = \varnothing$，并且 $(s, h_1) \vDash \varphi_1$，

then $(s, h_1 \uplus h) \vDash \varphi_2$.

$\sigma \vDash \exists x.\varphi$ 当且仅当存在 $x_0$ 属于位置 $\cup$ 值,

such that $(s[x \to x_0], h) \vDash \varphi$,

其中，$\mathsf{P}_\sigma^n$ 表示 $\mathsf{P}^n$ 的解释。

为方便起见，我们采用以下记法 [25]，表示 $t$ 指向 $t_1, \ldots, t_n$，其中 $n$ 是某个固定值。

$$t \mapsto t_1, \ldots, t_n \triangleq (t \mapsto t_1) * \cdots * (t + n - 1 \mapsto t_n). \quad (4)$$

在分离逻辑的某些片段中，堆模型定义为 $h : \mathsf{Loc} \xrightarrow{\text{fin}} \mathsf{Val}^n$，其中 $n$ [25] [26] 是某个固定值。在这种情况下，公式（4）将不再成立。该记法的语义定义如下:

$$\sigma \vDash t \mapsto t_1, \ldots, t_n \quad \text{iff} \quad [\![t]\!]\sigma \neq \mathbf{nil} \text{ and } \mathtt{dom}(h) = \{[\![t]\!]\sigma\}, \text{ and } h([\![t]\!]\sigma) = ([\![t_1]\!]\sigma, \ldots, [\![t_n]\!]\sigma).$$

我们注意到，无论是在分离逻辑中，还是在其某个片段（如符号堆 [11] [26], 强分离逻辑[17], 等）中，单例堆都是用某个固定数目 $n$ 定义的。它可能不适合描述块或图的性质，因为其中每个节点的出度都可能无界。

本文 [9]将序列引入分离逻辑，以描述基于块的云存储系统的性质。该逻辑的模型定义为如下形式的五元组 $(s_F, s_B, s_V, h_B, h_V)$，其中 $s_F, s_B, s_V$分别是文件变量、块变量和位置变量的赋值，$h_B, h_V$ 分别是块堆和值堆。文件栈 $s_F$是从文件变量到块地址序列的映射，记作 $s_F \triangleq \mathsf{FVar} \to \mathsf{BLoc}^*$；而 $h_B$ 是从块到位置序列的有限偏映射，记作 $h_B \triangleq \mathsf{BLoc} \xrightarrow{\text{fin}} \mathsf{Loc}^*$。尽管$s_F$ 和 $h_B$ 都取值于序列，但该逻辑并未引入序列变量、序列函数或序列单例堆。该逻辑通过对块变量使用存在量词，并对块变量的值使用相等关系来访问这些序列。

### *B.* 字方程与序列谓词逻辑

字方程可以看作序列谓词逻辑的一个特例。其定义如下:

定义 *2.2*（字方程）：假设 $X^*$ 是一个满足 $X = \{n_1, \ldots, n_k\}$的自由半群，其中包含未知量$\alpha_1, \ldots, \alpha_m \in X^*$，而 $n_1, \ldots, n_k$是生成元。字方程是如下形式的相等:

$$U(n_1, \ldots, n_k, \alpha_1, \ldots, \alpha_m) == V(n_1, \ldots, n_k, \alpha_1, \ldots, \alpha_m),$$

其中，$==$ 表示两个序列的连接之间的相等。如果 $U == V$有解，则该字方程成立。

字方程的定义可以通过布尔组合加以扩展，如定义 2.3 所示。

定义 *2.3*（字方程的布尔组合）：字方程的布尔组合定义如下:

$$t \quad ::= \quad n \mid \alpha \mid t \circ t$$

$$\varphi \quad ::= \quad t == t \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi,$$

其中，$n$ 表示常量，$\alpha$ 表示序列变量，$t_1 \circ t_2$ 表示两个序列 $t_1$ 和 $t_2$的拼接。

字方程能够描述序列的许多性质。根据定理 2.1[18], 每个字方程的布尔组合都可以化简为一个字方程。

定理 *2.1*：对于上述定义的每个布尔组合 $\varphi$，都存在一个单独的字方程 $T(\varphi) \triangleq U == V$，它与 $\varphi$ 等价。该定理的证明概要见附录。

如果在字方程中加入序列函数，以及对变量和序列的量词，就可以得到序列谓词逻辑的如下定义。

定义 *2.4*（序列谓词逻辑）：序列谓词逻辑 [8]的语法定义如下:

$$\begin{aligned} t \quad &::= \quad n \mid x \mid \alpha \mid t \circ t \mid \mathtt{f}(t, \ldots, t) \mid \overline{\mathtt{f}}(t, \ldots, t) \\ \varphi \quad &::= \quad t == t \mid \mathtt{P}(t, \ldots, t) \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \\ &\quad \mid \varphi \Rightarrow \varphi \mid \exists x.\varphi \mid \exists \alpha.\varphi. \end{aligned}$$

变量 $x$ 表示个体变量，$\alpha$ 表示序列变量。f、f 和 P 的元数可以是固定的，也可以是灵活的。若元数固定，则参数中不含序列变量；若元数灵活，则可以包含序列变量。

该逻辑的结构定义为 (Val $I$)，其中 Val 是论域，$I$ 是对个体常量、序列常量、函数和谓词的解释。该逻辑的赋值可定义为 ($s_x$, $s_\alpha$)，其中 $s_x$ : IVar $\to$Val, $s_\alpha$ : SVar $\to$ Val$^*$，且 IVar $= \{x, y, z, \ldots\}$、SVar $= \{\alpha, \beta, \gamma, \ldots\}$。

在序列谓词逻辑中，$t_1 \circ t_2$ 表示两个序列 $t_1$ 和 $t_2$的连接，$t_1 == t_2$ 表示两个方程 $t_1$ 和 $t_2$ 相等。由于篇幅有限，我们不列出该逻辑的语义。详情请参见 [8] [27]。

### *C.* 序列谓词逻辑的表达能力

我们回顾序列谓词逻辑能够表达的性质。这些性质有助于描述序列堆分离逻辑的性质。

**长度**。序列谓词逻辑可用于求序列的长度。函数length($\alpha$) 表示序列 $\alpha$的长度，其可归纳定义如下:

$$\begin{aligned} &\mathtt{length}(\varepsilon) \triangleq 0 \\ &\mathtt{length}(n \circ \alpha) \triangleq 1 + \mathtt{length}(\alpha). \end{aligned}$$

为方便起见，我们定义 $|\alpha|$表示length($\alpha$)。

**统计**。序列谓词逻辑可用于查找特定项的出现次数。函数 find($\alpha, x$) 表示 $x$ 在序列 $\alpha$ 中的出现次数，其归纳定义如下:

$$\mathtt{find}(\varepsilon, x) \triangleq 0$$

$$\texttt{find}(n \circ \alpha, x) \triangleq 1 + \texttt{find}(\alpha, x), \qquad n = x$$
$$\texttt{find}(n \circ \alpha, x) \triangleq \texttt{find}(\alpha, x), \qquad n \neq x$$

为方便起见，我们定义 $|\alpha|_x$ 表示 find($\alpha, x$)。**查找**。谓词 eq($x_1, \alpha, x_2$) 表示序列 $\alpha$ 的第 $x_2$ 项为 $x_1$，可表示如下：

$$\texttt{eq}(x_1, \alpha, x_2) \triangleq \exists \alpha_1 \exists \alpha_2.\ \alpha == \alpha_1 \circ x_1 \circ \alpha_2 \wedge |\alpha_1 \circ x_1| = x_2.$$

请注意，$\alpha(x)$ 并未作为项出现在定义 2.4 中。若如此，则考虑 $x$ 超出序列 $\alpha$ 长度的情况。应引入一个新值 $\perp$ 来表示非法项，这可能会使逻辑变得复杂。为方便起见，我们用 $x_1 = \alpha(x_2)$ 表示 eq($x_1, \alpha, x_2$)：

$$x_1 = \alpha(x_2) \triangleq \texttt{eq}(x_1, \alpha, x_2).$$

其他性质可以用查找来表达。下面列出其中一些。

项目与序列之间的关系可定义为 $x \in \alpha$，这意味着可以在序列 $\alpha$ 中找到 $x$：

$$x \in \alpha \triangleq \exists x_1.\ x = \alpha(x_1).$$

序列 $\alpha_1$ 的第 $x_1$ 个项目等于序列 $\alpha_2$ 的第 $x_2$ 个项目：

$$\texttt{eq}(\alpha_1, x_1, \alpha_2, x_2) \triangleq \exists x_3.\ x_3 = \alpha_1(x_1) \wedge x_3 = \alpha_2(x_2).$$

**一元性质**。序列 $\alpha$ 中的所有项目都满足一元性质 $\mathrm{P}^1$($x$)：

$$\forall x \forall \alpha_1 \forall \alpha_2.\ \alpha == \alpha_1 \circ x \circ \alpha_2 \Rightarrow \mathrm{P}^1(x).$$

**二元性质**。序列 $\alpha$ 中任意两个索引不同的项目都满足二元性质 $\mathrm{P}^2$($x_1, x_2$)：

$$\forall x_1 \forall x_2 \forall \alpha_1 \forall \alpha_2 \forall \alpha_3.\ \alpha == \alpha_1 \circ x_1 \circ \alpha_1 \circ x_2 \circ \alpha_3 \Rightarrow \mathrm{P}^2(x_1, x_2).$$

下文定义的递增和类集合是二元属性。**递增**。序列 $\alpha$ 严格递增：

$$\texttt{Inc}(\alpha) \triangleq \forall x_1 \forall x_2 \forall \alpha_1 \forall \alpha_2 \forall \alpha_3.\ \alpha == \alpha_1 \circ x_1 \circ \alpha_2 \circ x_2 \circ \alpha_3 \Rightarrow x_1 < x_2.$$

**类集合**。序列 $\alpha$ 中的任意两项都不相同：

$$\texttt{Diff}(\alpha) \triangleq \forall x_1 \forall x_2 \forall \alpha_1 \forall \alpha_2 \forall \alpha_3.\ \alpha == \alpha_1 \circ x_1 \circ \alpha_1 \circ x_2 \circ \alpha_3 \Rightarrow x_1 \neq x_2.$$

**片段**。序列 $\alpha_2$ 是 $\alpha_1$的连续子序列：

$$\alpha_2 \in \texttt{Seg}(\alpha_1) \triangleq \exists \alpha_3 \exists \alpha_4.\alpha_1 == \alpha_3 \circ \alpha_2 \circ \alpha_4.$$

**截断**。序列 $\alpha_2$ 是 $\alpha_1$ 的连续子序列，其索引范围为 $[x_1, x_2)$：

$$\begin{aligned} & \alpha_2 == \texttt{Trunc}(\alpha_1, x_1, x_2) \\ \triangleq\ & \exists \alpha_3 \exists \alpha_4.\ \alpha_1 == \alpha_3 \circ \alpha_2 \circ \alpha_4 \\ & \qquad \wedge |\alpha_3| = x_1 - 1 \wedge |\alpha_3 \circ \alpha_2| = x_2 - 1. \end{aligned}$$

III. 序列堆分离逻辑

序列堆分离逻辑是经典分离逻辑和序列谓词逻辑的扩展。它可以描述变长序列和多层数据结构的性质。有些性质仅用经典分离逻辑或序列谓词逻辑难以表达。

本节介绍序列堆分离逻辑（SeqSL）的定义及其表达能力。

### A. 序列堆分离逻辑的定义 *gic*

定义 *3.1*（序列堆分离逻辑的语法）：该逻辑的语法定义如下：

$$\begin{array}{lcl} t_x & ::= & \mathbf{nil} \mid \# \mid n \mid x \\ t_\alpha & ::= & \varepsilon \mid t_x \mid \alpha \mid t_\alpha \circ t_\alpha \\ \varphi & ::= & t_x = t_x \mid t_\alpha == t_\alpha \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \Rightarrow \varphi \\ & & \mid \mathbf{emp} \mid t_x \mapsto t_\alpha \mid \varphi * \varphi \mid \varphi \mathrel{-\!\!*} \varphi \mid \exists_x x.\varphi \mid \exists_\alpha \alpha.\varphi. \end{array}$$

请注意，根据 [8], 中的定义，可以在 SeqSL 中定义其他函数和谓词，例如比较序列以及从序列中取出一项。上述逻辑可以表达最常用的谓词和函数。

定义 *3.2*（序列堆分离逻辑的模型）：SeqSL 的签名中有三类变量：程序变量 PVar $= \{x, y, z, \dots\}$、序列变量 SVar $= \{\alpha, \beta, \gamma, \dots\}$。这些集合都是可数的。模型 $\sigma = (s_x,\ s_\alpha,\ h)$可定义如下：

$$\begin{array}{ll} \text{Loc}, \text{Val} \subseteq \mathbb{N} & \mathbf{nil}, \# \in \text{Atoms} \\ \text{Loc} \cap \text{Atoms} = \varnothing & s_x : \text{PVar} \to \text{Loc} \cup \text{Val} \\ s_\alpha : \text{SVar} \to (\text{Loc} \cup \text{Val})^* & h : \text{Loc} \xrightarrow{\text{fin}} (\text{Loc} \cup \text{Val})^*, \end{array}$$

其中，Loc 是位置集合，Val 是值集合，Atoms 是保留字集合，$s_x$ 表示栈，$s_\alpha$表示对序列变量的赋值，而 $h$ 表示用于存储位置、值或原子序列的堆。

定义*3.3*（项的语义）：符号 $t_x$ 表示**个体项**，其值取自 Val。该项的指称语义可归纳定义如下：

$$\begin{array}{ll} [\![\mathbf{nil}]\!]\sigma = \mathbf{nil} & [\![\#]\!]\sigma = \# \\ [\![n]\!]\sigma = n & [\![x]\!]\sigma = s_x(x), \end{array}$$

其中**空值**，Atoms 中的 # 不能用作位置，# 用于描述多级数据结构。

符号 $t_\alpha$ 表示**序列项**，其值取自 (Loc∪Val)$^*$。该项的指称语义可归纳定义如下：

$$\begin{array}{ll} [\![\varepsilon]\!]\sigma = \varepsilon & [\![t_x]\!]\sigma \text{ is defined above} \\ [\![\alpha]\!]\sigma = s_\alpha(\alpha) & [\![t_{\alpha_1} \circ t_{\alpha_2}]\!]\sigma = [\![t_{\alpha_1}]\!]\sigma \circ [\![t_{\alpha_2}]\!]\sigma, \end{array}$$

其中，$[\![t_{\alpha_1}]\!]\sigma \circ [\![t_{\alpha_2}]\!]\sigma$ 表示两个序列 $[\![t_{\alpha_1}]\!]\sigma$ 和 $[\![t_{\alpha_2}]\!]\sigma$的连接。

定义*3.4*（序列堆分离逻辑语义）：在 SeqSL 中，序列项目的索引从 1 开始。为方便起见，我们将值集合记为 =Val ∪ Loc。SeqSL 的语义定义如下：

$\sigma \vDash t_{x_1} = t_{x_2}$ 当且仅当 $[\![t_{x_1}]\!]\sigma = [\![t_{x_2}]\!]\sigma$.
$\sigma \vDash t_{\alpha_1} == t_{\alpha_2}$ 当且仅当 $[\![t_{\alpha_1}]\!]\sigma == [\![t_{\alpha_2}]\!]\sigma\sigma \vDash \neg\varphi$ 当且仅当 $\sigma \nvDash \varphi\sigma \vDash \varphi_1 \wedge \varphi_2$ 当且仅当 $\sigma \vDash \varphi_1$，且 $\sigma \vDash \varphi_2\sigma \vDash \varphi_1 \vee \varphi_2$ 当且仅当 $\sigma \vDash \varphi_1$，或者 $\sigma \vDash \varphi_2$. $\sigma \vDash \varphi_1 \Rightarrow \varphi_2$ 当且仅当如果 $\sigma \vDash \varphi_1$，那么 $\sigma \vDash \varphi_2$. $\sigma \vDash$ **空堆** 当且仅当定义域(*h*) $= \varnothing\sigma \vDash t_x \mapsto t_\alpha$ 当且仅当 $[\![t_x]\!]\sigma \notin$ Atom，定义域(*h*) $= \{[\![t_x]\!]\sigma\}$，且 $h([\![t_x]\!]\sigma) = [\![t_\alpha]\!]\sigma$

$\sigma \vDash$ 当且仅当存在堆 $h_1$ 和 $h_2$，使得 dom($h_1$) ∩ dom($h_2$) $= \varnothing$，且 $h = h_1 \uplus h_2$，($s_x, s_\alpha, h_1$) $\vDash \varphi_1$，并且($s_x, s_\alpha, h_2$) $\vDash \varphi_2\sigma \vDash \varphi_1 \mathrel{-\!\!*} \varphi_2$ 当且仅当对于所有堆 $h_1$，如果 dom($h_1$) ∩ dom($h$) $= \varnothing$，且($s_x, s_\alpha, h_1$) $\vDash \varphi_1$，那么($s_x$, $s_\alpha, h_1 \uplus h$) $\vDash \varphi_2\sigma \vDash \exists_x x.\varphi$

当且仅当存在 $x_0$ 属于值，

使得($s_x[x \to x_0]$， $s_\alpha$， $h$) $\vDash \varphi\sigma \vDash \exists_\alpha \alpha.\varphi$ 当且仅当存在 $\alpha_0$ 属于值*，

such that $(s_x, s_\alpha[\alpha \to \alpha_0], h) \vDash \varphi$,

其中 $s_x[x \to x_0]$ 和 $s_\alpha[\alpha \to \alpha_0]$ 分别表示程序变量和序列变量的替换。它们定义如下。

$$s_x[x \to x_0](x') = \begin{cases} s_x(x_0), & x' = x \\ s_x(x), & \text{otherwise} \end{cases}$$

$$s_\alpha[\alpha \to \alpha_0](\alpha') = \begin{cases} s_\alpha(\alpha_0), & \alpha' == \alpha \\ s_\alpha(\alpha), & \text{otherwise} \end{cases}$$

为方便起见，我们用 $\exists x.\varphi$ 表示 $\exists_x x.\varphi$，用 $\exists\alpha.\varphi$ 表示 $\exists_\alpha \alpha.\varphi$。

SeqSL 能表达一些性质，这些性质广泛用于指针程序的逻辑推理。下面列出其中一些。本文将使用这些记法。

| | |
|---|---|
| 分离牵引 | $\varphi_1 \mathrel{-\!\!\!\!*} \varphi_2 \triangleq \neg(\varphi_1 \mathrel{-\!\!*} \neg\varphi_2)$ |
| 全称量词 | $\forall\alpha.\varphi \triangleq \neg\exists\alpha.\ \neg\varphi$ |
| $\alpha$ 存储于 $xx$ | $x \hookrightarrow \alpha \triangleq x \mapsto \alpha * \mathbf{true}$ |
| 已分配 | $\mathtt{alloc}(x) \triangleq \exists\alpha.\ x \hookrightarrow \alpha$ |
| $\alpha_2$ 存储于 $\alpha_1(x_1)$ | $\alpha(x_1) \hookrightarrow \alpha_2$ |
| | $\triangleq\ \exists x.\ x = \alpha_1(x_1) \wedge x \hookrightarrow \alpha_2.$ |

*B.* 序列堆分离逻辑的表达能力

在本小节中，我们列举几个例子来展示 SeqSL 的表达能力。我们将这些例子分为两类：程序中关于可变长度序列的性质，以及关于多级数据结构的性质。

与经典分离逻辑类似，序列堆分离逻辑在公式中不区分位置和值。为了适用于实践中的更多场景，我们使用原子符号 # 手动区分位置和值。因此，序列单例堆可写为 $x \mapsto \alpha_l \circ \# \circ \alpha_x$，其中 $\alpha_l$ 表示位置序列，$\alpha_x$ 表示值序列。

*1)* 程序中变长序列的性质：我们列举一些关于图的性质，以展示 SeqSL 描述程序中变长序列的表达能力。这些性质包括：出度和可达性。

**出度**。节点 $x$ 的出度可记为 Outdeg($x$)。公式 Outdeg($x$) $= n$ 表示节点 $x$ 的出度为 $n_x$。定义如下：

$$\begin{aligned} &\mathtt{Outdeg}(x) = n_x \\ \triangleq\ &\exists\alpha_l \exists\alpha_x.\ x \hookrightarrow \alpha_l \circ \# \circ \alpha_x \wedge |\alpha_l| = n_x. \end{aligned} \tag{5}$$

请注意，保留字 **空值** 可能出现在 $\alpha_l$ 中。这不会影响定义的真值，因为 **空值** 可以视为位置 0。其出度为 0，而入度不一定为 0。

**可达性**。首先，我们定义一步可达性 $x_1 \rightsquigarrow x_2$，表示从 $x_1$ 到 $x_2$ 存在一条边。

$$x_1 \rightsquigarrow x_2 \triangleq \exists\alpha_{l_1} \exists\alpha_{l_2} \exists\alpha_x.\ l_1 \hookrightarrow \alpha_{l_1} \circ l_2 \circ \alpha_{l_2} \circ \# \circ \alpha_x.$$

然后，我们定义 reach$^n$($x_1$， $x_2$)，表示从 $x_1$ 到 $x_2$ 存在一条路径。该路径的长度为 $n$ ($n \geq 0$)。

$$\begin{aligned} \mathtt{reach}^0(x_1, x_2) &\triangleq x_1 = x_2 \\ \mathtt{reach}^{n+1}(x_1, x_2) &\triangleq \exists x_3.\ x_1 \rightsquigarrow x_3 * \mathtt{reach}^n(x_3, x_2) \end{aligned}$$

最后，我们定义 reach($x_1$， $x_2$)，表示从 $x_1$ 到 $x_2$ 存在一条长度大于或等于 0 的路径。

$$\mathtt{reach}(x_1, x_2) \triangleq \exists n.\ n \geq 0 \wedge \mathtt{reach}^n(x_1, x_2).$$

请注意，SeqSL 中定义的 reach($x_1$， $x_2$) 与形状分析中的分离逻辑片段所定义的 ls($x_1$， $x_2$) 不同。前者可以描述出度不受限制的图上的可达性，而后者只能描述链表上的可达性。

*2)* 多级数据结构的性质：我们列出两种存储系统上的一些性质，以展示 SeqSL 描述多级数据结构的表达能力。

不同于 SeqSL，SeqSL 描述存储系统的模型 $\sigma = (s_x, s_\alpha, h)$ 应在磁盘而非内存中进行解释。

**Windows 存储系统的属性**。基于树的 Windows 存储系统结构可以用

SeqSL 表示。文件的索引 $\alpha_l$ 存储在位置 $l$。文件 $\alpha_x$ 存储在位置 $l_1$。

$$\begin{aligned} & l \hookrightarrow \alpha_l \circ \# \circ \varepsilon * \\ & \forall l_1 \exists \alpha_x .\, (l_1 \in \alpha_l \Rightarrow l_1 \hookrightarrow \varepsilon \circ \# \circ \alpha_x). \end{aligned} \tag{6}$$

注意，子公式 $\forall l_1 \exists \alpha_x .(l_1 \in \alpha_l \Rightarrow l_1 \hookrightarrow \varepsilon \circ \# \circ \alpha_x)$ 不能替换为 $\forall x.\, \alpha_l(x) \hookrightarrow \varepsilon \circ \# \circ \alpha_x$。原因是，$\alpha_l$ 是一个有限序列。当 $x$ 超过$\alpha_l$的长度时，它会使 $\forall x.\, \alpha_l(x) \hookrightarrow \varepsilon \circ \# \circ \alpha_x$不成立。因此，我们必须用 $l_1 \in \alpha_l$限制$l_1$。

我们在公式（6）中使用 # 来描述双层数据结构。第一层仅存储位置 $\alpha_l$，而第二层仅存储数据 $\alpha_x$。这两种不同形式的内容，确保第二层中的位置 $l_1$ 不会指向第一层中的位置 $l$。

有了公式（6），我们就可以描述文件属性。我们列举其中两项。

可以通过文件的索引 $x_1$找到该文件：

$$\exists \alpha_x .\, \alpha_l(x_1) \hookrightarrow \varepsilon \circ \# \circ \alpha_x .$$

可以在文件 $\alpha_x$中找到内容 $x_1$：

$$\exists x_2 .\, x_1 = \alpha_x(x_2).$$

此外，我们可以使用 $x \mapsto \alpha_l \circ \#^n \circ \alpha_x$ 来描述多级数据结构的第 n 层。这有助于描述 C++中的多级指针、多级分页、多级寻址等。

**基于块的云存储系统中的性质**。SeqSL 可用于描述云存储系统中的块。一般而言，公式 $x \hookrightarrow \alpha$ 中的序列 $\alpha$ 可以视为一个块。给定相对地址 $l$，即可找到内容$\alpha(l)$。

为方便起见，我们定义谓词 IncIndex($\alpha_0, n$)，表示 $\alpha_0$ 的长度 $n+1$ 严格递增。$\alpha$ 的第一项为 1，最后一项为 $n+1$。

$$\begin{aligned} \texttt{IncIndex}(\alpha_0, n) \triangleq\ & \texttt{Inc}(\alpha_0) \wedge |\alpha_0| = n+1 \\ & \wedge 1 = \alpha_0(1) \wedge n+1 = \alpha_0(n+1). \end{aligned}$$

存储在位置 $l$ 的大文件 $\alpha$ 被划分为 $n$ 个较小的块，并存储在磁盘中。该文件的索引 $\alpha_l$ 存储在位置 $l$。函数 Trunc($\alpha, x_1, x_2$) 可用于定位这些块。以下公式中的序列变量 $\alpha_0$ 表示分割点序列，它应使谓词 IncIndex($\alpha_l, n$) 成立。还可以通过公式 $|\alpha_x| = 64$将每个块 $\alpha_x$ 的长度固定为 64。

$$\begin{aligned} & \exists \alpha_0 .\, \Big( \texttt{IncIndex}(\alpha_l, n) \wedge l \hookrightarrow \alpha_l \circ \# \circ \varepsilon \\ & * \big( \forall l_1 \exists x .\, (l_1 = \alpha_l(x) \wedge \alpha_x == \texttt{Trunc}(\alpha, \alpha_0(x), \alpha_0(x+1))) \\ & \Rightarrow l_1 \hookrightarrow \varepsilon \circ \# \circ \alpha_x \Big). \end{aligned}$$

上述定义的公式可由图（2）说明。

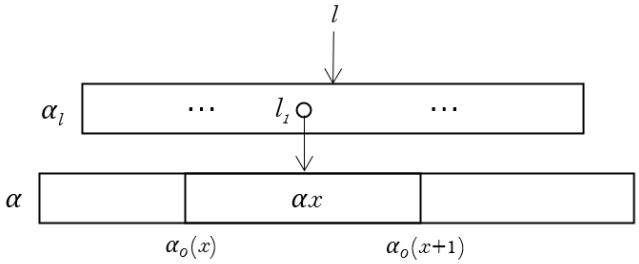


图 2：基于块的云存储系统结构

IV. 可判定片段

本节中，我们考虑 SeqSL 的一个可判定 $\Sigma_1$ 片段。它包含序列单例堆、分离合取、分离蕴含、相等以及序列连接。我们给出本节的以下主要结果。

定理 *4.1*（*SeqSL* 的可判定片段）：当语言限制如下时，其可满足性问题是可判定的：

$$\begin{aligned} t_x \quad & ::= \quad \mathbf{nil} \mid \# \mid x \\ t_\alpha \quad & ::= \quad \varepsilon \mid t_x \mid \alpha \mid t_\alpha \circ t_\alpha \\ \varphi \quad & ::= \quad t_x = t_x \mid t_\alpha == t_\alpha \mid \mathbf{false} \mid \varphi \Rightarrow \varphi \mid \mathbf{emp} \\ & \quad\ \mid x \mapsto t_\alpha \mid \varphi * \varphi \mid \varphi \mathrel{-\!\!*} \varphi . \end{aligned}$$

该片段的模型与 SeqSL 的模型相同，后者定义于定义 3.2 中。

我们将上述片段称为分离逻辑的 $\Sigma_1$ 片段（PSeqSL，其中字母“P”表示命题）。PSeqSL 是一个非平凡片段，原因如下：

- PSeqSL 的表达能力强于 $\Sigma_1$ 中提出的经典分离逻辑的片段，因为前者能够表达关于变长序列和多层数据结构的性质。

- 在 PSeqSL 中，堆存储变长序列。PSeqSL 中一个公式的模型可能比经典分离逻辑中公式的模型有更多可能性。

- 该片段中的序列和堆操作并非相互独立。它们的可满足性可能相互影响。例如，要判定公式 $x_1 \mapsto \alpha_1 \wedge x_1 \mapsto \alpha_2$ 是否为真，我们需要同时考虑堆上的两个条件，以及序列上的隐含条件 $\alpha_1 == \alpha_2$。要判定公式 ($x_1 \mapsto \alpha_1 * x_2 \mapsto \alpha_2$) $\wedge$($x_1 \mapsto \alpha_3 * x_2 \mapsto \alpha_4$) $\wedge\, x_1 \neq x_2$ 是否为真，我们需要考虑隐含条件 $\alpha_1 == \alpha_3 \wedge \alpha_2 == \alpha_4$。

- PSeqSL 可以描述 SeqSL 中一些由存在量词构成的公式。SeqSL 中的性质 alloc($x$) $\triangleq$ $\exists \alpha .\, x \hookrightarrow \alpha$ 可以用 PSeqSL 表示为：

$$\texttt{alloc}(x) \triangleq (x \mapsto \mathbf{nil}) \mathrel{-\!\!*} \mathbf{false}.$$

我们通过以下三个步骤证明定理 4.1：

1) 证明以下问题是可判定的：给定 PSeqSL 中的公式 $\varphi$，以及模型 $s_x, h$ 的一部分，判断是否存在 $s_\alpha$，使得 $(s_x, s_\alpha, h) \vDash \varphi$。2) 证明以下问题是可判定的：给定 PSeqSL 中的公式 $\varphi$，以及模型 $s_x$ 的一部分，判断是否存在 $s_\alpha, h$，使得 $(s_x, s_\alpha, h) \vDash \varphi$。3) 证明以下问题是可判定的：给定 PSeqSL 中的公式 $\varphi$，判断是否存在 $s_x, s_\alpha, h$，使得 $(s_x, s_\alpha, h) \vDash \varphi$。（定理 4.1）

第一步是证明的主要部分。没有分离蕴含的片段，其结果显然成立，因为 $h$ 的有限性意味着满足公式 $\varphi$ 的模型只有有限种可能。

与论文 [13], 类似，我们定义公式$\varphi$ 的大小 sz($\varphi$)，以表示判断公式 $\varphi$为真或其否定为真所需的最大堆大小。

定义 *4.1*（公式大小）：公式 $\varphi$的大小 sz($\varphi$) 可以归纳定义如下：

$$\begin{array}{ll}
\mathtt{sz}(t_{x_1} = t_{x_2}) = 0 & \mathtt{sz}(t_{\alpha_1} == t_{\alpha_2}) = 0 \\
\mathtt{sz}(\mathbf{false}) = 0 & \mathtt{sz}(\mathbf{emp}) = 1 \\
\mathtt{sz}(x \mapsto t_\alpha) = 1 & \mathtt{sz}(\varphi_1 \mathrel{-\!\!*} \varphi_2) = \mathtt{sz}(\varphi_2) \\
\multicolumn{2}{l}{\mathtt{sz}(\varphi_1 \Rightarrow \varphi_2) = \mathtt{max}(\mathtt{sz}(\varphi_1), \mathtt{sz}(\varphi_2))} \\
\multicolumn{2}{l}{\mathtt{sz}(\varphi_1 * \varphi_2) = \mathtt{sz}(\varphi_1) + \mathtt{sz}(\varphi_2).}
\end{array}$$

自由程序变量定义如下。

定义 *4.2*（*PSeqSL* 中的自由程序变量）：在 PSeqSL 中，项 $t_x$中的自由程序变量 FV$_{t,x}$($t_x$) 以及项 $t_\alpha$ 中的自由程序变量 FV$_{t,x}$($t_\alpha$) 可归纳定义如下：

$$\begin{array}{lll}
\mathrm{FV}_{t,x}(\mathbf{nil}) = \{\} & \mathrm{FV}_{t,x}(\#) = \{\} & \mathrm{FV}_{t,x}(x) = \{x\} \\
\mathrm{FV}_{t,x}(\alpha) = \{\} & \mathrm{FV}_{t,x}(\varepsilon) = \{\} & \\
\multicolumn{3}{l}{\mathrm{FV}_{t,x}(t_{\alpha_{x_1}} \circ t_{\alpha_{x_2}}) = \mathrm{FV}_{t,x}(t_{\alpha_{x_1}}) \cup \mathrm{FV}_{t,x}(t_{\alpha_{x_2}}).}
\end{array}$$

公式 $\varphi$ 中的自由程序变量 FV$_x$($\varphi$) 可归纳定义如下：

$$\begin{array}{l}
\mathrm{FV}_x(t_{x_1} = t_{x_2}) = \mathrm{FV}_{t,x}(t_{x_1}) \cup \mathrm{FV}_{t,x}(t_{x_2}) \\
\mathrm{FV}_x(t_{\alpha_1} == t_{\alpha_2}) = \mathrm{FV}_{t,x}(t_{\alpha_1}) \cup \mathrm{FV}_{t,x}(t_{\alpha_2}) \\
\mathrm{FV}_x(\mathbf{false}) = \{\} \\
\mathrm{FV}_x(\varphi_1 \Rightarrow \varphi_2) = \mathrm{FV}_x(\varphi_1) \cup \mathrm{FV}_x(\varphi_2) \\
\mathrm{FV}_x(\mathbf{emp}) = \{\} \\
\mathrm{FV}_x(x \mapsto t_\alpha) = \{x\} \cup \mathrm{FV}_{t,x}(t_\alpha) \\
\mathrm{FV}_x(\varphi_1 * \varphi_2) = \mathrm{FV}_x(\varphi_1) \cup \mathrm{FV}_x(\varphi_2) \\
\mathrm{FV}_x(\varphi_1 \mathrel{-\!\!*} \varphi_2) = \mathrm{FV}_x(\varphi_1) \cup \mathrm{FV}_x(\varphi_2).
\end{array}$$

我们定义序列项集合，以收集公式中出现的项。这有助于我们得到一个有限集合，其中包含所有可能满足该公式的堆模型。

定义 *4.3*（序列项集合）：公式 $\varphi$ 中出现的序列项集合 SeqTerms($\varphi$) 可归纳地定义如下：

$$\begin{array}{l}
\mathtt{SeqTerms}(t_{l_1} = t_{l_2}) = \{t_{l_1}, t_{l_2}\} \\
\mathtt{SeqTerms}(t_{\alpha_1} == t_{\alpha_2}) = \{t_{\alpha_1}, t_{\alpha_2}\} \\
\mathtt{SeqTerms}(\mathbf{false}) = \{\} \\
\mathtt{SeqTerms}(\mathbf{emp}) = \{\} \\
\mathtt{SeqTerms}(x \mapsto t_\alpha) = \{x, t_\alpha\} \\
\mathtt{SeqTerms}(\varphi_1 \Rightarrow \varphi_2) = \mathtt{SeqTerms}(\varphi_1) \cup \mathtt{SeqTerms}(\varphi_2) \\
\mathtt{SeqTerms}(\varphi_1 * \varphi_2) = \mathtt{SeqTerms}(\varphi_1) \cup \mathtt{SeqTerms}(\varphi_2) \\
\mathtt{SeqTerms}(\varphi_1 \mathrel{-\!\!*} \varphi_2) = \mathtt{SeqTerms}(\varphi_1) \cup \mathtt{SeqTerms}(\varphi_2).
\end{array}$$

分离蕴含涉及对堆的全称量词。论文 [13] 提出了一种方法，用于搜索经典分离逻辑片段中公式的所有可能模型。该方法将堆 $h$ 的定义域和值域限制为有限集合。为了处理序列，我们对 $h$ 的值域采用了不同的限制。$h$ 的值域包括公式中出现的所有序列项组成的集合、空序列 $\varepsilon$，以及一个值不同于公式中所有序列项的新序列项。因此，对堆的全称量词可以归约为对有限堆的全称量词。该归约保持可满足性。

我们提出以下引理，以证明公式 $\varphi_1 \mathrel{-\!\!*} \varphi_2$ 的小模型性质。

引理 4.1：给定模型 $\sigma = (s_x, s_\alpha, h)$，以及满足 $\varphi = \varphi_1 \mathrel{-\!\!*} \varphi_2$ 的公式 $\varphi$、$\varphi_1$ 和 $\varphi_2$，集合 $L$ = FV$_x$($\varphi_1$) ∪FV$_x$($\varphi_2$)，包含 Loc \ (dom($h$) ∪ $s$($L$)) 中前 max(sz($\varphi_1$), sz($\varphi_2$)) 个值的集合 $B$，以及满足 $[\![\beta]\!]\sigma' \notin \{[\![t_\alpha]\!]\sigma' \not\vDash_\alpha \in \mathrm{SeqTerms}(\alpha)\}$ 的新序列变量 $\beta$。我们定义 $\sigma' = (s_x, s'_\alpha, h)$，其中 $s'_\alpha$ 包含对 $\beta$ 的新赋值。令 $\mathcal{D} = B \cup s(L)$，且 $\mathcal{R} = \{[\![t_\alpha]\!]\sigma' \mid t_\alpha \in \mathrm{SeqTerms}(\varphi)\} \cup \{\varepsilon, [\![\beta]\!]\sigma'\}$。那么，当且仅当对于所有满足以下条件的 $h'$ 时，$(s_x, s_\alpha, h) \vDash \varphi_1 \mathrel{-\!\!*} \varphi_2$ 成立：

- $\mathtt{dom}(h') \cap \mathtt{dom}(h) = \varnothing$ and $(s_x, s'_\alpha, h') \vDash \varphi_1$,
- $\mathtt{dom}(h') \subseteq \mathcal{D}$,
- $\mathtt{rng}(h') \subseteq \mathcal{R}$,

命题 $(s_x, s'_\alpha, h \uplus h') \vDash \varphi_2$ 成立，其中 rng($h'$) 表示 $h'$ 的值域。

根据引理 4.1，我们定义如下归约函数。

定义*4.4*：归约函数 $T(s_x, h, \varphi)$ 是一个映射，它将栈 $s_x$、堆 $h$ 和 PSeqSL 公式 $\varphi$ 映射为 $\Sigma_1$ 序列谓词逻辑中的公式。归约可按图 (3) 中的方式归纳定义，其中 $\varphi = \varphi_1 \mathrel{-\!\!*} \varphi_2$。

给定 $s_x$ 和 $h$，公式 $T(s_x, h, \varphi_1 * \varphi_2)$ 由有限多个形如 $T(s_x, h, \varphi)$ 的项组成，因为分离合取中的隐式存在量词不会产生新的堆。根据引理 4.1，公式 $T(s_x, h, \varphi_1 \mathrel{-\!\!*} \varphi_2)$ 也由有限多个形如 $T(s_x, h, \varphi)$ 的项组成。此外，以下三个公式的真值：dom($h$) = ∅、dom($h$) = $s_x$($x$) 和 $s_x(x) \notin$Atom，均可确定；$h(s_x(x))$ 的值（它是一个序列）也可确定。因此，$T(s_x, h, \varphi)$ 仅由 $t_{\alpha_1} == t_{\alpha_2}$ 及其合取、析取和否定组成。根据定理 2.1，还可将其进一步化简为单个字方程。因此，PSeqSL 可归约为

$$
\begin{aligned}
T(s_x,h,t_{x_1}=t_{x_2}) &\triangleq t_{x_1}==t_{x_2}\\
T(s_x,h,t_{\alpha_1}==t_{\alpha_2}) &\triangleq t_{\alpha_1}==t_{\alpha_2}\\
T(s_x,h,\mathbf{false}) &\triangleq \mathbf{false}\\
T(s_x,h,\varphi_1\Rightarrow\varphi_2) &\triangleq T(s_x,h,\varphi_1)\Rightarrow T(s_x,h,\varphi_2)\\
T(s_x,h,\mathbf{emp}) &\triangleq \mathtt{dom}(h)=\varnothing\\
T(s_x,h,x\mapsto t_\alpha) &\triangleq \mathtt{dom}(h)=\{s_x(x)\}\wedge s_x(x)\notin \mathrm{Atom}\wedge h(s_x(x))==t_\alpha\\
T(s_x,h,\varphi_1*\varphi_2) &\triangleq \bigvee_{h=h_1\uplus h_2}(T(s_x,h_1,\varphi_1)\wedge T(s_x,h_2,\varphi_2))\\
T(s_x,h,\varphi_1-\!\!*\,\varphi_2) &\triangleq \bigwedge_{\substack{\mathtt{dom}(h_\varphi)\cap\mathtt{dom}(h)=\varnothing\\ \mathtt{dom}(h_\varphi)\subseteq\mathcal{D}\\ \mathtt{rng}(h_\varphi)\subseteq\mathcal{R}}}(T(s_x,h_\varphi,\varphi_1)\Rightarrow T(s_x,h_\varphi\uplus h,\varphi_2)),
\end{aligned}
$$

图3：归约函数 $T(s_x,\ h,\ \varphi)$

序列谓词逻辑的可满足性问题。文献 [19] 证明了序列谓词逻辑的 $\Sigma_1$片段是可判定的。

引理*4.2*：给定栈 $s_x$ 和堆 $h$。对于所有赋值 $s_\alpha$ 和公式 $\varphi$，命题 $(s_x,\ s_\alpha,\ h)\ \vDash\ \varphi$ 成立，当且仅当存在赋值 $(s'_x,\ s'_\alpha)$，使得 $(s'_x,\ s'_\alpha)\vDash T(s_x,\ h,\ \varphi)$ 成立，其中 $s'_x$ 和 $s'_\alpha$ 分别是程序变量和序列变量的赋值，$T(s_x,\ h,\ \varphi)$ 是根据定义4.4定义的从 PSeqSL 到 $\Sigma_1$ 序列谓词逻辑的归约。

注意，在根据定义4.4进行归约并应用定理 2.1 后，可以得到一个单一的字方程 $U==V$，其中含有常量（例如 $k$ 内容）和序列变量（例如 $k'$ 变量）。容易证明以下事实：$U==V$ 有解，当且仅当 $U==V$ 在具有 $k+k'$ 个生成元的自由幺半群上有解，其中前 $k$ 个生成元与 $k$ 常量相同，其余 $k'$ 个生成元在 $\mathbb{N}$ 中是新的。

为帮助读者直观理解引理4.2，我们列举以下两个例子。

例*4.1*：考虑公式 $\varphi=x_1\mapsto\alpha_1\circ x_3\wedge((x_1\mapsto\alpha_2\ -\!\!*\ x_1\mapsto\alpha_2*x_2\mapsto\alpha_3)*((\ )\ *\ (()\ ()$ 是否可满足，其中栈满足$\{(x_1,\ n_1),\ (x_2,\ n_2),\ (x_3,\ n_3)\}\subseteq s$，其中 $n_1,\ n_2$和 $n_3$是互不相同的整数，堆为 $h=\{(n_1,\ n_1\circ n_3),\ (n_2,\ n_2\circ n_3)\}$。赋值 $s_\alpha$ 尚未确定。

假设 $\varphi=\varphi_1*\varphi_2$，其中 $\varphi_1=x_1\mapsto\alpha_1\circ x_3$, $\varphi_2=(x_1\mapsto\alpha_1\vee x_1\mapsto\alpha_2)\ -\!\!*\ (x_1\mapsto\alpha_2*x_2\mapsto\alpha_3)$。我们得到图（4）中的公式。

假设 $\varphi_2=\varphi_{21}-\!\!*\,\varphi_{22}$，其中 $\varphi_{21}=x_1\mapsto\alpha_1\vee x_1\mapsto\alpha_2$，且 $\varphi_{22}=x_1\mapsto\alpha_2*x_2\mapsto\alpha_3$。我们有：$T(s_x,\ \{(n_2,\ n_2\circ n_3)\},\ \varphi_2)$

$$
\begin{aligned}
=\ \bigwedge_{\substack{\mathtt{dom}(h_{\varphi_2})\cap\{n_2\}=\varnothing\\ \mathtt{dom}(h_{\varphi_2})\subseteq\mathcal{D}\\ \mathtt{rng}(h_{\varphi_2})\subseteq\mathcal{R}}}&\big(T(s_x,h_{\varphi_2},\varphi_{21})\\
&\Rightarrow T(s_x,h_{\varphi_2}\uplus\{(n_2,n_2\circ n_3)\},\varphi_{22})\big),
\end{aligned}
$$

其中

$$
\begin{aligned}
&\mathcal{D}=B\ \cup\ \{n_1,n_2,n_3\},\quad B=\{m_1,m_2,m_3\},\\
&\mathcal{R}=\{[\![\alpha_1\circ x_3]\!]\sigma',[\![\alpha_1]\!]\sigma',[\![\alpha_2]\!]\sigma',[\![\alpha_3]\!]\sigma'\}\cup\{\varepsilon,[\![\overline{\beta}]\!]\sigma'\},
\end{aligned}
$$

且 $m_1,\ m_2,\ m_3$ 是 $\mathrm{Loc}\backslash(\mathrm{dom}(h)\cup s(L))$ 中的前三个值，项 $\beta$ 满足：

$$
[\![\overline{\beta}]\!]\sigma'\notin\{[\![\alpha_1\circ x_3]\!]\sigma',[\![\alpha_1]\!]\sigma',[\![\alpha_2]\!]\sigma',[\![\alpha_3]\!]\sigma'\}.
$$

定义域 $\mathrm{dom}(h_{\varphi_2})$ 包含许多有限的可能性。然而，只有两种可能性 $h_{\varphi_2}=\{(n_1,\ s'_\alpha(\alpha_1))\}$ 和 $h_{\varphi_2}=\{(n_1,\ s'_\alpha(\alpha_2))\}$ 可能满足该公式。因此，

$$
\begin{aligned}
&T(s_x,\{(n_2,n_2\circ n_3)\},\varphi_2)\\
=\ &(T(s_x,\{(n_1,s'_\alpha(\alpha_1)\},\varphi_{21})\\
&\Rightarrow T(s_x,\{(n_1,s'_\alpha(\alpha_1)),(n_2,n_2\circ n_3)\},\varphi_{22}))\\
&\wedge(T(s_x,\{(n_1,n_2\circ n_3)\},\varphi_{21})\\
&\Rightarrow T(s_x,\{(n_1,n_2\circ n_3),(n_2,n_2\circ n_3)\},\varphi_{22}))\\
=\ &T(s_x,\{(n_1,s'_\alpha(\alpha_1)),(n_2,n_2\circ n_3)\},\varphi_{22})\\
&\wedge T(s_x,\{(n_1,n_2\circ n_3),(n_2,n_2\circ n_3)\},\varphi_{22}).
\end{aligned}
$$

类似地，我们讨论满足公式 $\varphi_{22}$ 的模型的可能性。然后我们有：

$$
\begin{aligned}
&T(s_x,\{(n_1,s'_\alpha(\alpha_1)),(n_2,n_2\circ n_3)\},\varphi_{22})\\
=\ &\alpha_1==\alpha_2\wedge n_2\circ n_3==\alpha_3,\\
&T(s_x,\{(n_1,n_2\circ n_3),(n_2,n_2\circ n_3)\},\varphi_{22})\\
=\ &\alpha_2==\alpha_2\wedge n_2\circ n_3==\alpha_3.
\end{aligned}
$$

So,

$$
\begin{aligned}
&T(s_x,h,\varphi)\\
=\ &(n_1\circ n_3==\alpha_1\circ n_3)\wedge(\alpha_1==\alpha_2\wedge n_2\circ n_3==\alpha_3)\\
&\wedge(\alpha_2==\alpha_2\wedge n_2\circ n_3==\alpha_3).
\end{aligned}
$$

我们观察到 $T(s_x,\ h,\ \varphi)$ 可以满足。因此，给定 $s_x,\ h$，存在 $s_\alpha$ 使得 $(s_x,\ s_\alpha,\ h)\vDash\varphi$ 成立。

例*4.2*：考虑公式 $\varphi=x_1\mapsto\alpha_1\circ x_3\wedge((x_2\mapsto\alpha_1\vee x_2\mapsto\alpha_2)\ -\!\!*\ (x_1\mapsto\alpha_1*x_2\mapsto\alpha_3))$是否可满足

$$
\begin{aligned}
T(s_x,h,\varphi) \quad &= \quad \bigvee_{h=h_1 \uplus h_2} (T(s_x,h_1,\varphi_1) \wedge T(s_x,h_2,\varphi_2)) \\
&= \quad \big(T(s_x,\{\},\varphi_1) \wedge T(s_x,\{(n_1,n_1\circ n_3),(n_2,n_2\circ n_3)\},\varphi_2)\big) \\
&\quad \vee \big(T(s_x,\{(n_1,n_1\circ n_3)\},\varphi_1) \wedge T(s_x,\{(n_2,n_2\circ n_3)\},\varphi_2)\big) \\
&\quad \vee \big(T(s_x,\{(n_2,n_2\circ n_3)\},\varphi_1) \wedge T(s_x,\{(n_1,n_1\circ n_3)\},\varphi_2)\big) \\
&\quad \vee \big(T(s_x,\{(n_1,n_1\circ n_3),(n_2,n_2\circ n_3)\},\varphi_1) \wedge T(s_x,\{\},\varphi_2)\big) \\
&= \quad T(s_x,\{(n_1,n_1\circ n_3)\},\varphi_1) \wedge T(s_x,\{(n_2,n_2\circ n_3)\},\varphi_2).
\end{aligned}
$$

图4：例1中的构造

已知 $s=\{(x_1,\ n_1),\ (x_2,\ n_2),\ (x_3,\ n_3)\},\ h=\{(x_1,\ n_1\circ n_3)\}$。

根据定义定义4.4，我们有：

$$
\begin{aligned}
& T(s_x,h,\varphi) \\
= \quad & n_1\circ n_3 == \alpha_1\circ n_3 \wedge \big((\alpha_1 == \alpha_3 \wedge \alpha_1 == n_1\circ n_3) \\
& \wedge (\alpha_2 == \alpha_3 \wedge \alpha_1 == n_1\circ n_3)\big).
\end{aligned}
$$

我们注意到，$T(s_x,\ h,\ \varphi)$不可满足。因此，给定$s_x,\ h$，不存在满足$(s_x,\ s_\alpha,\ h) \vDash \varphi$ 的 $s_\alpha$。此处省略细节。

对于证明的第二步，事先并未给定堆 $h$。PSeqSL中公式的可满足性问题可通过以下引理归约为第一步中的问题。

引理*4.3*：给定栈 $s_x$ 和PSeqSL公式 $\varphi$，以下问题是可判定的：是否存在 $s_\alpha$ 和$h$，使得$(s_x,\ s_\alpha,\ h) \vDash \varphi$。

上述问题等价于以下问题：给定 $s_x$，是否存在 $s_\alpha$，使得$(s_x,\ s_\alpha,\ \{\}) \vDash \varphi \twoheadrightarrow$**为真**。因此我们可以得出引理4.3。

对于第三步，未给出栈 $s_x$。与论文[13], 类似，我们需要找出 $s_x$的一个有限范围，使得公式在该范围上为真，当且仅当公式在 $s_x$的无限范围上为真。

定义*4.5*：给定两个模型（$s_x,\ s_\alpha,\ h$），（$s'_x,\ s'_\alpha,\ h'$），集合 $P \subseteq$ PVar，以及集合 $S \subseteq$ SVar。关系（$s_x,\ s_\alpha,\ h$）$\approx_P$（$s'_x,\ s'_\alpha,\ h'$）成立，当且仅当存在一个同构 $r$：(Loc$^*$, $\circ$) $\to$ (Loc$^*$, $\circ$)，使得满足以下条件：

- 对每个 $\alpha_1,\ \alpha_2 \in S$，都有 $r(s_\alpha(\alpha_1)\circ s_\alpha(\alpha_2)) = r(s_\alpha(\alpha_1)) \circ r(s_\alpha(\alpha_2))$,
- $r$（**空值**）$=$ **空值**，$r$（#）$=$ #，并且对于每个 $x\in P$，我们都有$r$（$s_x$（$x$））$=s'_x$（$x$），
- 对于每个 $x\in P$，我们都有 $r$（$h$（$x$））$=h'$（$r$（$x$））,
- 对于每个 $\alpha\in S$，我们都有 $r$（$s_\alpha$（$\alpha$））$=s'_\alpha$（$\alpha$）。

命题*4.1*：给定两个模型（$s_x,\ s_\alpha,\ h$），（$s'_x,\ s'_\alpha,\ h'$），以及公式 $\varphi$满足（$s_x,\ s_\alpha,\ h$）$\approx_{\mathrm{FV}_x(\varphi)}$（$s'_x,\ s'_\alpha,\ h'$）。若（$s_x,\ s_\alpha,\ h$）$\vDash\varphi$，则（$s'_x,\ s'_\alpha,\ h'$）$\vDash\varphi$。

引理*4.4*：给定模型$(s_x,\ s_\alpha,\ h)$和公式$\varphi$。令 $B$ 为 Loc 中前$|\mathrm{FV}_x(\varphi)|$ 个位置。存在$(s'_x,\ s'_\alpha,\ h')$，使得 $s'_x$(PVar $\setminus$ FV$_x(\varphi)$) $\subseteq$ {**空值**, #}, $s'_x(\mathrm{FV}_x(\varphi)) \subseteq B \cup$ {**空值**, #}，以及 $(s_x,\ s_\alpha,\ h) \approx_{\mathrm{FV}_x(\varphi)} (s'_x,\ s'_\alpha,\ h')$。

由引理 4.2 至 4.4，我们可以得出结论：PSeqSL 的可满足性问题是可判定的，也就是说，定理 4.1 成立。

众所周知，序列谓词逻辑的 $\Sigma_1$ 片段的真值是可判定的 [19]。由于该片段由不受任何限制的否定构成，因此序列谓词逻辑 $\Sigma_1$ 片段中的实例与序列谓词逻辑 $\Pi_1$ 片段中的实例之间存在一一映射。因此，我们得到推论 4.1。

推论 *4.1*：序列谓词逻辑 $\Pi_1$ 片段的理论是可判定的。

定理 4.1 的证明表明，PSeqSL 具有小模型性质。PSeqSL 中公式的可满足性可以归约为序列逻辑 $\Sigma_1$片段中有限长度公式的可满足性。根据 PSeqSL 的小模型性质和推论 4.1，我们得到推论 4.2。

推论 *4.2*：PSeqSL 的$\Pi_1$ 片段的可满足性和有效性问题是可判定的，其中 PSeqSL 的 $\Pi_1$ 片段的形式为 $\forall^*_x\forall^*_\alpha.\varphi$，且 $\varphi$是无量词的。

## V. 不可判定片段

本节主要关注形式为 $(\forall^*_x\forall^*_\alpha \cap \forall^*_x\exists^*_x\exists^*_\alpha)$SeqSL$(*)$ 的不可判定片段。它是 $(\forall^*_x\forall^*_\alpha)$SeqSL$(*)$ 中的公式与 $(\forall^*_x\exists^*_x\exists^*_\alpha)$SeqSL$(*)$ 公式的合取，其中 $\exists^*_\alpha$ 表示存在 0 个或多个序列变量存在量词，$\exists^*_x$ 表示存在 0 个或多个程序变量存在量词。$\forall^*_\alpha$ 和 $\forall^*_\alpha$ 的含义类似。我们给出本节的以下主要结果。

定理 *5.1*（*SeqSL* 的不可判定片段）：当语言受到如下限制时，其可满足性问题是不可判定的：

$$
\begin{aligned}
t_x \quad &::= \quad \mathbf{nil} \mid \# \mid x \\
t_\alpha \quad &::= \quad \varepsilon \mid t_x \mid \alpha \mid t_\alpha \circ t_\alpha \\
\psi \quad &::= \quad t_x = t_x \mid t_\alpha == t_\alpha \mid x \hookrightarrow t_\alpha \mid \mathbf{false} \mid \psi \Rightarrow \psi \\
&\qquad \mid \mathbf{emp} \mid \psi * \psi \\
\varphi \quad &::= \quad \forall^*_x\forall^*_\alpha.\psi \wedge \forall^*_x\exists^*_x\exists^*_\alpha.\psi,
\end{aligned}
$$

其中 $\varphi$ 是一个句子。

该片段的模型在定义 3.2 中给出。

我们将问题归约自双计数器明斯基机的停机问题。在介绍归约的细节之前，我们先回顾双计数器明斯基机的定义。

定义 *5.1*（双计数器明斯基机）：设 $M$ 是一台包含 $n \geq 1$ 条指令的米ンス基机。该机器 $M$ 有两个计数器 $C_1$ 和 $C_2$。指令定义如下：

1) $I$： $C_j := C_j + 1$；转到 $k$，2) $I$：若 $C_j = 0$，则转到 $k_1$，否则（$C_j := C_j - 1$；转到 $k_2$），3) $n$：停机，

其中 $j \in [1,2]$，$I \in [1, n-1]$，和 $k$，$k_1$，$k_2 \in [1,n]$。若存在如下形式的运行，则机器 $M$ 停机

$$(I_0, c_0^1, c_0^2),\ (I_1, c_1^1, c_1^2), \ldots, (I_m, c_m^1, c_m^2),$$

使得 $(I_i,\ c_i^1,\ c_i^2) \in [1,n] \times \mathbb{N}^2 (i \in [1,m])$，$I_0 = 1$，$I_m = n$，且 $c_0^1 = c_0^2 = 0$。该运行遵循上述定义的指令。

类型 1 的指令集合（记作 $\mathcal{I}_1$）由元组 $(k_0,\ c_j,\ c'_j,\ k)$ 组成；类型 2 的指令集合（记作 $\mathcal{I}_2$）由元组 $(k_0,\ c_j,\ c'_j,\ k_1,\ k_2)$ 组成，其中 $k_0$ 表示当前指针，$c_j,\ c'_j \in [1,2]$ 表示两个计数器（$c_j \neq c'_j$），$k$，$k_1$，$k_2$ 表示后继指针。

判定机器 $M$ 是否停机的问题已知是不可判定的 [28]。

我们首先说明，定理 5.1 中定义的不可判定片段并不容易得到。一般而言，序列逻辑片段和分离逻辑片段的可判定性结果都限制了 SeqSL 中不可判定片段的形式。

自由半群中的可判定性结果表明，$\Sigma_2$ 片段 [21]，的理论、$\Pi_2$ 片段 [20] 的正向理论以及字方程的 $\Sigma_3$ 片段 [20] 都是不可判定的。分离逻辑中的结果表明，分离逻辑的 $\Sigma_3$ 片段的可满足性问题是不可判定的。在这种情况下，我们可以得到事实 5.1。

事实 *5.1*：在具有前束范式的 SeqSL 片段中，如果程序变量上的量词交替次数不少于 3 次，或序列变量上的量词交替次数不少于 2 次，那么所得片段的可满足性问题是不可判定的。例如，$(\exists^*_x \exists^*_\alpha \forall^*_\alpha)$SeqSL、$(\exists^*_x \forall^*_\alpha \exists^*_x)$SeqSL 和 $(\exists^*_x \forall^*_x \forall^*_\alpha \exists^*_x)$SeqSL 的可满足性问题都是不可判定的。

为了得到一个包含否定且非平凡的 SeqSL 不可判定片段，我们必须将程序变量上的量词交替次数限制为少于 3 次，并将序列变量上的量词交替次数限制为少于 2 次。

为了证明定理 5.1，我们首先讨论归约的编码。其一般结构为小树，如图（5）所示。

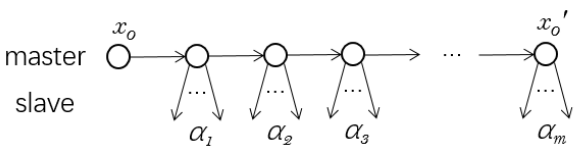


图 5：小树的结构

我们讨论小树的编码。小树是一种鱼骨状结构，其“骨骼”朝向相反的方向。小树由一条主分支和若干条从枝组成，这些从枝连接到主分支上的节点。每条从枝的深度均为 1，且其端点不指向主分支中的任何节点。与 [12] 中定义的列表 $x \xrightarrow{\circlearrowleft}{}^{+} y$ 以及 [16], 中定义的鱼骨堆的编码类似，我们按以下方式对小树进行编码：

1）每个节点的前驱至多有一个。该性质由公式 $\Psi_1$表示；2）第一个主节点没有前驱。该性质由公式 $\Psi_2$表示；3）第一个主节点已分配，且最后一个节点不指向下一个主节点。该性质由公式 $\Psi_3$ 表示（无需将最后一个节点的前驱设为 1）；4）除最后一个节点外，每个节点之后都有另一个已分配节点。该性质由公式 $\Psi_4$表示。

从节点 $x_0$ 到节点 $x'_0$ 的小树（此处假设 $x_0 \neq x'_0$）可表示为

$$\texttt{sapling}(x_0, x'_0) \triangleq \Psi_1 \wedge \Psi_2 \wedge \Psi_3 \wedge \Psi_4. \tag{7}$$

其中 $\Psi_1$，$\Psi_2$，$\Psi_3$ 和 $\Psi_4$ 在图（4）中定义。

很容易证明，谓词 sapling($x_0$，$x'_0$) 的形式为 $(\forall^*_x \forall^*_\alpha \cap \forall^*_x \exists^*_x \exists^*_\alpha)\Psi$。

注意，公式（7）中的编码只是一个一般性思路。它并不禁止从节点中的内容指向其他节点。不过，这不会影响主要结果，因为这些内容仅由**空值** 构成，这将在公式（8）中展示。

还要注意，谓词 Sapling 不仅对仅由一棵小树构成的图进行编码，也对由一棵小树和圆构成的图进行编码。没有必要移除所有这些圆，因为这不会影响归约的正确性。引理 5.1 中对此进行了证明。

引理 *5.1*：根据公式（7）定义的谓词小树($x_0$，$x'_0$)，当且仅当存在一个满足小树($x_0$，$x'_0$)且其中不含圆的模型 $\sigma'$ 时，模型 $\sigma$ 可以满足该谓词。

引理 5.1 的证明见附录。现在我们证明定理 5.1。

定理 *5.1* 的证明。我们将详细介绍该归约，其示意图见图（7）。

主分支是从起始主节点到末尾主节点的最长路径。每个主节点指向 0 个或多个从节点**空值**$^*$。主节点的周期为 4。每个周期表示一次运行状态。在第$i$个周期（$1 \leq i \leq m$）中，第一个主节点表示一个分隔符，用于分隔每个运行状态。它没有附加从节点。第二至第四个主节点表示满足 $I = k_i$，$C_1 = c_i^1 + 1$，和$C_2 = c_i^2 + 1$的第 $i$个三元组（$I$，$C_1$，$C_2$）。这 3 个节点所指向的从节点分别为**空值**$^{k_i}$，**空值**$^{c_i^1+1}$，和**空值**$^{c_i^2+1}$。第一个

$$
\begin{array}{lcl}
\Psi_1 & \triangleq & \forall_x x_1 \forall_x x_2 \forall_x x_3 \forall_x x_4 \forall_\alpha \alpha_1 \forall_\alpha \alpha_2.\, (x_1 \hookrightarrow x_3 \circ \alpha_1 * x_2 \hookrightarrow x_4 \circ \alpha_2 \Rightarrow x_3 \neq x_4) \\
\Psi_2 & \triangleq & \forall_x x_1 \forall_\alpha \alpha.\, \neg(x_1 \hookrightarrow x_0 \circ \alpha) \\
\Psi_3 & \triangleq & (\exists_x x_1.\, x_0 \hookrightarrow x_1 \circ \varepsilon) \wedge (\exists_\alpha \alpha.\, x_0' \hookrightarrow \varepsilon \circ \alpha) \\
\Psi_4 & \triangleq & \forall_x x_1 \forall_x x_2 \exists_x x_3 \exists_\alpha \alpha_1 \exists_\alpha \alpha_2.\, \big((x_1 \hookrightarrow x_2 \circ \alpha_1 \wedge x_2 \neq x_0') \Rightarrow x_2 \hookrightarrow x_3 \circ \alpha_2\big).
\end{array}
$$

图 6： $\Psi_1$， $\Psi_2$， $\Psi_3$ 和 $\Psi_4$ 的定义

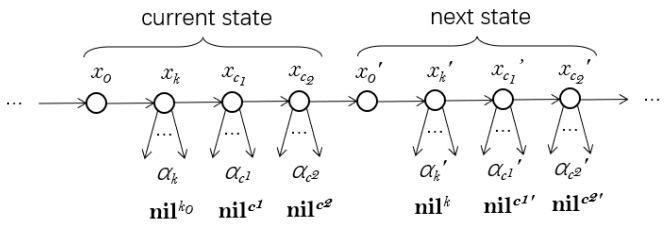


图 7：归约的结构

周期表示初始状态，而最后一个周期表示最终状态。

我们注意到，小树中两个计数器的值比运行中计数器的实际值大 1。这是因为我们需要将这些节点与每个周期的第一个节点区分开来，后者没有连接从节点。

一般来说，归约公式由以下三个部分组成。

1) 构造小树的基本结构，用公式 $\Phi_1 = \forall_x^* \forall_\alpha^*.\Phi_1'$ 表示。2) 初始化初始状态和最终状态，用公式 $\Phi_2 = \exists_x^* \exists_\alpha^*.\Phi_2'$ 表示。3) 根据当前指令，对从一个状态到另一个状态的转移进行编码，用公式 $\Phi_3 = \forall_x^* \exists_x^* \exists_\alpha^*.\Phi_3'$ 表示。

该公式的形式为

$$
\begin{array}{rcl}
\Phi & = & \Phi_1 \wedge \Phi_2 \wedge \Phi_3 \\
& = & \forall_x^* \forall_\alpha^*.\Phi_1' \wedge \exists_x^* \exists_\alpha^*.\Phi_2' \wedge \forall_x^* \exists_x^* \exists_\alpha^*.\Phi_3',
\end{array}
\tag{8}
$$

or

$$
\Phi \quad = \quad \forall_x^* \forall_\alpha^*.\Phi_1' \wedge \forall_x^* \exists_x^* \exists_\alpha^*.(\Phi_2' \wedge \Phi_3').
$$

详细构造见附录。

对于每个双计数器明斯基机 M，设 $\Phi$ 为公式（8）中定义的公式。如果机器 M 停机，则存在一次有限运行。容易证明，存在一棵有限小树，其第一个周期编码初始状态，最后一个周期编码最终状态，并且除最终状态外的每个状态都可以通过相应的指令转换为后继状态。

对于证明的另一部分，我们假设满足公式$\Phi$，该公式具有公式（8）中定义的形式。$\Phi$ 的模型 $\sigma$可能包含圆。根据引理 5.1，我们可以得到一个满足$\Phi$ 且不含圆的模型 $\sigma'$。□

## VI. 结论

本文提出了序列堆分离逻辑，它结合了序列谓词逻辑和分离逻辑。它能够描述以下性质，而这些性质仅靠序列谓词逻辑或分离逻辑无法描述，或难以描述。

- 程序中的序列操作，例如链表反转和查找。
- 程序中与变长序列对应的性质，例如栈、队列以及出度无界的图。
- 程序中的多级数据结构，例如 Windows 存储中的数据结构，以及基于块的云存储系统中的数据结构。

此外，我们发现了前束范式 SeqSL 可判定片段与不可判定片段之间的界限。我们证明了以下两个可判定性结果： $\Sigma_1$ 片段和 $\Pi_1$ 片段的可满足性问题是可判定的，而($\forall_x^* \forall_\alpha^* \cap \forall_x^* \exists_x^* \exists_\alpha^*$)SeqSL($*$)的可满足性问题是不可判定的。作为推论，程序变量上有 3 次量词交替或序列变量上有 2 次量词交替的片段都是不可判定的。

未来我们将开展以下工作。

- 寻找序列堆分离逻辑的其他可判定与不可判定片段之间的界限，例如量化变量数量以及归纳谓词方面的界限。
- 深入研究序列堆分离逻辑的表达能力。
- 为序列堆分离逻辑构建一个证明系统。
- 为序列堆分离逻辑片段实现形式化验证工具。

## 参考文献


[1] J. C. Reynolds，《分离逻辑：一种用于共享可变数据结构的逻辑》，载于第 *17* 届 *IEEE* 计算机科学逻辑年会（*LICS*）论文集。IEEE，2002年，第 55–74 页。[2] M. J. Parkinson 和 G. M. Bierman，《分离逻辑、抽象与继承》，载于*ACM SIGPLAN* 通讯，第 43 卷，第 1 期，第 75–86 页，2008年。[3] K. Batz、B. L. Kaminski、J.-P. Katoen、C. Matheja 和 T. Noll，《定量分离逻辑：一种用于推理概率指针程序的逻辑》，载于*ACM* 编程语言会议论文集（*POPL*），第 3 卷，POPL 专刊，第 1–29 页，2019年。[4] B. Biering、L. Birkedal 和 N. Torp-Smith，《双超教义、高阶分离逻辑与抽象》，载于 *ACM* 编程语言与系统汇刊（*TOPLAS*），第 29 卷，第 5 期，第 24–es 页，2007年。

[5] J. Brotherston、C. Fuhs、J. A. N. P´erez 和 N. Gorogiannis，《带归纳谓词的分离逻辑可满足性判定过程》，载于第二十三届 EACSL 计算机科学逻辑年会联合会议（CSL）与第二十九届 ACM/IEEE 计算机科学逻辑年会（LICS），2014年，第 1–10 页。[6] C. Barrett、C. L. Conway、M. Deters、L. Hadarean、D. Jovanovi´c、T. King、A. Reynolds 和 C. Tinelli，《Cvc4》，载于计算机辅助验证国际会议（CAV）。施普林格，2011年，第 171–177 页。[7] M. Berzish、V. Ganesh 和 Y. Zheng，《Z3str3：一种采用理论感知启发式的字符串求解器》，载于2017年计算机辅助设计形式化方法会议（FMCAD）。IEEE，2017年，第 55–59 页。[8] T. Kutsia 和 B. Buchberger，《带序列变量和序列函数符号的谓词逻辑》，载于数学知识管理国际会议（MKM）。施普林格，2004年，第 205–219 页。[9] Z. Jin、B. Zhang、T. Cao、Y. Cao 和 H. Wang，《通过分离逻辑推理基于块的云存储系统》，载于理论计算机科学（TCS），第 936 卷，第 43–76 页，2022年。[10] S. Demri、´E. Lozes 和 A. Mansutti，《在命题分离逻辑中添加可达性谓词的影响》，载于软件科学与计算结构基础国际会议（FoSSaCS）。施普林格，2018年，第 476–493 页。[11] J. Berdine、C. Calcagno 和 P. W. O’hearn，《分离逻辑的一个可判定片段》，载于软件技术与理论计算机科学基础国际会议（FSTTCS）。施普林格，2004年，第 97–109 页。[12] R. Brochenin、S. Demri 和 E. Lozes，《论全能魔杖》，载于信息与计算（IANDC），第 211 卷，第 106–137 页，2012年。[13] C. Calcagno、H. Yang 和 P. W. O’hearn，《数据结构空间断言语言的可计算性与复杂度结果》，载于软件技术与理论计算机科学基础国际会议（FSTTCS）。施普林格，2001年，第 108–119 页。[14] A. Reynolds、R. Iosif 和 C. Serban，《在分离逻辑的 Bernays-Sch¨ önfinkel-Ramsey 片段中进行推理》，载于验证、模型检验与抽象解释国际会议（VMCAI）。施普林格，2017年，第 462–482 页。[15] S. Demri、D. Galmiche、D. Larchey-Wendling 和 D. M´ery，《带一个量化变量的分离逻辑》，载于计算系统理论，第 61 卷，第 2 期，第 371–461 页，2017年。[16] S. Demri 和 M. Deters，《双变量分离逻辑及其内圈》，载于ACM 计算逻辑汇刊（TOCL），第 16 卷，第 2 期，第 1–36 页，2015 年。[17] J. Pagel 和 F. Zuleger，《强分离逻辑》，载于 ACM 编程语言与系统汇刊（TOPLAS），第 44 卷，第 3 期，第 1–40 页，2022年。[18] J. Karhum¨aki、F. Mignosi 和 W. Plandowski，《字方程对语言和关系的可表达性》，载于ACM学报（JACM），第 47 卷，第 3 期，第 483–505 页，2000年。[19] G. S. Makanin，《自由半群中方程可解性问题》，载于数学文集，第 145 卷，236，1977年 2，147 no. pp. – 。[20] V. G. Durnev，《自由半群正向 ∀∃3理论的不可判定性》，载于西伯利亚数学杂志，第 36 卷，第 5 期，第 917–929 页，1995年。[21]J. D. Day、V. Ganesh、P. He、F. Manea 和 D. Nowotka，《字方程的可满足性：可判定与不可判定理论》，载于可达性问题国际会议（RP）。施普林格，2018年，第 15–29 页。[22] J. M. Va˘zenin 和 B. V. Rozenblat，《自由可数生成半群正向理论的可判定性》，载于《苏联数学文集》，第 44 卷，第 1 期，第 109 页，1983年。[23] Y. Jing、H. Wang、Y. Huang、L. Zhang、J. Xu 和 Y. Cao，《一种用于描述信息物理系统海量数据存储管理的建模语言》，载于并行与分布式计算杂志（JPDC），第 103 卷，第 113–120 页，2017年。[24] B. Zhang、Z. Jin、H. Wang 和 Y. Cao，《基于分离逻辑的云块存储验证工具》，载于软件学报，第 33 卷，第 6 期，第 2264–2287 页，2022年。[25] J. C. Reynolds，《关于共享可变数据结构的直觉主义推理》，载于《计算机科学千年展望》，第 2 卷，第 1 期，第 303–321 页，2000年。[26] J. Berdine、C. Calcagno 和 P. W. O’hearn，《使用分离逻辑进行符号执行》，载于亚洲编程语言与系统研讨会 *(APLAS)*。施普林格，2005年，第 52–68 页。[27] T. Kutsia，《求解含序列变量和序列函数的方程》，符号计算杂志 *(JSC)*，第 42 卷，第 3 期，第 352–388 页，2007年。[28] M. Minsky，《计算：有限与无限机器》。JSTOR，1968年。

# 附录

**定理 *2.1* 的证明概要**：对于定义 2.3 中定义的每个布尔组合 $\varphi$，都存在一个单一的字方程$T(\varphi) \triangleq U == V$，它以如下归纳方式等价于 $\varphi$：

$$
\begin{array}{lll}
T(t_1 == t_2) & \triangleq & t_1 == t_2 \\
T(t_1 == t_2 \wedge t_1' == t_2') & \triangleq & F(t_1, t_2) == F(t_1', t_2') \\
T(t_1 == t_2 \vee t_1' == t_2') & \triangleq & T_0(t_1 \circ t_2' == t_2 \circ t_2' \vee t_2 \circ t_1' == t_2 \circ t_2') \\
T(\neg\, t_1 == t_2) & \triangleq & \exists\beta\exists\beta'\exists\beta''.(\bigvee_{n\in\Sigma} t_1 == t_2 \circ n \circ \beta) \vee (\bigvee_{n\in\Sigma} t_2 == t_1 \circ n \circ \beta) \\
 & & \quad \vee (\bigvee_{\substack{n,\, n' \in \Sigma \\ n \neq n'}} t_1 == \beta \circ n \circ \beta' \wedge t_2 == \beta \circ n' \circ \beta'') \\
T_0(u == v \vee u' == v) & \triangleq & \exists\beta\exists\beta'.X == \beta \circ Y \circ \beta',
\end{array}
$$

其中，$\Sigma$ 是生成集合，并且

$$
\begin{array}{rcll}
X & = & G(u \circ u')^2 \circ u \circ G(u \circ u')^2 \circ u' \circ G(u \circ u')^2 & \\
Y & = & G(u \circ u')^2 \circ v \circ G(u \circ u')^2 & \\
F(t_1, t_2) & = & t_1 \circ n \circ t_2 \circ t_1 \circ n' \circ t_2, & n \neq n' \\
G(t) & = & t \circ n \circ t \circ n', & n \neq n'
\end{array}
$$

□

**引理 *5.1* 的证明概要**：引理的“如果”部分是显然的。对于“仅当”部分，我们假设$\sigma \vDash$ 小树($x_0$，$x_0'$)。只需证明以下两个事实。

- 不可能 $\sigma$ 中的所有连通分量都是圆，
- 如果 $\sigma$中有一个或多个圆，那么令 $\sigma'$ 为仅在 $\sigma$ 中包含一棵小树且不含圆的模型。我们可以得到 $\sigma' \vDash$ 小树($x_0$，$x_0'$)。

首先证明第一个事实。我们假设 $\sigma$ 中的所有连通分量都是圆。由此可以推出每个主节点都有一个前驱。这与 $\Psi_2$ 相矛盾，后者声称第一个节点 $x_0$上没有前驱。

然后我们证明第二个事实。根据第一个事实，$\sigma$中存在一棵小树。根据 $\Psi_1$ 所述，每个主节点至多有 2 个前驱，因此这棵小树不会与其他圆相交。令$\sigma$中的小树为 $\sigma'$，并令 $\sigma$中的所有圆为 $\sigma''$。我们可以得到 $\sigma = \sigma' \uplus \sigma''$。因此，命题 $\sigma' \vDash$ sapling($x_0$，$x_0'$)成立，因为移除圆不会影响 $\Psi_1$，$\Psi_2$，$\Psi_3$和 $\Psi_4$的真假。□

**定理 *5.1* 中的构造**。第 1 部分中的公式 $\Phi_1$ 恰好具有式（7）中定义的 $\Psi_1 \wedge \Psi_2$ 的形式。它可以写成如下形式：

$$
\Phi_1 = \forall_x x_1 \forall_x x_2 \forall_x x_3 \forall_x x_4 \forall_\alpha \alpha_1 \forall_\alpha \alpha_2.\ (x_1 \hookrightarrow x_3 \circ \alpha_1 * x_2 \hookrightarrow x_4 \circ \alpha_2 \Rightarrow x_3 \neq x_4) \wedge \forall_x x_1 \forall_\alpha \alpha.\ \neg(x_1 \hookrightarrow x \circ \alpha). \tag{9}
$$

第 2 部分中的公式 $\Phi_2$ 可以用以下方式表示。

$$
\begin{array}{rcl}
\Phi_2 & = & \exists_x x_0 \exists_x x_k \exists_x x_{c^1} \exists_x x_{c^2} \exists_x x_1 \exists_x x_0' \exists_x x_k' \exists_x x_{c^1}' \exists_x x_{c^2}' \exists_\alpha \alpha_{c^1}' \exists_\alpha \alpha_{c^2}'. \\
 & & \mathtt{InitState}(x_0, x_k, x_{c^1}, x_{c^2}, x_1) * \mathtt{FinState}(x_0', x_k', x_{c^1}', x_{c^2}', \alpha_{c^1}', \alpha_{c^2}').
\end{array} \tag{10}
$$

谓词 InitState($x_0$，$x_k$，$x_{c^1}$，$x_{c^2}$，$x_1$) 编码初始状态 ($I_0$，$c_0^1$，$c_0^2$) =(1，0，0)（对应于图（7）中三个主节点 $x_k$，$x_{c^1}$，$x_{c^2}$ 上的从节点），并且

FinState($x'_0$, $x'_k$, $x'_{c^1}$, $x'_{c^2}$, $\alpha'_{c^1}$, $\alpha'_{c^2}$) 编码最终状态 ($n$, $c^1_m$, $c^2_m$)。状态 ($n$, $c^1_m$, $c^2_m$) 对应于图（7）中三个主节点 $x'_k$, $x'_{c^1}$, $x'_{c^2}$ 上的从节点。这两个谓词定义如下：

$$\begin{aligned}&\texttt{InitState}(x_0, x_k, x_{c^1}, x_{c^2}, x_1)\\ \triangleq\ & x_0 \hookrightarrow x_k \circ \varepsilon * x_k \hookrightarrow x_{c^1} \circ \mathbf{nil} * x_{c^1} \hookrightarrow x_{c^2} \circ \mathbf{nil} * x_{c^2} \hookrightarrow x_1 \circ \mathbf{nil},\end{aligned} \tag{11}$$

$$\begin{aligned}&\texttt{FinState}(x'_0, x'_k, x'_{c^1}, x'_{c^2}, \alpha'_{c^1}, \alpha'_{c^2})\\ \triangleq\ & x'_0 \hookrightarrow x'_k \circ \varepsilon * x'_k \hookrightarrow x'_{c^1} \circ \mathbf{nil}^n * x'_{c^1} \hookrightarrow x'_{c^2} \circ \alpha'_{c^1} \circ \mathbf{nil} * x'_{c^2} \hookrightarrow \varepsilon \circ \alpha'_{c^2} \circ \mathbf{nil}.\end{aligned}$$

注意，当给定双计数器明斯基机时，$n$是固定的。

在构造公式 $\Phi_3$之前，我们需要形式化下面展示的一些附加性质。

序列 $\alpha$ 的形式为 **空值**$^*$：

$$\texttt{ini}(\alpha) \triangleq \mathbf{nil} \circ \alpha == \alpha \circ \mathbf{nil}. \tag{12}$$

形式为 **空值**$^*$ 的序列 $\alpha_1$ 比同样形式的序列 $\alpha_2$ 长一个元素：

$$\alpha_1 == \alpha_2 \circ \mathbf{nil}.$$

借助 $\Psi_4$ 中的思路，我们可以按如下方式构造 $\Phi_3$：

$$\begin{aligned}\Phi_3 \quad = \quad & \forall_x x_0 \forall_x x_k \forall_x x_{c^1} \forall_x x_{c^2} \forall_x x'_0.\ \exists_x x'_k \exists_x x'_{c^1} \exists_x x'_{c^2} \exists_x x'_1 \exists_\alpha \alpha_k \exists_\alpha \alpha_{c^1} \exists_\alpha \alpha_{c^2} \exists_\alpha \alpha'_k \exists_\alpha \alpha'_{c^1} \exists_\alpha \alpha'_{c^2}.\\ & \big(\texttt{CurrState}(x_0, x_k, x_{c^1}, x_{c^2}, x'_0, \alpha_k, \alpha_{c^1}, \alpha_{c^2}) \wedge \neg \texttt{FinState}(x_0, x_k, x_{c^1}, x_{c^2}, \alpha_{c^1}, \alpha_{c^2})\big)\\ & \Rightarrow \texttt{NextState}(x_0, x_k, x_{c^1}, x_{c^2}, x'_0, x'_k, x'_{c^1}, x'_{c^2}, x''_0, \alpha_k, \alpha_{c^1}, \alpha_{c^2}, \alpha'_k, \alpha'_{c^1}, \alpha'_{c^2}).\end{aligned} \tag{13}$$

在 $\Phi_3$ 中，谓词 CurrState 表示从没有从节点的主节点开始的当前状态，FinState 表示公式（11）中定义的最终状态，而 NextState 表示从一个状态到另一个状态的转移。公式 $\Phi_3$ 编码了以下事实：对于每个非最终状态，都存在一个可从当前状态转移到的后继状态。谓词 CurrState 和 NextState 定义如下：

$$\begin{aligned}&\texttt{CurrState}(x_0, x_k, x_{c^1}, x_{c^2}, x'_0, \alpha_k, \alpha_{c^1}, \alpha_{c^2})\\ \triangleq\ & x_0 \hookrightarrow x_k \circ \varepsilon * x_k \hookrightarrow x_{c^1} \circ \alpha_k * x_{c^1} \hookrightarrow x_{c^2} \circ \alpha_{c^1} \circ \textbf{空值} * x_{c^2} \hookrightarrow x'_0 \circ \alpha_{c^2} \circ \textbf{空值},\end{aligned}$$

$$\begin{aligned}&\texttt{NextState}(x_0, x_k, x_{c^1}, x_{c^2}, x'_0, x'_k, x'_{c^1}, x'_{c^2}, x''_0, \alpha_k, \alpha_{c^1}, \alpha_{c^2}, \alpha'_k, \alpha'_{c^1}, \alpha'_{c^2})\\ \triangleq\ & \text{初始}(\alpha_k, \alpha_{c^1}, \alpha_{c^2}, \alpha'_k, \alpha'_{c^1}, \alpha'_{c^2}) \wedge \text{CurrState}(x'_0, x'_k, x_{c^1}, x_{c^2}, x_0, \alpha'_k, \alpha'_{c^1}, \alpha'_{c^2}) \wedge \text{后继}(\alpha_k, \alpha_{c^1}, \alpha_{c^2}, \alpha'_k, \alpha'_{c^1}, \alpha'_{c^2}),\end{aligned}$$

谓词初始定义如下。谓词 ini 已在公式（12）中定义。

$$\begin{aligned}&\texttt{Init}(\alpha_k, \alpha_{c^1}, \alpha_{c^2}, \alpha'_k, \alpha'_{c^1}, \alpha'_{c^2})\\ \triangleq\ & \texttt{ini}(\alpha_k) \wedge \texttt{ini}(\alpha_{c^1}) \wedge \texttt{ini}(\alpha_{c^2}) \wedge \texttt{ini}(\alpha'_k) \wedge \texttt{ini}(\alpha'_{c^1}) \wedge \texttt{ini}(\alpha'_{c^2}).\end{aligned}$$

谓词后继定义如下：

$$\begin{aligned}&\texttt{Next}(\alpha_k, \alpha_{c^1}, \alpha_{c^2}, \alpha'_k, \alpha'_{c^1}, \alpha'_{c^2})\\ \triangleq\ & \bigwedge_{(k_0, c_j, c'_j, k) \in \mathcal{I}_1} \Big(\alpha_k == \mathbf{nil}^{k_0} \wedge \alpha'_{c^j} == \alpha_{c^j} \circ \mathbf{nil} \wedge \alpha'_{c^{j'}} == \alpha_{c^{j'}} \wedge \alpha'_k == \mathbf{nil}^k\Big)\\ \wedge\ & \bigwedge_{(k_0, c_j, c'_j, k_1, k_2) \in \mathcal{I}_2} \Big(\alpha_k == \mathbf{nil}^{k_0} \wedge \big(\alpha_{c^j} == \mathbf{nil} \Rightarrow (\alpha'_{c^1} == \alpha_{c^1} \wedge \alpha'_{c^2} == \alpha_{c^2} \wedge \alpha'_k == \mathbf{nil}^{k_1})\big)\\ & \qquad \wedge \big(\neg(\alpha_{c^j} == \mathbf{nil}) \Rightarrow (\alpha'_{c^j} \circ \mathbf{nil} == \alpha_{c^j} \wedge \alpha'_{c^{j'}} == \alpha_{c^{j'}} \wedge \alpha'_k == \mathbf{nil}^{k_2})\big)\Big).\end{aligned}$$

在公式后继中，集合 $\mathcal{I}_1$ 和 $\mathcal{I}_2$ 已在定义 5.1 中定义。该归约可按照公式（8）至（10）和（13）进行。□