arXiv:2301.07967v1 [cs.LO] 19 Jan 2023

基于视图的 PSO 公理推理（扩展版）*

Lara Bargmann 和 Heike Wehrheim

奥尔登堡大学计算机科学系，德国

摘要。弱内存模型描述了现代多核架构上并发程序的语义。并发程序的推理技术，例如 Owicki-Gries 风格证明演算，必须以这类语义为基础，因此需要针对每种新的内存模型重新开发。近期，有人提出了一种更统一的推理方法，它基于若干核心 $axioms$ 构建正确性证明。这使得我们能够证明内存模型的程序正确性$independent$，并通过证明特定内存模型满足证明所需的全部公理，将这些证明迁移到相应的内存模型。该公理化以线程 $views$ 的概念为基础，将其作为语义中的一等元素。

本文研究这种公理化推理方法对部分存储顺序（PSO）内存模型的适用性。由于 PSO 的标准语义并非基于视图，我们首先为 PSO 提供一种基于视图的语义，并证明它与标准语义等价。随后，我们证明这种新的基于视图的语义满足除一条之外的所有公理。缺失的公理涉及内存模型的消息传递（MP）能力，而 PSO 并不保证这种能力。因此，只有不使用 MP 公理的证明才能迁移到 PSO。我们通过证明一个使用栅栏确保消息传递的微基准测试的正确性，来展示这一推理技术。

关键词：弱内存模型 · 公理化推理 · 基于视图的语义 · Owicki-Gries 证明演算。

1 引言

在多核架构上，并发程序的语义会偏离通常假设的顺序一致性（Lamport [28]）。顺序一致性保证并发程序按照线程内的程序顺序交错执行语句。相比之下，现代多核架构上的并发程序行为看起来像是程序语句被重新排序，例如允许写-写重排序（针对不相交的变量）。弱内存模型描述了此类架构上程序的语义。

* Bargmann 和 Wehrheim 获得 DFG-WE2290/14-1 的资助。

如今，已有针对多种不同架构（例如 TSO [32], 、PSO 和 Power [2], 、ARM [17]）以及编程语言（例如 C11 [8]）的弱内存模型。其语义采用公理式或操作风格进行规范。操作风格更适用于 Hoare 风格证明演算 [20] 等验证方法，以及由 Owicki 和 Gries [31] 提出的、面向并行程序的扩展；这些方法需要构造证明大纲并对程序语句进行推理。近期，已有多种基于视图概念的操作语义被提出 [16, 11, 22, 10, 23, 9]。视图是并发程序中特定于线程的概念——简单来说，它指定线程在某个特定状态下可以读取（即观察到）的共享变量的值[1]。基于视图的语义很适合 Owicki-Gries 风格推理 [31], ：它以描述线程对共享变量可能视图的基于视图的断言，取代针对程序变量的标准断言。然而，每出现一种新的弱内存模型，就需要构建一种新的推理证明演算。

为缓解这一问题，Doherty 等人 [15] 提出了一种通用推理技术，用于采用基于视图的操作语义的弱内存模型。该技术的核心是关于内存模型所生成迁移系统的一系列公理。这些公理详细说明了读写操作（例如，对不同程序变量进行操作的动作具有半交换性）以及栅栏指令的性质。随后，只需依据这些公理，而无需依赖具体的内存模型，即可对并发程序进行正确性证明。一旦证明某个内存模型满足证明中采用的所有公理，这些证明便可迁移到该特定内存模型上。

本文研究了这种方法对内存模型部分存储顺序（PSO） [21]的适用性。许多研究探讨了这一内存模型，提出了无状态模型检查 [1], ，证明了可达性问题的可判定性 [7] ，或证明了测试（或一致性）问题的 NP 困难性 [18]。除公理语义 [3], 外，PSO 也有操作语义，但它并不容易用于定义视图，尤其难以定义视图极大性——这是 [15]中公理化推理的核心概念。因此，我们首先为 PSO 开发一种新的语义定义，并通过模拟证明它与标准语义一致 [30]。基于这一新语义，我们证明 PSO 满足除一条之外的所有公理 [15]。缺失的公理涉及某些内存模型提供消息传递（MP）功能的能力，即当线程 t_1 读取了线程 t_1写入的值时，将线程 t_1 的视图传递给另一线程 t_2 。这一性质在 PSO 中并不成立。因此，在 PSO 中，只有使用 MP 以外公理的正确性证明才有效。除 [15], 中已有的证明外，我们还为一个带栅栏的消息传递示例提供了新的正确性证明。对于不支持消息传递功能的弱内存模型，要保证正确性就必须使用栅栏，也就是说，证明中不能使用 MP 公理。基于此，我们举例说明了公理化证明技术。

[1] 请注意，与顺序一致执行不同，在同一状态下，线程可能会看到共享变量的不同值。

$$\textsc{Exp}\frac{v = [\![e]\!]_{ls}}{(r := e, ls) \xrightarrow{\tau} (skip, ls[r := v])} \qquad \textsc{Read}\frac{a = rd(x, r, v)}{(r := x, ls) \xrightarrow{a} (skip, ls[r := v])}$$

$$\textsc{Write}\frac{a = wr(x, v)\ \ v = [\![e]\!]_{ls}}{(x := e, ls) \xrightarrow{a} (skip, ls)} \qquad \textsc{Fence}\frac{a = fence}{(fnc, ls) \xrightarrow{a} (skip, ls)}$$

$$\textsc{ParCom}\frac{(C_1, ls) \xrightarrow{a} (C_1', ls')}{(C_1;\ C_2, ls) \xrightarrow{a} (C_1';\ C_2, ls')} \qquad \textsc{Skip}\frac{}{(skip;\ C_2, ls) \xrightarrow{\tau} (C_2, ls)}$$

$$\textsc{If1}\frac{[\![b]\!]_{ls}}{(\text{if } b \text{ then } C_1 \text{ else } C_2, ls) \xrightarrow{\tau} (C_1, ls)} \qquad \textsc{If2}\frac{\neg[\![b]\!]_{ls}}{(\text{if } b \text{ then } C_1 \text{ else } C_2, ls) \xrightarrow{\tau} (C_2, ls)}$$

$$\textsc{While1}\frac{[\![b]\!]_{ls}}{(\text{while } b \text{ do } C, ls) \xrightarrow{\tau} (C;\ \text{while } b \text{ do } C, ls)}$$

$$\textsc{While2}\frac{\neg[\![b]\!]_{ls}}{(\text{while } b \text{ do } C, ls) \xrightarrow{\tau} (skip, ls)}$$

图 1. 局部程序语义

2 背景

在考察 PSO 及其基于视图的语义之前，我们先定义程序语法和语义，部分遵循 [15] 中的记法。

2.1 程序语法与语义

我们将并发程序定义为顺序程序的并行组合。每个线程 t ∈ Tid 执行一个顺序程序 Com，并通过函数Π : Tid → Com 对线程 Tid 上的并发程序进行建模。我们令 $\mathsf{Var_G}$为全局变量集合，$\mathsf{Var_L}$ 为局部变量（或寄存器）集合，其中 $\mathsf{Var_G} \cap \mathsf{Var_L} = \varnothing$，且 $\mathsf{Var} = \mathsf{Var_G} \cup \mathsf{Var_L}$。我们假设初始时所有变量的值均为 0。

对于 x ∈ $\mathsf{Var_G}$、r ∈$\mathsf{Var_L}$ 和值 v ∈Val，我们定义以下语法

E ::= v := e | skip | fnc | r := E | r := x | x := ECom ::= com | Com; Com | if b then Com else Com | while b do Com

其中 e ∈Exp 和 b ∈BExp 是关于局部变量的表达式，e 为算术表达式，b 为布尔表达式。

与近期一些关于弱内存模型语义的方法类似 [15,16, 26], ，我们通过先定义独立于内存模型的程序语义，再将其与内存模型语义结合，来定义并发程序在弱内存模型上运行时的语义。首先，程序语义假定变量可以读到任意值

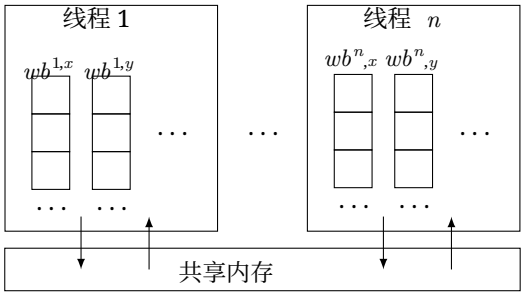


图 2. PSO 架构

，随后内存模型会限制这些值。下面我们将介绍 PSO 内存模型的操作规则（图 3）。

首先，我们从动作集开始

$$\mathsf{Act} = \{rd(x, r, v), wr(x, v), fence \mid x \in \mathsf{Var_G} \wedge r \in \mathsf{Var_L} \wedge v \in \mathsf{Val}\}$$

并添加读操作 $\mathrm{r} := \mathrm{v}$ 和一个静默操作 τ，使得 $\mathsf{Act_{ext}} = \mathsf{Act} \cup \{\mathrm{r} := \mathrm{v}\ \tau \mid \mathrm{r} \in \mathsf{Var}_L \wedge \mathrm{v} \in \mathsf{Val}\}$。对于动作 $\mathrm{a} \in \mathsf{Act}$，我们需要函数 var、$rdval$和 $wrval$，其中 $var(\mathrm{a}) \in \mathsf{Var_G}$ 描述该动作对应的全局变量。如果 a 是读操作，则 $rdval(\mathrm{a}) \in \mathsf{Val}$ 描述其值；否则，我们令 $rdval(\mathrm{a}) = \perp \notin \mathsf{Val}$。写操作的定义方式相同。借助这些函数，我们可以定义 Act 的子集：$\mathsf{Rd} = \{\mathrm{a} \in \mathsf{Act} \mid wrval(\mathrm{a}) = \perp \wedge rdval(\mathrm{a}) \neq \perp\}$，以及 $\mathsf{Wr} = \{\mathrm{a} \in \mathsf{Act} \mid rdval(\mathrm{a}) = \perp \wedge wrval(\mathrm{a}) \neq \perp\}$。对于值 v，我们假设 $\mathsf{Rd}[\mathrm{v}] = \{\mathrm{a} \in \mathsf{Rd} \mid rdval(\mathrm{a}) = \mathrm{v}\}$，以及 $\mathsf{Wr}[\mathrm{v}] = \{\mathrm{a} \in \mathsf{Wr} \mid wrval(\mathrm{a}) = \mathrm{v}\}$。我们令 $\mathsf{Act}|_x$ 为所有满足 $var(\mathrm{a}) = \mathrm{x}$ 的动作 a 的集合。因此，$\mathsf{Rd}|_x$ 是 $\mathsf{Act}|_x$ 中的所有读操作集合，而 $\mathsf{Wr}|_x$ 则是所有写操作集合。

顺序程序的操作语义由图 1 中的迁移规则给出。其中，$\mathrm{ls} : \mathsf{Var_L} \to \mathsf{Val}$ 定义了局部状态，它将每个局部变量映射到其值；$[\![\mathrm{e}]\!]_{ls}$ 表示表达式 e 在局部状态 ls 中的值。用 $\mathrm{ls}[\mathrm{r} := \mathrm{v}]$ 表示这样一个局部状态：对于除 r 以外的所有局部变量，它都与 ls 相同，而 r 在 $\mathrm{ls}[\mathrm{r} := \mathrm{v}]$ 中的值为 v。特别要注意，READ 规则允许 v 取任意值。为了将一个线程的顺序程序提升为并发程序（$\Pi : \mathsf{Tid} \to \mathsf{Com}$），我们还采用以下规则

$$\textsc{Par}\frac{(\Pi(t), lst(t)) \xrightarrow{a} (C, ls)\ \ a \in \mathsf{Act_{ext}}}{(\Pi, lst) \xrightarrow{a}_t (\Pi[t := C], lst[t := ls])}$$

其中，$\mathrm{lst} : \mathsf{Tid} \to (\mathsf{Var_L} \to \mathsf{Val})$ 将每个线程映射到其局部状态。

2.2 PSO 语义

接下来，我们给出 PSO 的操作语义。该语义采用体系结构风格，因为它直接对体系结构特定细节进行建模。PSO 内存模型包含共享内存，以及每个线程和每个变量各自的写缓冲区

$$\text{PSO-Read}\ \frac{a = rd(x, r, v) \quad v = val_\sigma(t, x)}{\sigma \overset{a,t}{\rightsquigarrow}_{PSO} \sigma}$$

$$\text{PSO-Write}\ \frac{a = wr(x, v) \quad wb' = \sigma.wb[(t, x) := \sigma.wb^{t,x} \cdot \langle v \rangle]}{\sigma \overset{a,t}{\rightsquigarrow}_{PSO} (s, wb')}$$

$$\text{PSO-Fence}\ \frac{a = fence \quad \sigma.wb^{t,x} = \langle\ \rangle \text{ for all } x \in \mathsf{Var_G}}{\sigma \overset{a,t}{\rightsquigarrow}_{PSO} \sigma}$$

$$\text{PSO-Flush}\ \frac{wb^{t,x} = \langle v \rangle \cdot w \quad s' = \sigma.s[x := v] \quad wb' = \sigma.wb[(t, x) := w]}{\sigma \overset{flush,t}{\rightsquigarrow}_{PSO} (s', wb')}$$

图 3. PSO 语义

（示意图见图 2）。每变量一个写缓冲区，这一区别使 PSO 有别于 TSO 内存模型 [1]，后者每个线程只有一个写缓冲区。线程 t 的全局变量 x 对应的写缓冲区 $wb^{t,x}$ 是一个按先进先出顺序排列、以值为条目的列表。每个写缓冲区都可以随时将其首个条目刷新到共享内存中。如果线程 t 想要读取变量 x 的值，它可以读取 $wb^{t,x}$ 中的最后一个条目；或者，如果 $wb^{t,x}$ 为空，则从共享内存中读取。接下来，我们将正式定义描述这一行为的 PSO 语义。

在 PSO 语义中，每个状态 $\sigma = (s, wb) \in \Sigma_{PSO}$ 都包含一个共享内存 $s : \mathsf{Var_G} \to \mathrm{Val}$ 和一个写缓冲区映射 $wb : (\mathrm{Tid} \times \mathsf{Var_G}) \to \mathrm{Val}^*$，它将每个线程 $t \in \mathrm{Tid}$ 和每个全局变量 $x \in \mathsf{Var_G}$ 映射到一个写缓冲区列表 $wb^{t,x}$。我们用 $I_{PSO} \subseteq \Sigma_{PSO}$ 表示初始状态，并假定其中对所有 x 都有 $s(x) = 0$，且写缓冲区为空。图 3 中的规则 PSO-Write表明，每当程序在线程 t 中向全局变量 x 写入某个值 $v \in \mathrm{Val}$ 时，我们就将 v 添加到写缓冲区 $wb^{t,x}$中。如果我们刷新写缓冲区 $wb^{t,x}$（规则 PSO-Flush），就会将共享内存 s 中 x 的条目改为 $wb^{t,x}$ 的首个条目，并将该条目从写缓冲区中删除。当程序想要在线程 t 中将变量 x 的值 v 读入寄存器 r 时（规则 PSO-Read），如果写缓冲区不为空，就读取 $wb^{t,x}$ 中的最后一个值。否则，程序直接从共享内存中读取 x 的条目。在规则中，这由函数 val_σ描述。如果写缓冲区 $wb^{t,x}$为空，则 $val_\sigma(t, x) = \sigma.s(x)$；否则，$val_\sigma(t, x) = \mathrm{last}(\sigma.wb^{t,x})$。要在线程 t 中通过一条栅栏语句（即应用规则 PSO-Fence），我们首先需要刷新该线程的所有写缓冲区，直到对每个 x 都有 $wb^{t,x}$ 为空。

为了将 PSO 的操作语义整合到程序语义中，我们需要定义由在 PSO 上运行的程序生成的迁移系统 TS_{PSO}。形式化地，我们将其写作 $TS_{PSO} = (\mathrm{Act}, \Sigma_{PSO}, I_{PSO}, T_{PSO})$，其中状态集合为 $\Sigma_{PSO} = (\mathsf{Var_G} \to \mathrm{Val}) \times ((\mathrm{Tid} \times \mathsf{Var_G}) \to \mathrm{Val}^*)$，初始状态集合为 $I_{PSO} = \{(s, wb) \in \Sigma_{PSO} \mid \forall t \in \mathrm{Tid}\ \forall x \in \mathsf{Var_G} : wb^{t,x} = \langle\ \rangle \wedge s(x) = 0\}$，迁移集合为 $T_{PSO} \in \mathrm{Tid} \times \mathrm{Act} \to 2^{\Sigma_{PSO} \times \Sigma_{PSO}}$。对于线程 $t \in \mathrm{Tid}$ 中的动作 $a \in \mathrm{Act}$，我们定义 $T_{PSO}(t, a) = FL_{PSO} \mathbin{⨾} \overset{a,t}{\rightsquigarrow}_{PSO}$，其中 $FL_{PSO} = \left(\bigcup_{t \in \mathrm{Tid}} \overset{flush,t}{\rightsquigarrow}_{PSO}\right)^*$，而 ⨾ 表示关系复合。

初始： $x := 0;\quad y := 0;$ 线程 1 线程 2 1: $r_1 := y;$ 2: $= x;$ 3: $x := 1;$ 4: $y := 1;$ $\{r_1 \in \{0,1\} \wedge r_2 \in \{0,1\}\}$

图 4. 无栅栏的消息传递

初始化： $x := 0;\quad y := 0;$ 线程 1 1： $x := 1;$ 2： $fnc;$ 3： $y := 1;$ 线程 2 4： $r_1 := y;$ 5： $r_2 := x;$ $\{r_1 = 1 \Rightarrow r_2 = 1\}$

图 5. 带栅栏的消息传递

2.3 组合语义

程序与弱内存语义通过以下三条提升规则进行组合。

$$\textsc{Silent}\frac{(\Pi, lst) \xrightarrow{\tau}_t (\Pi', lst')}{(\Pi, lst, \sigma) \Rightarrow_t (\Pi', lst', \sigma)} \qquad \textsc{Local}\frac{(\Pi, lst) \xrightarrow{r:=v}_t (\Pi', lst')}{(\Pi, lst, \sigma) \xRightarrow{r:=v}_t (\Pi', lst', \sigma)}$$

$$\textsc{Memory}\frac{(\Pi, lst) \xrightarrow{a}_t (\Pi', lst') \quad a \in \mathsf{Act} \quad (\sigma, \sigma') \in T(t, a)}{(\Pi, lst, \sigma) \xRightarrow{a}_t (\Pi', lst', \sigma')}$$

Memory规则将局部程序语义中针对 $\in$ 动作集的规则（Read、 Write、 Fence 和 ParCom）与 PSO 语义中的规则结合起来。它用于影响全局内存的动作。这里，lst 只有在我们将全局变量的值读入寄存器时才会改变。我们使用 Local规则将值直接读入寄存器（Read），这会改变 lst；但由于不涉及全局变量， σ 保持不变。对于局部语义中的其余规则，我们使用 Silent规则，只有在将表达式读入寄存器时（Exp），它才会改变 lst。

我们通过图 4 中所谓的消息传递测试 [6] 来说明 PSO 的语义。它被称为“消息传递”，因为预期行为如下：当线程 2 读到 y 的值为 1 时，随后也会读到 x 的值为 1。因此，线程 1 将 x 的值为 1 这一消息，在读到 y 的值为 1 时传递给线程 2。然而，PSO 也允许出现结果 $\mathrm{r}_1 = 1 \wedge \mathrm{r}_2 = 0$（没有消息传递）。这可以解释如下。首先，线程 1 将值 1 写入 x，因此 $\mathrm{wb}^{1,\ x} = \langle 1 \rangle$。之后，y 也会发生同样的情况，因此 $\mathrm{wb}^{1,\ y} = \langle 1 \rangle$。接着，在读操作之前只刷新一个写缓冲区，即 $\mathrm{wb}^{1,\ y}$。在这种情况下，线程 1 从共享内存中读到 y 的值为 1，x 的值为 0。另一方面，在带栅栏的消息传递示例（图 5）中，不允许出现结果 $\mathrm{r}_1 = 1 \wedge \mathrm{r}_2 = 0$。在那里，只有在栅栏操作之后才能写入 y 的值；此时线程 1 的所有写缓冲区都必须刷新。因此，线程 2 只有在也读到 x 的值为 1 时，才能读到 y 的值为 1。稍后我们将使用带栅栏的消息传递示例来说明我们的推理技术如何运作。不过，我们先在下一节介绍公理化推理。

3 公理化推理

[15] 的公理针对任意迁移系统 TS $\widehat{=}$(Act Σ, I, T) 进行推理，例如 PSO 的迁移系统，其中 Act 是动作集， Σ 是状态集合，I $\subseteq \Sigma$

$$\Sigma = \text{最弱前置条件}(R, \Sigma) \quad \text{(不终止)}$$

$$R' \subseteq R \wedge P \subseteq P' \Rightarrow \text{最弱前置条件}(R, P) \subseteq \text{最弱前置条件}(R', P') \quad \text{((反)单调性)}$$

$$\text{最弱前置条件}(R, \text{最弱前置条件}(R', P)) = \text{最弱前置条件}(R \fatsemi R', P) \quad \text{(复合)}$$

$$R[\text{最弱前置条件}(R, P)] \subseteq P \quad \text{(关系应用)}$$

$$\text{最弱前置条件}(R, P) \cap \text{最弱前置条件}(R, Q) = \text{最弱前置条件}(R, P \cap Q) \quad \text{(合取性)}$$

$$\text{最弱前置条件}(R, P) \cup \text{最弱前置条件}(R, Q) \subseteq \text{最弱前置条件}(R, P \cup Q) \quad \text{(析取性)}$$

图6。最弱前置条件的性质

是初始状态集合，而 T $\in$ Tid $\times$ Act $\rightarrow$ $2^{\Sigma\times\Sigma}$ 是迁移集合。公理化采用最弱自由前置条件变换器 [14], 作为性质规约与验证的基础。对于关系 R 和状态集合 P（表示一个谓词），令 wlp： $2^{\Sigma\times\Sigma} \times 2^{\Sigma} \rightarrow 2^{\Sigma}$ 为

$$\mathsf{wlp}(R, P) \mathrel{\widehat{=}} \{\sigma \in \Sigma \mid \forall \sigma' : (\sigma, \sigma') \in R \Rightarrow \sigma' \in P\}$$

图 6 给出了最弱前置条件的一些标准性质，其中 $\circ_9$ 再次表示关系复合，而 R[S] 表示关系像。

在本文中，R 通常实例化为关系 T (t, a)，其中 T 是迁移关系，t 是线程，a 是动作。我们称 R 在状态 σ 下被禁用，当且仅当 $\sigma \in$ dis(R) 成立，其中 dis(R) $\mathrel{\widehat{=}}$ wlp(R $\varnothing$)。

接下来，我们给出 [15] 的公理。这些公理按层级结构组织，每一层都会在较低层级的公理基础上添加一组公理。图 7 展示了这一层级。在实例化内存模型时，必须具体化参数 $vmax$、interf（第 Core 层）和 $sync$ （第 MsgPassing 层）。它们的含义将在下文说明。

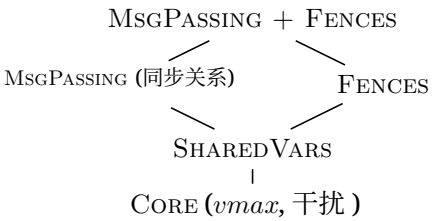


图7。公理层次结构

第 Core 层。最低层包含描述基于视图的内存模型语义核心属性的基本公理。这里的关键概念是视图极大性：如果一个线程在某个作用于变量 x $\in$ $\mathsf{Var_G}$ 的动作上具有视图极大性，那么它在该变量上拥有“最新”的视图。在 PSO 中，例如，对于线程 t 对 x 的读操作，如果 x 位于共享内存中且不在任何写缓冲区中，或者 t 的 x 写缓冲区中有一项会比所有其他 x 的值“更晚”刷新到共享内存，那么线程 t 在读取 x 时具有视图极大性。这意味着，虽然不具有视图极大性的线程可能能够读取 x 的较旧值，但线程 t 读取的是最新的值。然而，由于 PSO 的语义中并未体现“更晚刷新”这一概念，我们稍后需要定义一种替代语义。这样便能为 PSO 实例化 $vmax$ 。

公理系统的核心要素是保视图模拟。

C1 : $\forall t \in$ Tid $a \in$ 动作集 : $I \subseteq vmax(t, a)$
C2 : $\forall \sigma, \sigma' \in \Sigma, t, t' \in$ Tid $a \in$ 动作集 :$\sigma \in vmax(t, a) \wedge (\sigma, \sigma') \in T(t, a) \Rightarrow \exists \tau \in \Sigma$:
$(\sigma', \tau) \in \beta \wedge (\sigma, \tau) \in T(t', a)$ ⨟ βC3 : $\forall t \in$ Tid $a \in$ 动作集 : $T(t, a) \subseteq \beta$ ⨟ $interf(t, a)$
⨟ βC4 : $\forall t \in$ Tid $a, b \in$ 动作集 : $vmax(t, a) \subseteq$ 最弱前置条件$(interf(t, b)\ vmax(t, a))$

图 8. 第 CORE 级的公理

定义 1. 对于迁移系统 $TS =$(动作集 Σ, I , T), 保持视图的模拟记为 β, 是满足以下条件的最弱关系 R, 其中所有线程 t $\in$Tid 以及所有动作 a $\in$ 动作集

$$R \fatsemi T(t, a) \subseteq T(t, a) \fatsemi R \qquad \text{(semi-commutation)}$$
$$vmax(t, a) \subseteq \mathsf{wlp}(R, vmax(t, a)) \qquad \text{(view maximality)}$$

保视图模拟在概念上涵盖所有不执行读、写等可见动作而改变状态的各类系统步。在 PSO 中，这些系统步就是刷新操作。为了避免这些步骤破坏证明，与这些步骤相关联的关系应保持线程的视图极大性，并且应与迁移关系半交换。

此外，核心公理使用干扰关系 interf $\in$Tid $\times$ Act $\to 2^{\Sigma \times \Sigma}$，它与 β 一起对迁移关系 T (t, a) 提供过近似，以便抽象掉内存模型的细节，并恢复推理的标准性质（例如，对不同变量的写和读可交换）。图 8 给出了所有核心公理。

在推理中，我们之后会用到以下基于视图的断言。

定义 2. 对于线程 t、变量 x $\in \mathsf{Var_G}$ 和值 v $\in$ Val，我们定义

$$[x \not\approx v]_t \mathrel{\widehat{=}} \mathsf{dis}(T(t, Rd_{|x}[v])) \quad \textit{(Impossible value)}$$
$$[x \equiv v]_t \mathrel{\widehat{=}} \bigcap_{u \neq v} [x \not\approx u]_t \quad \textit{(Definite value)}$$
$$x_{\uparrow t} \mathrel{\widehat{=}} \bigcap_{a \in \mathsf{Act}_{|x}} vmax(t, a) \quad \textit{(Maximal view)}$$
$$[x = v]_t \mathrel{\widehat{=}} [x \equiv v]_t \cap x_{\uparrow t} \quad \textit{(Synced value)}$$

断言应当（事实上也确实）在 β 下保持稳定。

定义 3. 谓词 P $\in 2^{\Sigma}$ 是 β-稳定的，当且仅当 P $\subseteq$ wlp(β, P)。

事实上，上述引入的四个断言都是 β-稳定的。此外，以下性质可由公理推导得出。

引理1（[15]）。对于任意 a $\in$ 动作集、线程 t 和 β-稳定谓词 P，若 P $\subseteq$最弱前置条件(interf (t, a), P)，则 P $\subseteq$最弱前置条件(T (t , a), P)。

SV1 : $\forall a, b \in$ 动作集 $t, t' \in$ Tid s.t. $var(a) \neq var(b)$:$interf(t', b) \fatsemi T(t, a) \subseteq T(t, a) \fatsemi interf(t', b)$ SV2 : $\forall a, b \in$ 动作集 $t, t' \in$ Tid s.t. $var(a) \neq var(b)$:$vmax(t, a) \subseteq$最弱前置条件($interf(t', b)$ $vmax(t, a)$) RW1 : $\forall t, t' \in$ Tid $x \in$变量$_G$, $a_r \in Rd_{|x}$, $a_w \in Wr_{|x}$ s.t. $rdval(a_r) \neq wrval(a_w)$:$interf(t', a_w) \fatsemi T(t, a_r) \subseteq T(t, a_r) \fatsemi interf(t', a_w)$ RW2 : $\forall a \in$动作集 $t, t' \in$ Tid $a_r \in Rd_{|var(a)}$:$interf(t', a_r) \fatsemi T(t, a) \subseteq T(t, a) \fatsemi interf(t', a_r)$ RW3 : $\forall a \in$动作集 $t, t' \in$ Tid $a_r \in Rd_{|var(a)}$:$vmax(t, a) \subseteq$最弱前置条件($interf(t', a_r)$ $vmax(t, a)$) RW4 : $\forall x \in$变量$_G$, $t \in$ Tid : $\Sigma \subseteq$定义域($T(t, Rd_{|x})$) RW5 : $\forall x \in$变量$_G$, $a_w \in Wr_{|x}$, $v = wrval(a_w)$: $\Sigma \subseteq$最弱前置条件($T(t, a_w)$, 定义域($T(t, Rd_{|x}[v]$))) RW6 : $\forall x \in$变量$_G$, $t \in$ Tid $v \in$值 : $x_{\uparrow t} \subseteq \bigcup [x \equiv v]_{tv \in}$值 RW7 : $\forall x \in$变量$_G$, a_w, a_r, $a \in$动作集$_{|x}$, $t, t' \in$ Tid, s.t. $wrval(a_w) = rdval(a_r) \wedge t \neq t'$:$vmax(t, a_w) \cap$dis($T(t', a_r)$) $\subseteq$最弱前置条件($T(t, a_w)$, 最弱前置条件($T(t', a_r)$ $vmax(t', a)$))

图 9. 第 SharedVars 级的公理

第 SharedVars 级。公理层次结构的下一级讨论了关于动作及其访问的共享变量的公理。第 SharedVariables 级包含第 Core 级的所有公理，以及图 9 中给出的公理。我们以 SV1 和 RW6 为例，介绍其中的两个公理。公理 SV1 描述了作用于不同变量的动作（即读操作和写操作）的半交换性质。编程语言通常都期望具备这一性质。例如，如果我们先向 x 写入 4，再向 y 写入 5，且 y $\neq$ x，那么先向 y 写入 5，再向 x 写入 4，也应当到达相同的状态。然而，弱内存模型未必满足这一性质。例如，在 TSO [32] 中，如果所有位置共用一个写缓冲区，我们将到达两个不同的状态。为了在公理化推理中恢复这一性质，该公理使用 interf (t′, b)（而非 T (t′, b)）。

公理 RW6 使用线程 t 在变量 x $\in$ Var$_G$ 上的最大视图这一概念。RW6 指出，对于 x 而言处于最大视图的线程必须知道 x 的一个确定值。

当公理 C3、SV1 和 SV2 成立时，对于β-稳定谓词，我们可以得到以下性质：

引理 2。
对于所有 P $\subseteq \Sigma$ 和线程 t，若 P $\subseteq$wlp(β, P)，则 P $\subseteq$wlp(T (t, 栅栏), P)。

如果 RW2 和 RW3 也成立，我们可以证明读操作也具有相同的性质。

引理 3（[15]）。对于所有 P $\subseteq \Sigma$、线程 t 和 a$_r \in Rd$，如果 P $\subseteq$最弱前置条件(β, P)，那么 P $\subseteq$最弱前置条件(T (t, a$_r$), P)。

层级 MsgPassing 和 Fences。最后，我们还有两个层级，每个层级只增加一条公理（以及一个 MsgPassing + Fences 将这两个层级合并的层级）。层级MsgPassing 包含 SharedVars 的所有公理，外加以下消息传递公理。

MP 对于 a_w、a_r、b ∈Act 以及 t、t′ ∈Tid，若满足 (a_w, a_r) ∈ $sync$, var(a_w) =var(a_r)，$wrval$(a_w) = $rdval$(a_r)，var(b) ≠ var(a_w)，且 t ≠t′，则有

$$vmax(t, b) \cap \mathsf{wlp}(T(t', a_r), vmax(t', b)) \subseteq \mathsf{wlp}(T(t, a_w), \mathsf{wlp}(T(t', a_r), vmax(t', b))).$$

直观而言，MP 陈述了以下性质：考虑两个动作 a_w 和 a_r，第一个 a 是写操作，第二个 a 是读操作。读操作读取由 a_w 写入的值。此外，(a_w, a_r) ∈ $sync$ 表示读操作与写操作之间存在同步关系。关系 $sync$ 是公理化中的一个参数；与 interf 和 $vmax$ 一样，它需要针对具体的内存模型进行实例化。例如：在 TSO 中，任何写操作都与任何读操作存在 $sync$ 关系；在 C11 RAR [16] 中，只有标记为 Releasing 的写操作才与标记为 Acquiring 的读操作同步。

该公理要求如下：如果线程 t 在动作 b 上是视图极大的（例如，变量 x 上），并且读操作 a_r 会使线程 t′ 在 b 上也成为视图极大，那么执行写操作 a_w 和读操作 a_r 后，线程 t′ 实际上会在 b[2] 上成为视图极大。

请注意，该公理不适用于 PSO（或者说，我们可以通过将 $sync$ 设为 ∅ 来使其成立，但这样一来就无法在任何地方应用它）。原因在于每个变量都有独立的写缓冲区，这一点已在图 4 的示例中解释过。

级别 Fences 的最后一条公理涉及栅栏指令。它指出，当线程 t 在变量 a 上的视图达到最大时，执行栅栏指令会使其他所有线程在变量 a 上的视图也达到最大。

FNC ∀a ∈Act, t, t′ ∈Tid: $vmax$(t , a) ⊆最弱前置条件(T (t , fence) $vmax$(t ′, a)。

最后，要对局部寄存器 r ∈Var 的赋值进行推理，需要一条标准的最弱前置条件规则：

$$e[r := v] \subseteq \mathsf{wlp}(T(t, rd(x, r, v)), e) \tag{1}$$

这里，$expr$[r := v] 表示在表达式 e 中用 v 替换 r。至此，公理集已完整。接下来，我们再次考察 PSO，将参数$vmax$ 和 interf 实例化为 PSO，然后证明 PSO 满足级别SharedVars 的所有公理以及公理 FNC。

4 预言式PSO

要使用我们的公理化方法，我们需要为具体的内存模型实例化参数最大状态（vmax）和干扰（interf）。然而，对于

[2] 这类似于 Lahav 和 Boker 的共享内存因果性原则 [25]。

$$\text{PP-READ}\ \frac{a = rd(x, r, v) \quad v = val_\sigma(t, x)}{\sigma \overset{a,t}{\rightsquigarrow}_{PP} \sigma}$$

$$\text{PP-WRITE}\ \frac{a = wr(x, v) \quad fresh_\sigma(t, x, q) \quad wb' = \sigma.wb[(t, x) := \sigma.wb^{t,x} \cdot \langle (v, q) \rangle]}{\sigma \overset{a,t}{\rightsquigarrow}_{PP} (s, wb')}$$

$$\text{PP-FENCE}\ \frac{a = fence \quad \sigma.wb^{t,x} = \langle\ \rangle \text{ for all } x \in \mathsf{Var_G}}{\sigma \overset{a,t}{\rightsquigarrow}_{PP} \sigma}$$

$$\text{PP-FLUSH}\ \frac{wb^{t,x} = \langle (v, \cdot) \rangle \cdot w \quad s' = \sigma.s[x := v] \quad nextFlush_\sigma(t, x) \quad wb' = \sigma.wb[(t, x) := w]}{\sigma \overset{flush,t}{\rightsquigarrow}_{PP} (s', wb')}$$

图 10. PPSO 语义

PSO 而言，这并不简单，因为我们无法从状态中看出视图极大性；我们不知道写缓冲区中的条目最后是为哪个线程刷新的，因此也不知道哪个线程对某个变量拥有“最新”的视图。为了仍能使用这些公理对 PSO 进行推理，我们首先为 PSO 提供一种替代语义，称为预言式 PSO（PPSO）。它之所以“预言式”，是因为它预先知道写缓冲区条目的刷新顺序。为此，我们在写入时直接为每个条目分配一个时间戳，并且只按时间戳的顺序刷新条目。随后证明，PPSO 与 PSO 轨迹等价。

PPSO 的语义见图 10。PSO 和 PPSO 的区别仅在于写缓冲区的类型。在预言式版本中，写缓冲区不仅包含写入的值，还为每次写操作包含一个时间戳 q $\in \mathbb{Q}$。因此，只有当当前状态 σ中条件 fresh$_\sigma$(t, x , q) =为真时，我们才能在线程 t 中将值 v 写入变量 x。形式上，fresh$_\sigma$(t, x , q) 定义为

$$(\forall\, t' \in \mathsf{Tid}\ \forall\, x' \in \mathsf{Var_G} : (\cdot, q) \notin \sigma.wb^{t',x'}) \wedge (\forall (\cdot, q') \in \sigma.wb^{x,t} : q > q')$$

这意味着没有任何写缓冲区包含时间戳 q，并且 wb$^{t,\ x}$中的所有时间戳都小于 q。在这种情况下，我们可以将对 (v , q) 添加到 wb$^{t,\ x}$。要刷新写缓冲区 wb$^{t,\ x}$的第一个条目 (v , q)，条件 nextFlush$_\sigma$(t, x) 必须为真。也就是说，满足以下条件时

$$(\exists\, q : (\cdot, q) = wb^{t,x}(0)) \wedge (\forall\, t' \forall\, x' \forall (\cdot, q') \in wb^{t',x'} : t' \neq t \vee x' \neq x \Rightarrow q' > q)$$

然后，q 小于其他所有写缓冲区中的每个时间戳。接着，我们可以删除该条目，并像之前一样更改内存。读和栅栏规则与 PSO 语义中的规则相同。唯一的区别是，如果 wb$^{t,\ x}$ 非空，那么 val$_\sigma$(t , x) 现在是 σ.wb$^{t,\ x}$中最后一个条目的值。在这种情况下，我们令 val$_\sigma$(t, x) =val (last (σ.wb$^{t,\ x}$))，否则 val$_\sigma$(t , x) 仍为 σ.s(x)。

我们定义迁移系统 T$_{PP}$，它由运行在 PPSO 上的程序生成，与 TS$_{PSO}$ 类似。因此，我们设 TS$_{PP}$ =(Act Σ_{PP}, I$_{PP}$, T$_{PP}$)，其状态集合为 Σ_{PP} = (Var$_\mathsf{G}$ $\to$ Val) $\times$((Tid $\times$ Var$_\mathsf{G}$) $\to$ (Val $\times\, \mathbb{Q})^*$)，初始状态集合为 I$_{PP}$ = {(s, wb) | s $\in$ Var$_\mathsf{G}$ $\to$ Val $\wedge\, \forall$ t $\in$ Tid $\forall$x $\in$ Var$_\mathsf{G}$: wb^{t,x} =$\langle\ \rangle$}，迁移集合为 T$_{PP}$ $\in$ Tid $\times$ Act $\to 2^{\Sigma_{PP} \times \Sigma_{PP}}$。与 T$_{PSO}$ 类似，我们设 T$_{PP}$(t, a) =FL$_{PP}$ 9 $^{a}\!\rightsquigarrow^{t}_{PP}$，其中 FL$_{PP}$ = $\left(\bigcup_{t \in}$Tid $^{flush}\!\rightsquigarrow\ ^{t}{}_{PP}\right)^*$。

我们称这种语义为预言式 PSO，因为有了时间戳，我们现在可以预测写缓冲区刷新操作的顺序。因此，我们知道状态 $\sigma \in \Sigma_{PP}$ 是否对线程 t 中的动作 a 具有最大视图。由此，我们可以为 PPSO 定义 $vmax$ 和干扰，并研究第 3 节中的哪些公理成立。不过，我们首先证明 PSO 与 PPSO（更准确地说，TS_{PSO} 与 TS_{PP}）是轨迹等价的。因此，对于在 PPSO 上运行的程序，其任何正确性证明同样适用于 PSO。

为了证明迹等价，我们首先需要一些定义。这里我们部分沿用 Lynch 和 Vaandrager [30] 的记号。设 TS_A =(Act Σ_A, I_A, T_A) 和 TS_C =(Act Σ_C, I_C, T_C) 是两个迁移系统。TS_A 的一条迹是一个动作 $\alpha \in$ Act* 的有限序列 $\exists \sigma_0 \in I_A, \sigma \in \Sigma_A$: $\sigma_0 \stackrel{\alpha}{\Rightarrow} \sigma$。这里 $\sigma_0 \stackrel{\alpha}{\Rightarrow} \sigma$ 表示 $(\sigma_0, \sigma) \in T_A(t_1, a_1)$ o9 ... o9 $T_A(t_n, a_n)$, 其中 $\alpha = a_1 ... a_n \in$Act*，且 t1, ..., $t_n \in$Tid。我们用 traces(TS_A) 表示 TS_A 的所有迹。如果两个迁移系统 TS_A 和 TS_C 满足 traces(TS_A) = traces(TS_C), 则称它们迹等价。因此，为了证明 TS_{PSO} 和 TS_{PP} 迹等价，我们证明 (i) traces(TS_{PP}) $\subseteq$traces(TS_{PSO})，以及 (ii) traces(TS_{PSO}) $\subseteq$traces(TS_{PP})。为此，我们采用正向模拟和反向模拟。我们从正向模拟以及 (i) 的证明开始。

定义 4（[30]）。从 TS_A 到 TS_C 的正向模拟是一个关系 F $\subseteq \Sigma_A \times \Sigma_C$，满足对于所有 $\sigma_0 \in I_A$ 都有 F(σ_0) $\cap I_C \neq \varnothing$，并且对于每个 a $\in$动作集

$$(\sigma_1, \sigma_2) \in F \wedge (\sigma_1, \sigma_1') \in T_A(t, a) \Rightarrow \exists \sigma_2' : (\sigma_2, \sigma_2') \in T_C(t, a) \wedge (\sigma_1', \sigma_2') \in F.$$

Lynch 和 Vaandrager [30] 告诉我们，只要存在从 TS_{PP}到 TS_{PSO}的正向模拟，就有 traces(TS_{PP}) $\subseteq$traces(TS_{PSO})。因此，我们需要以下定理:

定理 1。从 TS_{PP} 到 TS_{PSO}存在一个正向模拟。

在该定理的证明中，我们选择一个关系 F $\subseteq \Sigma_{PP} \times \Sigma_{PSO}$，并证明 F 是一个正向模拟。对于 σ_1 = (s_1,wb_1) 和 σ_2 = (s_2,wb_2)，我们称

$$(\sigma_1, \sigma_2) \in F :\Leftrightarrow s_1 = s_2 \wedge \forall\, t \in \mathsf{Tid}\ \forall\, x \in \mathsf{Var_G} : wb_2^{t,x} = wb_1^{t,x}.val$$

借助 $wb^{t,\,x}$.val，我们描述一个只包含 $wb^{t,\,x}$的值的写缓冲区。这意味着，我们的关系只包含共享内存相同且每个写缓冲区中的值都相同的成对状态。完整证明见附录。(i)直接由该定理得出。接下来，我们研究后向模拟，以证明 (ii)。

定义 5（[30]）。从 TS_A 到 TS_C 的反向模拟是一个关系 B $\subseteq \Sigma_A \times \Sigma_C$，满足对于所有 $\sigma_0 \in I_A$，B(σ_0) $\subseteq I_C$；并且对于每个 a $\in$动作集

$$(\sigma_1', \sigma_2') \in B \wedge (\sigma_1, \sigma_1') \in T_A(t, a) \Rightarrow \exists \sigma_2 : (\sigma_2, \sigma_2') \in T_C(t, a) \wedge (\sigma_1, \sigma_2) \in B$$

且 B 在 Σ_A 上是全函数。

$$vmax_{PP}(t,a) = \begin{cases} maxTS(t, var(a)), & \text{if } a \in Wr \cup Rd \\ \Sigma_{PP}, & \text{else} \end{cases}$$

$$interf_{PP}(t,a) = \begin{cases} id, & \text{if } a \in Rd \cup \{fence\} \\ T_{PP}(t,a) \fatsemi FL_{PP}, & a \in Wr \end{cases}$$

$$\begin{aligned} maxTS(t,x) =& \{\sigma \mid \sigma.wb^{t,x} \neq \langle\,\rangle \wedge (\forall\, t', q : (\cdot, q) \in \sigma.wb^{t',x} \Rightarrow ts(last(\sigma.wb^{t,x})) \geq q)\} \\ & \cup \{\sigma \mid \forall\, t' : \sigma.wb^{t',x} = \langle\,\rangle\} \end{aligned}$$

图 11. PPSO 定义

在下一个定理中，我们将证明存在从 TS_{PSO} 到 TS_{PP} 的反向模拟。由于我们只考虑有限轨迹，[30] 证明了此时 $\mathrm{traces}(\mathrm{TS}_{PSO}) \subseteq \mathrm{traces}(\mathrm{TS}_{PP})$。

定理 2. 存在从 TS_{PSO} 到 TS_{PP}的反向模拟。

我们通过选择关系 $B \subseteq \Sigma_{PSO} \times \Sigma_{PP}$ 来证明此定理，该关系类似于正向模拟，其中

$$(\sigma_1, \sigma_2) \in B :\Leftrightarrow s_1 = s_2 \wedge \forall\, t \in \mathsf{Tid}\ \forall\, x \in \mathsf{Var_G} : wb_1^{t,x} = wb_2^{t,x}.val$$

对于 $\sigma_1 = (s_1, wb_1) \in \Sigma_{PSO}$和 $\sigma_2 = (s_2, wb_2) \in \Sigma_{PP}$。为了证明 B 是反向模拟，我们使用以下不变式：

引理 4. 设 $t \in \mathsf{Tid}$，$x \in \mathsf{Var_G}$，$\sigma_0 \in I_{PP}$ 且 $\sigma \in \Sigma_{PP}$ 满足 $\sigma_0 \overset{\alpha}{\Rightarrow} \sigma$ 对于$\alpha \in$迹(TS_{PP})。若 $q = ts(last(\sigma.wb^{t,x}))$，则有

$$\begin{aligned} &\forall\, t' \in \mathsf{Tid}\ \forall\, x' \in \mathsf{Var_G}, t' \neq t \vee x' \neq x : (\cdot, q) \notin \sigma.wb^{t',x'} \\ &\quad \wedge \forall (\cdot, q') \in \sigma.wb^{x,t}, (\cdot, q') \neq last(\sigma.wb^{t,x}) : q > q' \end{aligned}$$

我们将需要该引理来确保，对于每个可达状态 $\sigma \in \Sigma_{PP}$中的每个写缓冲区 $wb^{t,x}$，其最后一个时间戳 q 都满足 $\mathrm{fresh}_\sigma(t, x, q) =$为 true。因此，在 PPSO 下，我们可以随时为每个线程中的每个变量执行写操作。两项证明见附录。

推论 1. TS_{PSO} 和 TS_{PP} 轨迹等价。

利用这种等价关系，我们可以通过展示 PPSO 满足哪些公理，说明 PSO 在公理层次结构中处于哪个层级。为此，我们首先需要实例化第 3 节中提到的参数。图 11 概述了这些定义。例如，具有最大视图 $vmax_{PP}(t, a)$ 的状态集合包含所有满足 a 为 =栅栏的状态。对于动作 $a \in$写操作 $\cup$读操作，$vmax_{PP}(t, a)$ 包含所有对于线程 t 和变量 x =var (a) 具有最大时间戳的状态。这意味着，要么 x 的所有写缓冲区都为空，要么 $\sigma.wb^{t,x}$ 中的最后一个时间戳大于 x 的写缓冲区中的任何其他时间戳。还要注意，FL_{PP} 满足定义 1 的性质，因此 FL_{PP} 至少是保持视图的模拟 β的一个子集。

定理 3。PSO 满足 Core、 SharedVars 和 Fences 的公理，但不满足 MsgPassing 的公理。

关于满足这些公理的证明，请参见附录，其中我们证明 PPSO 满足这些公理。根据推论 1，这些公理对 PSO 也成立。通过考察图 4，我们可以证明 MP 对 PPSO 不成立。该示例的一个可能结果是 $r_1 = 1 \wedge r_2 = 0$ （见第 2.3 节）。令 a_w =写操作 (y, 1)， a_r =读操作 (y, r_1, 1)， b =写操作 (x , 1)， t = 1和 t′ = 2。我们选择 σ 为线程 1 中执行写操作 x := 1 之后的状态，使得$\sigma.wb^{1,x} = \langle(1, 2)\rangle$。因此 $\sigma \in vmax_{PP}$(1, b)，并且由于 $\sigma \notin$ dom(T_{PP}(2, a_r))，我们还得到 $\sigma \in$ wlp(T_{PP}(2, a_r) $vmax_{PP}$(2, b))。现在需要证明，$\sigma \notin$ wlp(T_{PP}(1, a_w), wlp(T_{PP}(2, a_r) $vmax$(2, b)))。为此，令 σ'为 y := 1之后的一个状态，使得 $\sigma'.wb^{1,x} = \langle(1, 2)\rangle$ 且 $\sigma'.wb^{1,y} = \langle(1, 1)\rangle$。刷新 $wb^{1,y}$ 后，我们得到状态 σ'' ，此时 $\sigma''.s(y) = 1$ 且 $\sigma''.wb^{1,x} = \langle(1, 2)\rangle$仍然成立。于是 $(\sigma, \sigma') \in T_{PP}$(1, a_w) 且 $(\sigma', \sigma'') \in T_{PP}$ =(2, a_r)，但 $\sigma'' \notin vmax_{PP}$(2, b)。因此， $\sigma \notin$wlp(T_{PP}(1, a_w), wlp(T_{PP}(2, a_r) $vmax_{PP}$ (2, b)))，由此可知 PPSO 和 PSO 均不满足 MP。

5 示例证明

本节将应用这种公理化推理形式，证明带栅栏的消息传递测试的正确性（见图 5）。为此，我们对并发程序 [31] 使用 Owicki-Gries 推理。与 [15] 一样，我们通过证明以下形式的霍尔三元组 [20] {P }com_t{Q} 成立来证明

$$P \subseteq \mathsf{wlp}(T(t, a), Q)$$

对于属于 com_t 的所有动作 a 均成立。其中，com_t 是线程 t 的程序命令，我们将 P 称为前置断言，将 Q 称为后置断言。这些断言出现在程序的证明大纲中，如 Owicki-Gries 推理 [31] 中所示。命令上方的断言是其前置断言，下方的断言是其后置断言。要检查证明大纲是否有效，我们需要区分局部正确性和全局正确性（无干扰性）。当每个线程均局部正确且证明大纲本身全局正确时，该证明大纲才有效。

定义 6（[15]）。

若线程 t 中每个以前置断言 P 和后置断言 Q 标注的程序命令 com 都满足 {P }com_t{Q}，则称线程 t 在证明大纲中局部正确。

若对于每一对线程 t、t′ {，对于线程 t 的证明大纲中的每个断言 R，以及线程 t′ 中以前置断言 P 标注的程序命令 com，都满足 R ∩P }$com_{t'}${R}，则称该证明大纲全局正确。

接下来我们来看带栅栏的消息传递示例。在证明大纲中，除了定义 2 中给出的断言外，我们还使用一种类型的断言。对于 t ∈ Tid、x、y ∈Var_G 以及 u、v ∈Val，我们称之为

$$\langle y = u\rangle[x = v]_t = \mathsf{wlp}(T(t, rd(y, r, u)), [x = v]_t)$$

初始化： $x := 0;\quad y := 0;$

$\{[x = 0]_1 \cap [y = 0]_1 \cap [x = 0]_2 \cap [x = 0]_2\}$

线程 1	线程 2
$\{[x = 0]_1 \cap [x = 0]_2 \cap [y \not\approx 1]_2\}$	$\{\langle y = 1\rangle[x = 1]_2\}$
1： $x := 1;$	4 : $r_1 := y;$
$\{[x = 1]_1 \cap [y \not\approx 1]_2\}$	$\{r_1 \neq 1 \cup [x = 1]_2\}$
2： $fnc;$	5： $r_2 := x;$
$\{[x = 1]_1 \cap [x = 1]_2 \cap [y \not\approx 1]_2\}$	$\{r_1 \neq 1 \cup r_2 = 1\}$
3： $y := 1;$	
$\{true\}$	

$\{r_1 \neq 1 \cup r_2 = 1\}$

图 12. 带栅栏的消息传递的证明大纲

条件观察断言。如果同步值断言 $[\mathrm{x} = \mathrm{v}]_t$ 成立，那么对于所有 t 能先为 y 读到值 u 的状态，该断言都成立。由于该断言只包含 β-稳定谓词，因此它本身也是 β-稳定的。这种推理技巧仅使用适用于 PSO 的公理。因此，给出的证明大纲适用于所有满足这些公理的内存模型。

引理 5. 图 12 中的证明大纲在 FENCES下有效。

为证明此引理，我们需要证明该证明大纲的局部正确性和全局正确性。下面通过证明线程 2 的全局正确性对应的一个 Hoare 三元组，详细说明证明过程。为此，我们从线程 2 的证明大纲中选取断言$\langle y = 1\rangle[x = 1]_2$，以及线程 1 的命令$y := 1$ 及其前置断言 $[x = 1]_1 \cap [x = 1]_2 \cap [y \not\approx 1]_2$:

$$\{\langle y = 1\rangle[x = 1]_2 \cap [x = 1]_1 \cap [x = 1]_2 \cap [y \not\approx 1]_2\} y :=_1 1 \{\langle y = 1\rangle[x = 1]_2\}$$

这意味着我们要证明

$$\begin{aligned}&\mathsf{wlp}(T(2, rd(y, r_1, 1)), [x = 1]_2) \cap [x = 1]_1 \cap [x = 1]_2 \cap [y \not\approx 1]_2\\ &\subseteq \mathsf{wlp}(T(1, wr(y, 1)), \mathsf{wlp}(T(2, rd(y, r_1, 1)), [x = 1]_2))\end{aligned}$$

由于 $\mathsf{wlp}(\mathrm{T}\ (2, \mathrm{rd}(\mathrm{y}, \mathrm{r}_1, 1)), [\mathrm{x} = 1]_2) \cap [\mathrm{x} = 1]_1 \cap [\mathrm{x} = 1]_2 \cap [\mathrm{y} \not\approx 1]_2 \subseteq [\mathrm{x} = 1]_2$,$[\mathrm{x} = 1]_2 \subseteq \mathsf{wlp}(\mathrm{T}\ (2, \mathrm{rd}(\mathrm{y}, \mathrm{r}_1, 1)), [\mathrm{x} = 1]_2)$（见引理 3），再结合图 6 中的单调性规则，我们只需证明

$$[x = 1]_2 \subseteq \mathsf{wlp}(T(1, wr(y, 1)), [x = 1]_2)$$

我们将其分为两部分：(i) $[\mathrm{x} \equiv 1]_2 \subseteq \mathsf{wlp}(\mathrm{T}\ (1, \mathrm{wr}(\mathrm{y}, 1)), [\mathrm{x} \equiv 1]_2)$ 和 (ii) $\mathrm{x}_{\uparrow 2} \subseteq \mathsf{wlp}(\mathrm{T}\ (1, \mathrm{wr}\ (\mathrm{y}, 1)), \mathrm{x}_{\uparrow 2})$。然后，利用图 6 中的合取性规则，即可证明该霍尔三元组。

对于 (i)，我们需要证明对每个值 $\mathrm{v} \neq 1$:

$$\mathsf{dis}(T(2, rd(x, r_2, v))) \subseteq \mathsf{wlp}(T(1, wr(y, 1)), \mathsf{dis}(2, rd(x, r_2, v)))$$

根据图 6 中的规则和 dis 的定义，我们得到

$$\begin{aligned}\mathsf{dis}(T(2, rd(x, r_2, v))) &\subseteq \mathsf{wlp}(T(2, rd(x, r_2, v)), \varnothing) \\ &\subseteq \mathsf{wlp}(T(2, rd(x, r_2, v)), \mathsf{wlp}(interf(1, wr(y, 1)), \varnothing)) \\ &\subseteq \mathsf{wlp}(T(2, rd(x, r_2, v)) \fatsemi interf(1, wr(y, 1)), \varnothing)\end{aligned}$$

应用 SV1 后，我们得到

$$\begin{aligned}&\mathsf{wlp}(T(2, rd(x, r_2, v)) \fatsemi interf(1, wr(y, 1)), \varnothing) \\ &\subseteq \mathsf{wlp}(interf(1, wr(y, 1)) \fatsemi T(2, rd(x, r_2, v)), \varnothing) \\ &\subseteq \mathsf{wlp}(interf(1, wr(y, 1)), \mathsf{wlp}(T(2, rd(x, r_2, v)), \varnothing)) \\ &\subseteq \mathsf{wlp}(interf(1, wr(y, 1)), \mathsf{dis}(T(2, rd(x, r_2, v))))\end{aligned}$$

现在，(i) 可由引理1和合取性规则得出。

对于 (ii)，我们使用 SV2，并对每个动作 a 得到 $\in \mathsf{Act}^x$

$$vmax(2, a) \subseteq \mathsf{wlp}(T(1, wr(y, 1)), vmax(2, a))$$

因此，(ii) 可由合取性规则得出。

6 相关工作

与我们最接近的工作是 Doherty 等人 [15] 提出的工作。他们给出了一种基于 Owicki-Gries 的通用公理化推理方法，并将其应用于多种弱内存模型（顺序一致性、C11 RAR 和 TSO）。由于我们展示了 PSO 在其公理化层级中的位置，我们的程序语义和公理化推理均基于他们的工作。我们在此基础上补充了针对 PSO 的语义，指出了哪些公理适用于 PSO，并将它们用于带栅栏的消息传递测试的正确性证明。

其他基于 Owicki-Gries、用于弱内存模型推理的方法由 [9, 11, 12, 27] 提出。Bila 等人 [9] 针对持久化 TSO 内存模型提出了一种基于 Owicki-Gries 的逻辑，而 Dalvandi 等人 [11, 12] 以及 Lahav 和 Vafeiadis [27] 则研究了 C11 内存模型。他们都没有提出通用的证明演算或专用于 PSO 的证明演算。

然而，也存在其他针对 PSO 的验证方法。Abdulla 等人 [1] 例如提出了一种在 TSO 和 PSO 下进行无状态模型检查的技术。该技术基于时间序迹，它们为程序执行定义了偏序关系。Alglave 等人 [5] 提出了一种转换技术，通过使用 SC 工具，在弱内存模型下验证程序。对于不同的内存模型（ TSO 、 PSO 、 POWER.....），他们定义了一种与其公理语义等价的抽象机。随后，他们在程序中找出所谓的不安全对，并将程序线性化。接着，顺序一致性分析器会验证这个新程序。Dan 等人 [13] 将 TSO 和 PSO 的存储缓冲区抽象为有界数组。对于这两种模型，他们能够将给定程序转换为一个在 SC 下行为近似相同的程序，并

随后也使用 SC Tools 对其进行验证。Xiao 等人 [33] 用进程代数通信顺序进程形式化 PSO 内存模型，并将这一形式化用于模型检查器。这些技术中有些并非只适用于 PSO。不过，它们并不具有通用性，因为很难将其应用于其他内存模型。

[4, 29, 19, 24]提出了更为通用的方法。Alglave 和 Cousot [4]提出了一种不变性证明方法，用于证明给定程序相对于给定内存模型及该程序的不变式规范是正确的。该方法首先证明所谓的通信规范足以保证程序的不变式。如果某个内存模型能够保证这种通信，则该程序在该模型下就是正确的。他们的方法与我们采用的方法不同。由于每个内存模型都需要各自的通信规范，因此，对于不同的模型，都必须重新完成冗长证明中的每一步。这使得他们的证明更加复杂。其他相关工作则侧重于模型检查。Ponce de Leon 等人 [29] 和 Gavrilenko 等人 [19] 提出了通用的有界模型检查器，将给定内存模型下的给定程序转换为 SMT 公式。之所以称其为通用，是因为其输入不仅包含程序，还包含以内存模型为基础、在 CAT 中形式化为一组关系的描述。最后，Kokologiannakis 等人 [24] 开发了一种通用模型检查器，将给定程序转换为执行图，以便根据具有公理语义的给定内存模型检查其正确性。

7 结论

本文展示了弱内存模型 PSO 如何实例化 [15] 的所有公理（仅有一条公理例外），从而实现与内存模型无关的推理。我们通过一个借助插入栅栏实现消息传递的微基准测试，展示了这种推理技术。

未来的工作包括将这些底层公理提升至基于视图的断言层面，以提供更抽象的推理层次。我们还计划考察其他弱内存模型，确认它们是否满足这些公理。

参考文献

1. Abdulla, P.A., Aronis, S., Atig, M.F., Jonsson, B., Leonardsson, C., Sagonas, K.：针对 TSO 和 PSO 的无状态模型检查。Acta Informatica 54(8), 789–818 (2017 0 2 10 1007 00236 016 0275)。https://doi.org/ . /s - - - . Adve, S.V., Gharachorloo, K.：共享内存一致性模型：教程。Computer 29 546611 3 12 66 76 1996 10 1109 2 ( ), – ( )。https://doi.org/ . / . . Alglave, J.：弱内存模型的形式化层级。Formal Methods Syst. Des. 41 5 4 2 178 210 2012 10 1007 10703 012 0161 ( ), – ( )。https://doi.org/ . /s - - - . Alglave, J., Cousot, P.：Ogre 和 Pythia：一种用于弱一致性模型的不变性证明方法。载于：Castagna, G., Gordon, A.D.（编）POPL。第 3–18 页。ACM (2017)。https://doi.org/10.1145/3009837.3009883

5. Alglave, J., Kroening, D., Nimal, V., Tautschnig, M.：通过程序变换对弱内存进行软件验证。载于：Felleisen, M., Gardner, P.（编）ESOP，ETAPS。《计算机科学讲义》，第 7792 卷，第 512–532 页。Springer (2013 6 28 6 10 1007 978 3 642 37036)。https://doi.org/ . / - - - - . Alglave, J., Maranget, L., Sarkar, S., Sewell, P.：Litmus：在硬件上运行测试。载于：Abdulla, P.A., Leino, K.R.M.（编）TACAS。《计算机科学讲义》，第 6605 卷，第 41–44 页。Springer (2011)。https://doi.org/10 9 5 7 1007 978 3 642 19835 . / - - - - . Atig, M.F., Bouajjani, A., Burckhardt, S., Musuvathi, M.：关于弱内存模型的验证问题。载于：Hermenegildo, M.V., Palsberg, J.（编）POPL。第 7 1706303 8 18 2010 10 1145 1706299 – 页。ACM ( )。https://doi.org/ . / . . Batty, M., Owens, S., Sarkar, S., Sewell, P., Weber, T.：C++ 并发的数学化。载于：POPL。第 55 1926394 9 66 2011 10 1145 1926385 – 页 ( )。https://doi.org/ . / . . Bila, E.V., Dongol, B., Lahav, O., Raad, A., Wickerson, J.：针对持久化 x86-TSO 的基于视图的 Owicki-Gries 推理。载于：Sergey, I.（编）ESOP。《计算机科学讲义》，第 13240 卷，第 234–261 页。Springer (2022)。https://doi.org/10 8 9 10 1007 978 3 030 99336 . / - - - - . Cho, K., Lee, S., Raad, A., Kang, J.：革新硬件持久性模型：面向 Intel-x86 和 Armv8 的基于视图和公理化持久性模型。载于：Freund, S.N., Yahav, E.（编）PLDI。第 16–31 页。ACM (2021)。https://doi.org/10 3454027 11 1145 3453483 . / . . Dalvandi, S., Doherty, S., Dongol, B., Wehrheim, H.：针对 C11 RAR 的 Owicki-Gries 推理。载于：Hirschfeld, R., Pape, T.（编）ECOOP。第 11:1–11:26 页。LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020)。https://doi.org/10 11 12 4230 2020 . /LIPIcs.ECOOP. . . Dalvandi, S., Dongol, B., Doherty, S., Wehrheim, H.：将面向 C11 风格内存模型的 Owicki-Gries 推理集成到 Isabelle/HOL 中。J. Autom. Reason. 66(1), 141–171 (2022 2 13 10 1007 10817 021 09610)。https://doi.org/ . /s - - - . Dan, A.M., Meshman, Y., Vechev, M.T., Yahav, E.：宽松内存模型下验证的有效抽象。载于：D'Souza, D., Lal, A., Larsen, K.G.（编）VMCAI。《计算机科学讲义》，第 8931 卷，第 449–466 页。Springer (2015 8 25 14 10 1007 978 3 662 46081)。https://doi.org/ . / - - - - . Dijkstra, E.W.：《程序设计的纪律》。Prentice-Hall (1976)，https://www.worldcat.org/oclc/01958445 15. Doherty, S., Dalvandi, S., Dongol, B., Wehrheim, H.：统一操作式弱内存验证：一种公理化方法。《ACM Trans. Comput. Log.》23(4), 27 3545117 16 1 27 39 2022 10 1145 : – : ( )。https://doi.org/ . / . Doherty, S., Dongol, B., Wehrheim, H., Derrick, J.：以操作方式验证 C11 程序。载于：PPoPP。第 355 3295702 17 365 2019 10 1145 3293883 – 页 ( )。https://doi.org/ . / . . Flur, S., Gray, K.E., Pulte, C., Sarkar, S., Sezgin, A., Maranget, L., Deacon, W., Sewell, P.：以操作方式建模 ARMv8 架构：并发与指令集架构。载于：Bodík, R., Majumdar, R.（编）POPL。第 608–621 页。ACM (2016)。https://doi.org/10 2837615 18 1145 2837614 . / . . Furbach, F., Meyer, R., Schneider, K., Senftleben, M.：内存模型感知测试：统一的复杂度分析。《ACM 嵌入式计算系统汇刊》14(4), 63 1 63 25 2015 10 1145 2753761 : – : ( )。https://doi.org/ . / . 19 Gavrilenko, N., de Leon, H.P., Furbach, F., Heljanko, K., Meyer, R.：弱内存模型的有界模型检查：用于紧凑 SMT 编码的关系分析。载于：Dillig, I., Tasiran, S.（编）CAV。《计算机科学讲义》，第 11561 卷，第 355–365 页。Springer (2019)。https://doi.org/10.1007/978-3-030-25540-4 19

20. Hoare, C.A.R.：《计算机程序设计的公理基础》。《ACM通讯》12，363–380（1969）。https://doi.org/10.1145/363235.363259。/ Inc., S.I.：《SPARC架构模型》，版本（）。22. Kaiser, J., Dang, H., Dreyer, D., Lahav, O., Vafeiadis, V.：《弱内存的强逻辑：在 Iris 中推理释放-获取一致性》。载于：ECOOP。第 17:1–17:17 页（2017）。https://doi.org/10.4230/LIPIcs.ECOOP.2017.17。23. Kang, J., Hur, C., Lahav, O., Vafeiadis, V., Dreyer, D.：《宽松内存并发的承诺语义》。载于：Castagna, G., Gordon, A.D.（编）POPL。第 175–189 页。ACM（2017）。https://doi.org/10.1145/3009837.3009850。24. Kokologiannakis, M., Raad, A., Vafeiadis, V.：《弱一致性库的模型检查》。载于：McKinley, K.S., Fisher, K.（编）PLDI。第 96–110 页。ACM（2019）。https://doi.org/10.1145/3314221.3314609。25. Lahav, O., Boker, U.：《关于因果一致共享内存，有哪些问题是可判定的？》《ACM程序设计语言与系统汇刊》44(2)，8:1–8:55（2022）。https://doi.org/10.1145/3505273。26. Lahav, O., Giannarakis, N., Vafeiadis, V.：《驯服释放-获取一致性》。载于：Bod´ık, R., Majumdar, R.（编），第 43 届 ACM SIGPLAN-SIGACT 编程语言原理年会论文集，POPL 2016，美国佛罗里达州圣彼得堡，2016 年 1 月 20—22 日。第 649–662 页。ACM（2016）。https://doi.org/10.1145/2837614.2837643。27. Lahav, O., Vafeiadis, V.：《弱内存模型的 Owicki-Gries 推理》。载于：Halld´orsson, M.M., Iwama, K., Kobayashi, N., Speckmann, B.（编）ICALP。《计算机科学讲义》第 9135 卷，第 311–323 页。Springer（2015）。https://doi.org/10.1007/978-3-662-47666-25。28. Lamport, L.：《如何构建能够正确执行多处理器程序的多处理器计算机》。《IEEE计算机汇刊》28(9)，690–691（1979）。https://doi.org/ 10 1T09 1979 1675439 ./TC.。29. de Leon, H.P., Furbach, F., Heljanko, K., Meyer, R.：《将内存模型作为模块的有界模型检查》。载于：Bjørner, N.S., Gurfinkel, A.（编）FMCAD。第 1–9 页。IEEE（2018）。https://doi.org/10.23919/FMCAD.2018.8603021。30. Lynch, N.A., Vaandrager, F.W.：《前向与后向模拟：I. 无时序系统》。《信息与计算》121(2)，214–233（1995）。https://doi.org/10.1016/inco.1995.1134。31. Owicki, S.S., Gries, D.：《并行程序的公理化证明技术 I》。《Acta Informatica》6，319–340（1976）。https://doi.org/10.1007/BF00268134。32. Sarkar, S., Sewell, P., Nardelli, F.Z., Owens, S., Ridge, T., Braibant, T., Myreen, M.O., Alglave, J.：《x86-CC 多处理器机器代码的语义》。载于：Shao, Z., Pierce, B.C.（编）POPL。第 379–391 页。ACM（2009）。https://doi.org/10.1145/1480881.1480929。33. Xiao, L., Zhu, H., Xu, Q., Vinh, P.C.：《使用 CSP 对 PSO 内存模型进行建模与验证》。《移动网络与应用》27(5)，2068–2083（2022）。https://doi.org/10.1007/s11036-022-01989-5

29

证明

在附录中，我们证明第4节的引理和定理。第一部分证明 TS_{PP} 和 TS_{PSO} 轨迹等价。然后在第二部分中，我们利用这一点证明主要结果，即 PSO 满足 Core、SharedVars 和 Fences 的公理。

A.1 正向与反向模拟

设 TS_{PP} =(动作集 Σ_{PP}, I_{PP}, T_{PP}) 和 TS_{PSO} =(动作集 Σ_{PSO}, I_{PSO}, T_{PSO})。

定理 1 的证明。令 $\mathrm{F} \subseteq \Sigma_{PP} \times \Sigma_{PSO}$，其中

$$(\sigma_1, \sigma_2) \in F :\Leftrightarrow s_1 = s_2 \wedge \forall\, t \in \mathsf{Tid}\ \forall\, x \in \mathsf{Var_G} : wb_2^{t,x} = wb_1^{t,x}.val$$

满足 σ_1 = (s_1,wb_1)，且 σ_2 = (s_2,wb_2)。我们要证明存在从 TS_{PP} 到 TS_{PSO}的正向模拟。因此，我们先考察初始状态。令 σ_{01} =(s_{01}, wb_{01}) ∈I_{PP}。我们需要证明存在一个 σ_{02} ∈ I_{PSO}，使得 (σ_{01},σ_{02}) ∈F。令 σ_{01} =(s_{01}, wb_{01}) ∈I_{PP}，且 σ_{02} =(s_{02}, wb_{02})，其中 s_{02} =s_{01}，并且 $\mathrm{wb}^{t,\ x}_{02}$ =$\mathrm{wb}^{t,\ x}_{01}$.val。于是 σ_{02} ∈I_{PP}，且 (σ_{01},σ_{02}) ∈F。

现在我们证明

$$\begin{aligned}&\forall (\sigma_1, \sigma_2) \in F\ \ \forall\, a \in \mathsf{Act} : (\sigma_1, \sigma_1') \in T_{PP}(t, a)\\ &\Rightarrow \exists\, \sigma_2' : (\sigma_2, \sigma_2') \in T_{PSO}(t, a) \wedge (\sigma_1', \sigma_2') \in F.\end{aligned}$$

令 (σ_1,σ_2) ∈F，其中 σ_1 =(s_1, wb_1) ∈ Σ_{PP}且 σ_2 =(s_2, wb_2) ∈ Σ_{PSO}。令 σ_1' =(s_1', wb_1') ∈ Σ_{PP}，且 (σ_1,σ_1') ∈T_{PP}(t, a)。如果 (σ_1,σ_1'') ∈FL_{PP}且 (σ_2,σ_2'') ∈ FL_{PSO}，那么两条刷新规则以相同方式改变 s，并删除 wb^{t}、$_1{}^{x}$和 wb^{t}、$_2{}^{x}$中的首个条目，则 (σ_1'',σ_2'') ∈F。因此，我们只需证明对于 $\sigma_1''\ {}^{a}\!\rightsquigarrow^{t}_{PP}\ \sigma_1'$ 和 $\sigma_2''\ {}^{a}\!\rightsquigarrow^{t}_{PSO}\ \sigma_2'$，($\sigma_1'$, σ_2') ∈ F:

– 情形 1：一个 =读操作(x , r , v)。 $\overset{a\ t}{\rightsquigarrow}_{PP,}$ 且 $\overset{a\ t}{\rightsquigarrow}_{PSO,}$ 不会改变 σ_1''或 σ_2''。因此 (σ_1',σ_2') =(σ_1'',σ_2'') ∈F。– 情形 2：一个 = 栅栏。如果状态 σ_1'' 满足 PP 内存屏障规则的条件，那么 σ_2''也满足 PSO 内存屏障规则的条件。执行栅栏操作后，这两个状态都不会改变，因此 (σ_1',σ_2') =(σ_1'',σ_2'') ∈F。

引理 4 的证明。设 $\sigma \in \Sigma_{PP}$ 和 $\sigma_0 \in \mathrm{I}_{PP}$，其中 $\sigma_0 \overset{\alpha}{\Rightarrow} \sigma$。那么存在 a_1，...，$\mathrm{a}_k \in$ 动作集 ∪{刷新} 以及 σ_1，...，$\sigma_{k-1} \in \Sigma_{PP}$，使得

$$\sigma_0 \overset{a_1}{\rightsquigarrow}_{PP}\ \sigma_1 \overset{a_2}{\rightsquigarrow}_{PP}\ \ldots\ \overset{a_{k-1}}{\rightsquigarrow}_{PP}\ \sigma_{k-1} \overset{a_k}{\rightsquigarrow}_{PP}\ \sigma.$$

由于 σ_0 ∈I_{PP}，σ_0.$\mathrm{wb}^{t',x'}$ 对每个 t′ ∈Tid 和 x ′ ∈$\mathsf{Var_G}$ 都为空。写缓冲区只会在执行写操作或刷新操作后发生变化。设 σ_i 是序列中第一个写缓冲区非空的状态 σ_i.wb^{t_i,x_i}，其中 i ∈ {1, ..., k − 1}、t_i ∈Tid 且 x_i ∈$\mathsf{Var_G}$。那么 a_i =wr (x_i, v_i) 必定是一个写操作，并且σ_i.wb^{t_i,x_i} = ⟨(v_i, q_i)⟩ 对某个 $\mathrm{q}_i \in \mathbb{Q}$ 成立。因此 σ_i 满足不变式。设 $\sigma_{j-1} \in \Sigma_{PP}$ 满足 i + 1 ≤ j < k、σ_{j-1}.wb ≠ σ_j.wb 和 σ_{j-1}，且序列中 σ_{j-1}之前的每个状态都满足不变式。现在我们证明，对于每个动作 a_j，σ_j 都满足不变式。由于读操作和栅栏操作不会改变 σ_{j-1}，我们只需考察写操作和刷新操作。于是，序列中的每个状态，尤其是 σ，都满足不变式，引理得证。

– 情形 1：a_j =wr(x_j, v_j)。在这种情况下，fresh$^{\sigma_{j-1}}$(t_j, x_j, q_j) = 为 true，这意味着

$$(\forall\, t' \in \mathsf{Tid}\ \forall\, x' \in \mathsf{Var_G} : (\cdot, q_j) \notin \sigma_{j-1}.wb^{t',x'}) \\ \wedge(\forall(\cdot, q') \in \sigma_{j-1}.wb^{x_j,t_j} : q_j > q').$$

因此，我们得到 σ_j

$$\forall\, t' \in \mathsf{Tid}\ \forall\, x' \in \mathsf{Var_G}, t' \neq t_j \vee x' \neq x_j : (\cdot, q_j) \notin \sigma_j.wb^{t',x'} \\ \wedge\, \forall(\cdot, q') \in \sigma_j.wb^{x_j,t_j}, (\cdot, q') \neq last(\sigma_j.wb^{t_j,x_j}) : q_j > q'.$$

由于 a_j 只会改变 $wb^{t_j,\ x_j}$，且 σ_{j-1} 已满足不变式，因此 σ_j 也满足不变式。

– 情况 2：a_j = 刷新。此时，$\sigma_j.wb^{t_j,x_j}$ 比 $\sigma_{j-1}.wb^{t_j,x_j}$ 少一个条目。如果 $\sigma_{j-1}.wb^{t_j,x_j}$ 只有一个条目，那么 $\sigma_j.wb^{t_j,x_j}$ 为空，并且 σ_j 满足不变式。如果 $\sigma_{j-1}.wb^{t_j,x_j}$ 多于一个条目，则有 ts(last($\sigma_j.wb^{t_j,x_j}$)) =ts(last ($\sigma_{j-1}.wb^{t_j,x_j}$))，且 {(·, q′) ∈ $\sigma_j.wb^{x_j,t_j}$ |(·, q′) ≠last ($\sigma_j.wb^{t_j,x_j}$)} ⊆{(·, q′) ∈ $\sigma_{j-1}.wb^{x_j,t_j}$ |(·, q′) ≠ last ($\sigma_{j-1}.wb^{t_j,x_j}$)}。因此，$\sigma_j$ 满足不变式。

定理 2 的证明。令 B $\subseteq \Sigma_{PSO} \times \Sigma_{PP}$，其中

$$(\sigma_1, \sigma_2) \in B :\Leftrightarrow s_1 = s_2 \wedge \forall\, t \in \mathsf{Tid}\ \forall\, x \in \mathsf{Var_G} : wb_1^{t,x} = wb_2^{t,x}.val.$$

该关系在 Σ_{PSO} 上是全定义的。我们要证明存在从 TS_{PSO} 到 TS_{PP} 的反向模拟。因此，我们先考察初始状态。令 $\sigma_{01} \in I_{PSO}$。我们需要证明，对于每个满足 (σ_{01}, σ_{02}) ∈B 的 $\sigma_{02} \in \Sigma_{PP}$，都有 $\sigma_{02} \in I_{PP}$。令 σ_{01} =(s_{01}, wb_{01})。那么，对于每个 t ∈Tid 和 x ∈ $\mathsf{Var_G}$，$wb^{t,\ x}_{01} = \langle\,\rangle$。因此，对于每个满足 ($\sigma_{01}, \sigma_{02}$) ∈ B 的 σ_{02}，所有写缓冲区均为空，所以 $\sigma_{02} \in I_{PP}$。现在我们证明，对于每个 (σ_1', σ_2') ∈B 以及每个动作 a

$$(\sigma_1, \sigma_1') \in T_{PSO}(t, a) \Rightarrow \exists\, \sigma_2 : (\sigma_2, \sigma_2') \in T_{PP}(t, a) \wedge (\sigma_1, \sigma_2) \in B$$

令 (σ_1', σ_2') ∈B，其中 σ_1' =(s_1', wb_1') ∈ Σ_{PSO} 和 σ_2' =(s_2', wb_2') ∈ Σ_{PP}。令 σ_1 =(s_1, wb_1) ∈ Σ_{PSO}，且 (σ_1, σ_1') ∈T_{PSO}(t, a)。那么，存在一个状态σ_1'' =(s_1'', wb_1'')，使得 (σ_1, σ_1'') ∈FL_{PSO} 且 $\sigma_1''\ \overset{a,\ t}{\rightsquigarrow}_{PSO}\ \sigma_1'$。首先，我们证明，对于每个 a ∈ Act，都存在一个 $\sigma_2'' \in \Sigma_{PP}$，满足 $\sigma_2''\ \overset{a,\ t}{\rightsquigarrow}_{PP}\ \sigma_2'$ 且 (σ_1'', σ_2'') ∈B。

– 情形 1：一个 =rd (x , r , v)。读操作不会改变 σ_1''和 σ_2''。因此$\sigma_1'' = \sigma_1'$和 $\sigma_2'' = \sigma_2'$，所以 (σ_1'', σ_2'') ∈B 。– 情形 2：一个 = 栅栏。如果 σ_1'' 满足 PSO 内存屏障规则的条件，那么 σ_2'' 满足 PP 内存屏障规则的条件。栅栏操作不会改变 σ_1'' 和 σ_2''。因此$\sigma_1'' = \sigma_1'$ 和 $\sigma_2'' = \sigma_2'$，所以 (σ_1'', σ_2'') ∈B 。– 情形 3：一个 =wr (x , v)。在这种情况下，$\sigma_1'.wb^{t,\ x} = \sigma_1''.wb^{t,\ x} \cdot \langle v \rangle$。由于 ($\sigma_1', \sigma_2'$) ∈B，我们有 last($\sigma_1'.wb^{t,x}$) =v =val (last ($\sigma_2'.wb^{t,x}$))。令 q =ts(last ($\sigma_2'.wb^{t,x}$))。现在可以应用引理 4，得到

$$\exists\, \sigma_2'' \in \Sigma_{PP} : fresh_{\sigma_2''}(t, x, q) = true \wedge \sigma_2''\ \overset{a,t}{\rightsquigarrow}_{PP}\ \sigma_2'$$

由于这两个写操作都通过添加一个新条目来改变 $wb^{t,x}$，因此我们得到 $(\sigma_1'', \sigma_2'') \in B$。

现在我们需要证明存在 $\sigma_2 \in \Sigma_{PP}$，使得 $(\sigma_2, \sigma_2'') \in FL_{PP}$，且 $(\sigma_1, \sigma_2) \in B$。对于 $\sigma_1 = \sigma_1''$，我们设定 $\sigma_2 = \sigma_2''$。如果 $\sigma_1 \neq \sigma_1''$，令 n 表示 σ_1 与 σ_1''之间的刷新操作数量:

$$\sigma_1 = \sigma_{1,n} \overset{flush,t_n}{\rightsquigarrow}_{PSO} \sigma_{1,n-1} \overset{flush,t_{n-1}}{\rightsquigarrow}_{PSO} \cdots \overset{flush,t_2}{\rightsquigarrow}_{PSO} \sigma_{1,1} \overset{flush,t_1}{\rightsquigarrow}_{PSO} \sigma_{1,0} = \sigma_1''$$

这里 $\sigma_{1,1}, \ldots, \sigma_{1,n-1} \in \Sigma_{PSO}$，并且对于 $1 \leq i \leq n$，wb^{t_i,x_i} 由 $\overset{flush}{\rightsquigarrow}_{PSO}^{t_i}$ 刷新，其中 $x_i \in Var_G$。每个刷新操作 $\overset{flush}{\rightsquigarrow}_{PSO}^{t_i}$ 都会删除 wb^{t_i,x_i}的第一个条目，并改变 $s(x_i)$ 的值。令 $\sigma_{2,0} = \sigma_2''$和 $\sigma_{2,i} \in \Sigma_{PP}$，其中 $\sigma_{2,i}.wb^{t,x} = \sigma_{2,i-1}.wb^{t,x}$，对于每个 $\sigma_{2,i}.wb^{t,x} \neq \sigma_{2,i}.wb^{t_i,x_i}$均成立。对于每个条目 $\sigma_{2,i_1}.wb^{t_i,x_i}(j)$，我们设定 $\sigma_{2,i}.wb^{t_i,x_i}(j+1) = \sigma_{2,i_1}.wb^{t_i,x_i}(j)$；并令 $\sigma_{2,i}.wb^{t_i,x_i}(0) = (\sigma_{2,i-1}.s(x_i), q)$，其中时间戳 q 小于 $\sigma_{2,i-1}$的每个写缓冲区中的所有时间戳。然后

$$\sigma_{2,n} \overset{flush,t_n}{\rightsquigarrow}_{PP} \sigma_{2,n-1} \overset{flush,t_{n-1}}{\rightsquigarrow}_{PP} \cdots \overset{flush,t_2}{\rightsquigarrow}_{PP} \sigma_{2,1} \overset{flush,t_1}{\rightsquigarrow}_{PP} \sigma_{2,0} = \sigma_2''$$

并且对于每个 i，$(\sigma_{1,i}, \sigma_{2,i}) \in B$。因此，如果我们设 $\sigma_2 = \sigma_{2,0}$，就有 $(\sigma_2, \sigma_2'') \in FL_{PP}$并且 $(\sigma_1, \sigma_2) \in B$。

A.2 公理证明

本节通过证明 PPSO 满足层级 Core, SharedVars 和 Fences的每条公理，来证明定理 3。

Core 公理

C1 的证明。

– 一个 = 栅栏。然后 $vmax_{PP}(t, a) = \Sigma_{PP} \supseteq I_{PP}$。– 一个 $\in$ 写操作 $\cup$ 读操作，其中 $var(a) = x$。在这种情况下

$$\begin{aligned} vmax_{PP}(t, a) &= maxTS(t, x) \\ &\supseteq \{\sigma \in \Sigma_{PP} \mid \forall\, t' \in Tid : \sigma.wb^{t',x} = \langle\,\rangle\} \\ &\supseteq I_{PP}. \end{aligned}$$

C2 的证明。

– 一个 = 栅栏或一个 = 读操作 (x , r , v)。那么 $T_{PP}(t, a) = FL = T_{PP}(t', a)$。令 $\sigma \in vmax_{PP}(t, a)$，并令 $(\sigma, \sigma') \in T_{PP}(t, a)$。对于 $\tau = \sigma'$，我们得到 $(\sigma'\tau) \in \beta$，且 $(\sigma, \tau) \in T_{PP}(t', a) \subseteq T_{PP}(t', a) \circ_g \beta$。– 一个 = 写操作 (x , v)。令 $\sigma \in vmax_{PP}(t, a) = maxTS(t, x)$，且 $(\sigma, \sigma') \in T_{PP}(t, a)$。令 $(\sigma', \tau) \in FL$，其中我们刷新的频率至少满足使 $\sigma'.wb^{t,x}$ 变为空所需的频率。因此 $(\sigma', \tau) \in \beta$，且 $\tau.s(x) = v$。由于 $\sigma \in maxTS(t, x)$，我们得到 $\tau.wb^{t'',x} = \langle\,\rangle$，对所有 $t'' \in Tid$ 均成立。因此 $(\sigma, \tau) \in T_{PP}(t', a) \circ_g \beta$。

C3 的证明。

– 一个 $=$栅栏或一个 $\in$读操作。那么 $\text{interf}_{PP}(t, a) = \text{id}$，因此 $T_{PP}(t, a) = FL \subseteq \beta \subseteq \beta \circ_9 \text{interf}_{PP}(t, a) \fatsemi \beta$。– 一个 $=$写操作 (x , v)。在这种情况下，$\beta \circ_9 \text{interf}_{PP}(t, a) \fatsemi \beta \supseteq FL \circ_9 T_{PP}(t, a) \circ_9 FL \supseteq T_{PP}(t, a)$。

C4 的证明。

– b 是栅栏或 b 是读操作。于是 $\text{interf}_{PP}(t, b) = \text{id}$，因此 $\text{wlp}(\text{interf}_{PP}(t, b), vmax_{PP}(t, a)) = vmax_{PP}(t, a)$。– b 是写操作 (x , v)。如果 a 是栅栏，那么 $vmax_{PP}(t, a) = \Sigma_{PP}$。因此 $vmax_{PP}(t, a) = \text{wlp}(\text{interf}_{PP}(t, b)\ vmax_{PP}(t, a))$。对于 a 是写操作 $\cup$读操作，让 $\sigma \in vmax_{PP}(t, a)$。由于 $\text{interf}_{PP}(t, b)$ 只能向 $\text{wb}^{t,x}$ 添加一个条目，因此得到 $\sigma' \in vmax_{PP}(t, a)$，从而 $\sigma \in \text{wlp}(\text{interf}_{PP}(t, b)\ vmax_{PP}(t, a))$。

SharedVars 公理

SV1 的证明。

– b 是栅栏或 b 是读操作。于是 $\text{interf}_{PP}(t, b) = \text{id}$，因此 $\text{interf}_{PP}(t', b) \circ_9 T_{PP}(t, a) \subseteq T_{PP}(t, a) \circ_9 \text{interf}_{PP}(t', b)$。– b 是写操作 (x , v)。如果 a 是写操作，那么 $T_{PP}(t, a) = FL$。因此，我们只需考虑 a 是写操作 (y, u)。由于 $var(a) \neq var(b)$，我们可以交换这两个写操作。

SV2 的证明。

– b $=$是栅栏或 b $\in$是读操作。于是干扰$_{PP}$(t′, b) $=$为 id，因此最弱前置条件$(\text{interf}_{PP}(t', b)\ vmax_{PP}(t, a)) = vmax_{PP}(t, a)$。– b $=$ 是对 (x , v) 的写操作。如果 a $=$ 是栅栏，那么 $vmax_{PP}(t, a) = \Sigma_{PP}$。因此 $vmax_{PP}(t, a)$=最弱前置条件$(\text{interf}_{PP}(t', b)\ vmax_{PP}(t, a))$。对于 a $\in$是写操作、$\cup$是读操作的情况，令 $\sigma \in vmax_{PP}(t, a)$。由于干扰$_{PP}$(t ′, b) 只能向 $\text{wb}^{t'}$ 添加一个条目, ^{x}并且 $var(a) \neq$x，我们得到$\sigma' \in vmax_{PP}(t, a)$，因此 $\sigma \in$最弱前置条件$(\text{interf}_{PP}(t, b)\ vmax_{PP}(t, a))$。

RW1 的证明。由于 a_r 是一次读操作，我们得到

$$\begin{aligned} interf_{PP}(t', a_w) \fatsemi T_{PP}(t, a_r) &= T_{PP}(t', a_w) \fatsemi FL \\ &= FL \fatsemi \overset{a_w, t'}{\rightsquigarrow}_{PP} \fatsemi FL \\ &= FL \fatsemi T_{PP}(t', a_w) \fatsemi FL \\ &= T_{PP}(t, a_r) \fatsemi interf_{PP}(t', a_w). \end{aligned}$$

RW2 的证明。由于 a_r是一次读干扰$_{PP}$(t′, a_r) $=$id 。因此

$$interf_{PP}(t', a_r) \fatsemi T_{PP}(t, a) = T_{PP}(t, a) \fatsemi interf_{PP}(t', a_r).$$

RW3 的证明。由于 a_r是一个读操作干扰$_{PP}$(t′, a_r) $=$id。因此

$$vmax_{PP}(t, a) \subseteq \text{wlp}(interf_{PP}(t', a_r), vmax_{PP}(t, a)).$$

RW4 的证明。我们有

$$\begin{aligned}\mathrm{dom}(T_{PP}(t, Rd_{|x})) &= \{\sigma \mid \exists\, v \in \mathsf{Val} : \sigma.s(x) = v \vee wbVal_{\sigma}(t, x) = v\} \\ &= \Sigma_{PP}.\end{aligned}$$

RW5 的证明。设 $\sigma \in \Sigma_{PP}$with (σ, σ') $\in$T$_{PP}$(t, a$_w$)成立。则 wbVal$_{\sigma'}$(t, x) = v 或 s(x) =v。因此 σ' $\in$dom(T$_{PP}$(t, 读操作$_{|x}$[v])), 从而

$$\Sigma_{PP} \subseteq \mathsf{wlp}(T_{PP}(t, a_w), \mathrm{dom}(T_{PP}(t, Rd_{|x}[v]))).$$

RW6 的证明。由于 x$^{\uparrow t}$ = $\bigcap_{a\in动作集_{PP|x}} vmax_{PP}$(t, a) =maxTS (t, x), 并且

$$\begin{aligned}\bigcup_{v\in\mathsf{Val}} [x \equiv v]_t &= \bigcup_{v\in\mathsf{Val}} \bigcap_{u\neq v} [x \not\approx v]_t \\ &= \bigcup_{v\in\mathsf{Val}} \bigcap_{u\neq v} \mathsf{dis}(T_{PP}(t, Rd_{|x}[u])) \\ &= \bigcup_{v\in\mathsf{Val}} \{\sigma \mid \sigma.s(x) = v \vee wbVal_{\sigma}(t, x) = v\} \\ &= \Sigma_{PP}\end{aligned}$$

我们得到 x$^{\uparrow t}$ $\subseteq \bigcup_{v\in值}$[x ≡v]$_t$。

RW7 的证明。令 a$_w$ =wr (x , v) 且 a$_r$ =rd (x , r , v)。对于 σ $\in$maxTS (t , x), 若σ.wb^t, $^x \neq \langle\rangle$, 则 $\sigma \notin$dis(T$_{PP}$(t′, a$_r$))。因此, 对于每个 $\sigma \in vmax_{PP}$(t , a$_w$) $\cap$dis(T$_{PP}$(t′, a$_r$)), 我们得到 σ.wb$^{t,x} = \langle\rangle$。于是, 对于所有 (σ, σ') $\in$FL, 都有 σ'.s(x) $\neq$v。令 (σ, σ') $\in$T $_{PP}$(t, a$_w$)。那么 σ'.wb$^{t,x} = \langle$(v , q)$\rangle$, 并且由于 t $\neq$t ′, 我们得到σ''.wb$^{t,x} = \langle\rangle$, 其中 (σ', σ'') $\in$T$_{PP}$(t′, a$_r$)。因此, $\sigma'' \in vmax_{PP}$(t′, a) 且$\sigma \in$最弱前置条件(T$_{PP}$(t, a$_w$), 最弱前置条件(T$_{PP}$(t ′, a$_r$) $vmax_{PP}$(t ′, a))。

FENCES 公理

FNC 的证明。若存在一个 =栅栏, $vmax_{PP}$(t , a) = Σ_{PP} = $vmax_{PP}$(t ′, a), 因此 $vmax_{PP}$(t, a) =最弱前置条件(T$_{PP}$(t , 栅栏) $vmax_{PP}$(t ′, a))。令 a $\in$写操作 $\cup$读操作, 且 var(a) =x 。则 $vmax_{PP}$(t , a) =maxTS (t , x)。令 $\sigma \in vmax_{PP}$(t , a)。

– σ.wb$^{t''}$,$^x = \langle\rangle$对所有 t ″ $\in$Tid。对于 (σ, σ') $\in$T$_{PP}$(t , 栅栏), 我们仍有 σ'. wb$^{t''}$,$^x = \langle\rangle$对所有 t ″ $\in$Tid, 因此 $\sigma' \in vmax_{PP}$(t′, a)。于是 σ $\in$最弱前置条件(T$_{PP}$(t, 栅栏) $vmax_{PP}$(t ′, a))。– σ.wb$^{t,x} \neq \langle\rangle$。令 ($\sigma, \sigma'$) $\in$T$_{PP}$(t , 栅栏)。对于 $\overset{fence}{\rightsquigarrow}{}^{t}{}_{PP}$, wb^{t,x}必须为空。由于 σ $\in$maxTS (t , x), 我们需要先刷新 wb$^{t''}$,^{x}对所有 t ″ $\in$Tid, 然后才能刷新 wb^{t,x}。接着 σ'.wb$^{t''}$,$^x = \langle\rangle$对所有 t″ $\in$ Tid, $\sigma' \in vmax_{PP}$(t ′, a), 且 $\sigma \in$最弱前置条件(T$_{PP}$(t, 栅栏) $vmax_{PP}$(t ′, a))。