arXiv:2301.10032v1 [cs.LO] 24 Jan 2023

有界活性规约综合中的大界驯服（完整版）*

Philippe Heim 和 Rayna Dimitrova

CISPA 亥姆霍兹信息安全中心，德国萨尔布吕肯{philippe.heim、dimitrova}@cispa.de

摘要 从时序逻辑规约自动综合，是手动系统设计的一种颇具吸引力的替代方案，因为它能够根据高层规约生成正确性由构造保证的实现。由于综合问题的复杂度很高，研究人员投入了大量精力，为受限的规范语言片段开发实际高效的方法。本文聚焦于语法上受限的线性时序逻辑（LTL）安全片段 *extended with bounded temporal operators*。我们提出一种新的合成方法，其主要目标是高效解决含有有界时序算子的规约综合问题，尤其是具有大界限的规约。对该方法的实验评估表明，在处理这类规约时，它的表现优于最先进的综合工具，说明该方法有望高效处理安全规范中的定量时序约束。

1 引言

反应式综合 [8] 旨在根据描述反应式系统所需行为的形式化规范，自动生成实现。系统需求通常使用线性时序逻辑（LTL）等时序逻辑来描述。时序逻辑是一类表达能力强的高级规范语言，能够描述丰富的性质，例如机器人任务 [16]。反应式系统的规约通常包含“某件好事最终会发生”这类要求。这些要求可以通过 LTL 中的时序算子 $\mathcal{U}$ （“直到”）和 $\Diamond$（“最终”）来表达。“最终”是一种抽象，表示在系统执行的未来某个未知时间点，某项性质成立。虽然这种抽象有助于避免过度规约，但在许多情况下，实际中对满足要求的时限存在约束。在这种情况下，合成过程必须检查这些时序要求是否可实现，并综合出符合这些界限的实现。

举一个简单的例子，考虑一份规约，用于描述办公楼前门控制器的期望行为。我们的规约规定，

* Philippe Heim 在德国萨尔兰大学攻读博士期间完成了这项工作。

夜间门必须始终上锁，其他时间则必须解锁。规约还规定，发生火灾时，门最终应当打开。以这种方式表述时，该规约是可实现的。然而，如果夜间发生火灾，综合得到的实现只会在白天开始时打开门。显然，这并非我们期望的行为！我们可以在 LTL 中使用时序算子 （“下一步”）来准确规定期望的行为，它允许我们指定某个属性应在下一个时间步成立。然而，为了表达所需的时间界限，我们需要使用嵌套算子。这很快就会变得不便，尤其是在需要指定多个不同的时间界限，其中一些界限还很大时。只需将时序算子的有界版本作为语法糖添加进来，就能轻松避免这种建模上的不便和规约规模的增加，同时不增加表达能力。

鉴于其实际意义，那些（处于否定范式的）公式仅包含有界版本 $\mathcal{U}$ 和算子的线性时序逻辑片段受到了广泛关注。最突出的此类片段是安全线性时序逻辑，即否定范式下线性时序逻辑中不含直到算子的片段。由于安全线性时序逻辑是线性时序逻辑的一个句法片段，因此它只能通过嵌套下一时刻算子来表达有界活性性质。另一个值得注意的例子是扩展有界响应线性时序逻辑（$\mathrm{LTL}_{\mathsf{EBR}}$） [9], ，它是线性时序逻辑的一个片段，包含有界时序算子以及无界全称时序算子（即“全局算子”和“释放算子”）。虽然每个 $\mathrm{LTL}_{\mathsf{EBR}}$ 公式都可以用安全线性时序逻辑表示，但 $\mathrm{LTL}_{\mathsf{EBR}}$ 的一项重要优势在于，时序算子的界限以二进制形式表示，因此能够以简洁得多的形式表示公式，简洁度可达指数级。然而，在 [9] 中介绍的合成过程中，这些界限会被展开为嵌套的“下一步”算子。文献 [9] 将保持界限的符号形式视为未来发展的一个有趣方向。事实上，在许多具有实际意义的情形中，由于同一系统在不同时间尺度上的需求，大界限是不可避免的。

本文通过提出一种针对带有有界算子的安全线性时序逻辑扩展的合成过程，来应对这一挑战。我们开发了专门的技术，以符号化且高效地处理时序界。

贡献。我们提出了一种综合方法，用于处理以线性时序逻辑的一个片段表示的规约；该片段是在安全线性时序逻辑中加入有界时序算子后形成的语法扩展。我们方法的显著特点是将问题归约为一种专门的博弈模型，称为倒计时器博弈，其中通过引入计时器，以符号化方式处理时序算子的界限。该转换的其他特点包括：对所构造博弈中的边进行即时剪枝，以及减少引入的计时器数量。我们提出了一种基于抽象的方法来求解所得博弈。我们已开发出该方法的原型实现，实验评估表明，它确实能够高效处理具有大界限的安全规范。我们在一组包含具有大界限的有界时序算子的基准测试上展示了这一点：与最先进的线性时序逻辑工具以及线性时序逻辑综合工具相比，我们的方法表现更优。

相关工作。由于相较于一般的线性时序逻辑综合， *Safety LTL* 的综合问题在算法上更为简单，因此受到了广泛关注 [25]。例如， [25] 中提出的符号化方法被证明优于当时最先进的线性时序逻辑综合工具。对于线性时序逻辑$_{\mathsf{EBR}}$， [9] 提出了一种综合算法，该算法基于完全符号化地转换为确定性安全自动机。我们的方法与上述技术的一个关键区别在于，我们的倒计时器博弈构造不会预先展开有界时序算子，而是以符号化方式处理它们。此外， [25] 的作者指出，对于较大的安全性线性时序逻辑公式，构造确定性安全自动机会成为性能瓶颈。我们的安全博弈构造利用剪枝来缓解这一问题，即时消除博弈图中无需探索的部分。

参数化时序逻辑（例如 PLTL） [1] 能够对“全局”算子的满足时间以及“最终”算子的等待时间指定参数化的下界和上界。在逻辑 PROMPT-LTL [17], 中，只有最终性由上界参数化。这些逻辑中的时序算子的界限是未知参数，而在我们所研究的情形中，界限是给定的整数常数。我们的工作旨在开发一种综合方法，高效处理常数界限。

在实时环境中，允许限制时序算子时间范围的时序逻辑已得到广泛研究。值得注意的逻辑包括度量时序逻辑（MTL） [15], 及其片段度量区间时序逻辑（MITL） [2]。与非定时环境相比，根据实时逻辑规约进行综合面临额外挑战。MTL [4], 、MITL [5,11], 乃至 MTL 的安全片段 [5]上的控制器综合问题都是不可判定的。通过固定控制器所用资源（时钟和守卫） [5,12], 可以恢复可判定性。关键挑战在于，综合需要确定性自动机，而通常无法为 MITL 构造确定性定时自动机。为绕过这一问题，通常会假设有界变化性。在此假设下， [20] 提出了一种针对有界响应性质的综合算法，而 [23]则介绍了从 MTL 到确定性定时自动机的转换方法。关于工具支持， [6] 和 [18], 提出了针对 MTL 片段的可靠但不完备的综合方法，并将其实现于使用 UPPAAL-TIGA [3]求解定时博弈的工具链中。 [13]介绍了一种通过转换为交替定时自动机来实现 MTL 控制器综合的工具。如果实时综合问题被表述为定时博弈，且规约是基于状态的获胜条件，那么计算控制策略的问题是可判定的 [21]。针对定时博弈的高效在线算法已被开发出来 [7], ，并成功实现于 UPPAAL-TIGA [3] 和 UPPAAL-STRATEGO [10]。由于我们关注的是离散时间系统，因此将问题限定在离散时间领域，着重高效处理其中的定量时序约束，从而避开稠密时间环境中的额外挑战。

2 预备知识

反应式综合 设 $\mathcal{I}$ 为有限的、不可控的环境输入布尔命题集， $\mathcal{O}$ 为有限的、可控的输出布尔命题集。反应式系统是一个三元组（C， c_0， γ），其中 C 是控制状态集，$c_0 \in C$ 是初始控制状态，而 γ： $C \times 2^{\mathcal{I}} \to C \times 2^{\mathcal{O}}$ 是转移函数。规约是由 $\mathcal{I} \cup \mathcal{O}$上的无限字构成的语言 $\mathcal{L} \subseteq \left(2^{\mathcal{I} \cup \mathcal{O}}\right)^{\omega}$。

系统 (C, c_0, γ) 实现规约 $\mathcal{L}$，当且仅当对于环境输入的所有无限序列 $i \in \left(2^{\mathcal{I}}\right)^{\omega}$，它都会产生一个由 $(c_{t+1}, o_t) = \gamma(c_t, i_t)$（其中 $t \in \mathbb{N}$）定义的输出序列 $o \in \left(2^{\mathcal{O}}\right)^{\omega}$，使得 $i \cup o \in \mathcal{L}$。反应式综合是为给定规约寻找一个能够实现该规约的实现这一问题。

带有界活性算子的安全线性时序逻辑 我们考虑使用时序逻辑表达的规约，更具体地说，是线性时序逻辑（LTL）的一个片段 [24], ，我们将其记作 SafeLTL$_B$。片段 SafeLTL$_B$ 是安全线性时序逻辑 [25]的语法扩展，由以下语法定义：

$$\varphi, \psi := ap \mid \neg ap \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \bigcirc[n]\varphi \mid \Diamond[n]\varphi \mid \varphi \,\mathcal{W}[n]\psi \mid \varphi \,\mathcal{W}\, \psi$$

对于 ap $\in \mathcal{I} \cup \mathcal{O}$ 和 $n \in \mathbb{N}$。SafeLTL$_B$ 通过带有二进制编码界的有界算子扩展了安全线性时序逻辑。虽然所有有界算子都有等价的安全线性时序逻辑公式（例如 $[n]\varphi \equiv \bigvee_{i \in \{0 \ldots n\}} \bigcirc^i \varphi$），但这些公式的编码规模会大出指数级。常量 $\top$（真）、$\bot$（假）、“全局”算子和“有界直到” $\mathcal{U}[n]$ 分别可以导出为 $\top := a \vee \neg a$、 $\bot$: $\Box$ $\overline{\Box}\, a \wedge \neg a$、 $\Box\varphi := \varphi \,\mathcal{W}\, \bot$、$[n]\varphi := \varphi \,\mathcal{W}[n]\bot$，以及 $\varphi \,\mathcal{U}[n]\psi := (\varphi \,\mathcal{W}[n]\psi) \wedge \Diamond[n]\psi$。

公式 $\Phi \in$ SafeLTL$_B$ 被无限字 $w = w_0 w_1 \ldots \in \left(2^{\mathcal{I} \cup \mathcal{O}}\right)^{\omega}$ 在时间点 $k \in \mathbb{N}$ 的满足关系记为 $w \vDash_k \Phi$，定义如下：

$w \vDash_k a :\Leftrightarrow a \in w_k$ $w \vDash_k \neg a :\Leftrightarrow a \notin w_k w \vDash_k \varphi \wedge \psi :\Leftrightarrow (w \vDash_k \varphi) \wedge ($
$w \vDash_k \psi)\, w \vDash_k \varphi \vee \psi :\Leftrightarrow (w \vDash_k \varphi) \vee (w \vDash_k \psi) w \vDash_k \Diamond[n]\varphi :$
$\Leftrightarrow \exists i \leq n.\, w \vDash_{k+i} \varphi$ $w \vDash_k \bigcirc[n]\varphi :\Leftrightarrow w \vDash_{k+n} \varphi w \vDash_k \varphi \,\mathcal{W}[n]\psi :\Leftrightarrow ($
$\forall i \leq n. w \vDash_{k+i} \varphi) \vee (\exists j \leq n. w \vDash_{k+j} \psi \wedge \forall i < j. w \vDash_{k+i} \varphi) w \vDash_k \varphi \,\mathcal{W}\, \psi :\Leftrightarrow ($
$\forall i. w \vDash_{k+i} \varphi) \vee (\exists j. w \vDash_{k+j} \psi \wedge \forall i < j. w \vDash_{k+i} \varphi)$.

$\Phi \in$ SafeLTL$_B$ 的语言定义为 $\mathcal{L}(\Phi) := \{w \in \left(2^{\mathcal{I} \cup \mathcal{O}}\right)^{\omega} \mid w \vDash_0 \Phi\}$。

双人安全博弈 时序逻辑规约的综合问题可以通过将规约转换为系统与环境之间的双人博弈来求解，然后求解该博弈以确定获胜方。如果系统获胜，则可以提取出一个实现。

博弈结构是一个元组 $G = (S, S_0, \mathcal{I}, \mathcal{O}, \rho)$，其中 S 是状态集合，$S_0 \subseteq S$ 是初始状态集合， $\mathcal{I}$ 和 $\mathcal{O}$ 是前文定义的命题集合，而 ρ: $S \times 2^{\mathcal{I}} \times 2^{\mathcal{O}} \to S$ 是转移函数。在 G 上的博弈由两位玩家进行：系统和环境。在给定状态 $s \in S$下，环境选择一个输入 $i \subseteq \mathcal{I}$，然后系统选择一个输出$o \subseteq \mathcal{O}$，这些选择共同决定下一个状态 $s' := \rho(s, i, o)$。随后，博弈从 s'继续进行。由此产生的状态无限序列 $\pi = s_0, s_1, s_2, \ldots$ 称为一场对局。形式上，对局是一个序列 $\pi = s_0, s_1, s_2, \ldots \in S^{\omega}$，满足

$s_0 \in S_0$并且对于每个 $t \in \mathbb{N}$, $s_{t+1} = \rho(s_t, i, o)$。系统策略是一个函数$\sigma$： $S^+ \times 2^{\mathcal{I}} \to 2^{\mathcal{O}}$。环境策略是一个函数 π： $S^+ \to 2^{\mathcal{I}}$。给定状态 $s \in S$、系统策略 σ 和环境策略 π，我们用 Outcome(s, π, σ)表示唯一的对局 $s_0, s_1, s_2, \ldots$，使得 $s_0 = s$，并且对于所有$k \in \mathbb{N}$, $s_{k+1} = \rho(s_k, i_k, \sigma((s_0, s_1 \ldots, s_k)\ i_k))$，其中 $i_k = \pi((s_0, s_1 \ldots, s_k))$。

安全博弈是一个元组 (G, 不安全状态集)，其中 UNSAFE $\subseteq S$ 表示不安全状态。如果系统拥有一个策略 σ，使得对于所有环境策略 π、 $s_0 \in S_0$、 $k \in \mathbb{N}$, 都有 Outcome(s_0, π, σ)$_k$ $\notin$ 不属于 UNSAFE，那么系统就在安全博弈中获胜。这样的策略称为系统的获胜策略。直观而言，无论环境做什么，系统都必须避开不安全状态。如果环境能够强制对局到达 UNSAFE 中的状态，即存在环境策略 π 和 $s_0 \in S_0$，使得对于每个系统策略 σ，都存在 $k \in \mathbb{N}$，令 Outcome(s_0, π, σ)$_k \in$ 属于 UNSAFE，那么环境就获胜。

3 使用倒计时定时器博弈进行 SafeLTL$_B$ 综合

SafeLTL$_B$ 综合 我们研究 SafeLTL$_B$ 片段的可实现性和综合问题。我们聚焦于如何高效处理有界时序算子中包含大界限的规约这一挑战，并提出一种旨在实现这一目标的新综合方法。所提出的方法分两个阶段进行。第一阶段，将给定的 SafeLTL$_B$ 公式转换为一种安全博弈，其中的界限以符号方式处理。我们将这类博弈称为倒计时定时器博弈，并将在本节后文介绍。综合算法的第二阶段是求解生成的倒计时定时器博弈，以确定获胜方并回答可实现性问题。我们在第 5 节提出一种采用符号表示和近似的方法，以便在实践中高效求解这类博弈。

倒计时器博弈 直观而言，倒计时器博弈类似于安全博弈，但增加了倒计时器。倒计时器是离散计时器，始终从指定的持续时间开始，并在博弈中的每次转移后递减一。计时器达到零时便会超时，倒计时器博弈的转移关系可以利用这一信息来确定后继状态。倒计时器可以重置为与其关联的持续时间。此外，持续时间相同的倒计时器可以交换其值；我们稍后会在生成定时器游戏时利用这一点，以避免计时器数量出现不必要的爆炸式增长。

定义 1（倒计时器博弈）。倒计时器博弈结构是一个元组 $G_T = (\mathcal{T}, d, L, L_0, \mathcal{I}, \mathcal{O}, \delta)$, 其中 $\mathcal{T}$ 是有限的倒计时器集合， d： $\mathcal{T} \to \mathbb{N}$ 将持续时间与每个计时器关联起来， L 是有限的游戏位置集合， $L_0 \subseteq L$ 是初始位置集合， $\mathcal{I}$、 $\mathcal{O}$ 分别是有限的不可控环境输入命题集合和可控系统命题集合，而 δ： $L \times 2^{\mathcal{I}} \times 2^{\mathcal{O}} \times 2^{\mathcal{T}} \to L \times \mathcal{E}$ 是转移关系。$\mathcal{E} := \mathcal{T} \to (\mathcal{T} \cup \{\text{重置}\})$ 是作用集合，其中对于所有 $e \in \mathcal{E}$:

1. 对所有 $t \in \mathcal{T}$，要么 $e(t) =$ 重置，要么 $e(t) \in \mathcal{T}$并且 $d(e(t)) = d(t)$，并且，

2. 对于 $t_1, t_2 \in \mathcal{T}$，在 $t_1 \neq t_2$ 的条件下，我们有 $e(t_1) \neq e(t_2)$ 或 $e(t_1) = e(t_2) =$ 重置。倒计时器博弈是一个对 $(G_T$, 不安全 $_L)$，其中不安全 $_L \subseteq L$ 是一个不安全位置集合。

效果 $\mathcal{E}$ 捕捉了转移发生时可能出现的计时器重置和重新映射。条件 (1) 指出，每个计时器要么被重置，要么被重新映射到一个持续时间相同的计时器。条件 (2) 要求重新映射是单射，即不能有两个计时器被映射到同一个计时器。如果计时器既未被重置，也未被重新映射到其他计时器，那么它们就映射到自身。

倒计时器博弈的语义是通过显式展开计时器可能的赋值而生成的安全博弈。直观来说，博弈结构中的每个状态都是一对 $s = (l, v)$，由一个位置 $l \in L$ 和一个计时器赋值 v组成。初始时，每个计时器 t 都被设为其对应的持续时间 $d(t)$。转移关系首先递减计时器的值，然后应用 G_T中相应转移的作用 e，以更新计时器的值。相关转移 G_T 由位置 l、输入集和输出集 i 和o，以及递减后数值变为 0 的计时器集合决定。

定义 2（倒计时器博弈语义）。在定义 1 的上下文中，令 $\mathcal{V} := \{v : \mathcal{T} \to \mathbb{N} \mid \forall t \in \mathcal{T}.\ v(t) \leq d(t)\}$ 为所有可能计时器赋值的空间。令 $G = (L \times \mathcal{V}, L_0 \times \{\lambda t.d(t)\}, \mathcal{I}, \mathcal{O}, \rho)$ 为一个博弈结构，其中 $\rho((l, v)\ i, o) := \mathsf{trans}(l, \mathsf{step}(v)\ i, o)$，其中

$$step(v) := \lambda t. \max\{0, v(t) - 1\}$$
$$trans(l, v, i, o) := \begin{cases} \left(l', \lambda t.\ \begin{cases} v(e(t)) & \text{if } e(t) \in \mathcal{T} \\ d(t) & \text{if } e(t) = RESET \end{cases}\right), \\ where\ (l', e) := \delta(l, i, o, \{t \in \mathcal{T} \mid v(t) = 0\}). \end{cases}$$

倒计时器博弈 $(G_T$, 不安全 $_L)$ 的语义是安全博弈 $(G$, 不安全 $_L \times \mathcal{V})$。当且仅当系统（环境）赢得表示其语义的安全博弈时，系统（环境）才能赢得倒计时器博弈。

4 倒计时器博弈构造

现在介绍综合算法的第一阶段，即将$SafeLTL_B$ 公式转换为倒计时器博弈。我们的构造基于展开规则。例如，公式 $[50]a$ 等价于 $a \vee \bigcirc\Diamond[49]a$。如果$a$ 为真，那么整个公式就为真。否则，在下一步中，必须满足 $[49]a$。将 $[50]a$ 解释为安全博弈中的一个状态时，它在 $a = \top$ 上有一条转移到 $\top$，并在 $a = \bot$上转移到 $[49]a$。这一过程可以在 $[49]a$ 上重复，依此类推。一旦到达 $[0]a$，就将其展开为 $a \vee \bigcirc\Diamond\bot$，因此 $a = \bot$ 会到达 $\bot$，即不安全状态。这种构造适用于安全公式，因为可以通过有限前缀判定拒绝。正如我们稍后将展示的，以这种方式生成博弈结构的优点在于，可以利用公式中的信息对其进行剪枝。

然而，这种显式展开会产生一个公式序列，其长度与界呈线性关系，因此相对于公式的描述长度呈指数关系。相反

，我们不采用显式界，而是使用表示多个取值的倒计时定时器。在上面的例子中，我们不会生成所有展开式$[50]a$、.....、$[0]a$，而是使用一个持续时间为 51 的计时器 t，在单个位置 $a \vee \Diamond[t]a$中表示从 50 到 0 的所有展开式。如果 t 超时，$[t]$ 就表示已到达展开式的末尾，并会转换为 $\bot$。因此，在安全博弈中，我们不再将 $[50]a$、.....、$[0]a$、$\top$ 和 $\bot$ 作为状态，而是在倒计时器博弈中只需使用位置 $a \vee \Diamond[t]a$、$\top$ 和 $\bot$。现在，我们将形式化地描述这一构造。

4.1 从 SafeLTL$_B$ 构造倒计时器博弈

生成的倒计时器博弈的位置是$SafeLTL_B$ 公式，此外还带有作为时序算子界限的计时器。我们将这些公式的集合记为$SafeLTL^t_B$。给定计时器集合 $\mathcal{T}$，$SafeLTL^t_B$ 的语法与$SafeLTL_B$ 的语法相同，但在$[n]$、 $\bigcirc[n]$和 $\mathcal{W}[n]$ 中，我们有 $n \in \mathbb{N} \cup \mathcal{T}$。对于 $\varphi \in SafeLTL^t_B$，计时器$(\varphi) \subseteq \mathcal{T}$表示在 φ.中出现的所有计时器。

博弈结构 设 Φ 是一个关于输入命题 $\mathcal{I}$ 和输出命题 $\mathcal{O}$ 的 SafeLTL$_B$ 公式。我们按如下方式构造一个倒计时器游戏结构 $(\mathcal{T}, d, L, L_0, \mathcal{I}, \mathcal{O}, \delta)$。计时器集合

$$\mathcal{T} := \{t_i^d \mid \bigcirc[d], \Diamond[d-1], \text{ or } \mathcal{W}[d-1] \text{ occurs in } \Phi, 0 \le i \le d\}$$

由计时器 t_i^d 组成，其索引为 i，持续时间为 $d(t_i^d) := d$ （对于 $0 \le i \le d$）。计时器的持续时间决定了它可用于 Φ 中的时序算子的界限，而索引则用于区分持续时间相同的多个计时器（这些计时器是在展开过程的不同位置引入的）。

令 $L :=$ PositiveBooleanCombinations (cl(Φ))（即由 cl(Φ) 使用$\wedge, \vee$构成）为位置集，其中 cl 是定义如下的闭包算子:

cl (l) $:= \{l, \top, \bot\}$ $l \in \{$ap $\neg$ap$\}$cl (
$\varphi \; o \; \psi$) $:=$ cl(φ) $\cup$ cl(ψ) $o \in \{\wedge, \vee\}$cl ($[n]\varphi$) $:=$ cl(φ) $\cup \{\bigcirc[t_i^n]\varphi \mid 0 \le i \le n\}$
cl ($\bigcirc[n]\varphi$) $:=$ cl(φ) $\cup \{\Diamond[t_i^{n+1}]\varphi \mid 0 \le i \le n+1\}$cl ($\varphi \, \mathcal{W}[n] \psi$) $:=$ cl(φ) $\cup$cl (
ψ) $\cup \{\varphi \, \mathcal{W}[t_i^{n+1}]\psi \mid 0 \le i \le n+1\}$cl ($\varphi \, \mathcal{W} \, \psi$) $:=$ cl(φ) $\cup$cl(ψ) $\cup \{\varphi \, \mathcal{W} \, \psi\}$.

直观来说，闭包包含展开过程中可能出现的所有时态算子子公式和文字。位置 L 因而表示展开后的公式，直观上对应于系统当前的义务。因此，初始位置对应于义务 Φ。注意，$L \subseteq$ SafeLTL^t_B。我们对生成的公式进行化简，以确保 L 有限。由于根据定义，cl(Φ) 是有限的，因此我们可以确保 $|L| \le 2^{|cl(\Phi)|}$。

在构造初始位置和转移函数时，我们使用两个辅助函数：introExp : SafeLTL$^t_B \to$ SafeLTL^t_B，它执行展开并引入新的计时器；以及 opt : SafeLTL$^t_B \to L$，它执行化简，以确保 L 有限。令 $L_0 :=$ {opt (introExp(Φ))} ，并且

$\delta(\varphi, i, o, T)$: $=$ (（优化 (introExp(ψ)) e）)，其中 (e, ψ): $=$ 挤压(至(T, 树(φ, i, o))).

这里，我们使用了附加函数树：SafeLTL$^t_B \times 2^{\mathcal{I}} \times 2^{\mathcal{O}} \to$ SafeLTL^t_B，用于执行输入和输出选择；至：$2^{\mathcal{T}} \times$SafeLTL$^t_B \to$ SafeLTL^t_B，用于处理超时；以及挤压：SafeLTL$^t_B \to \mathcal{E} \times$SafeLTL$^t_B$，用于确定计时器的重新映射和重置。下文将详细介绍这些函数。

备注：注意，对于$[b]$，我们使用持续时间为 b 的计时器；而对于$[b]$ 和$\mathcal{W}[b]$，我们使用持续时间为 $b+1$ 的计时器。这样做的原因是，对于后者，我们将最后一步视为计时的一部分，因为这会简化博弈结构。

在描述这些函数之前，我们先通过一个简单的例子来说明它们。

例1。设 $\mathcal{I} = \{r\}$，$\mathcal{O} = \{g\}$，并考虑以下 $SafeLTL_B$ 公式 $\Phi = (\square[100]\neg g) \wedge \square\bigcirc[10](r \to \Diamond[100]g)$. Φ 表示系统在前 100 步内不应授予请求；如果第 10 步出现请求，则应在接下来的 100 步内给予授权。我们将展示如何为 Φ构造初始位置以及倒计时器博弈中的一些转移。

初始状态 $\varphi_0 =$ 优化 (introExp(Φ))

初始状态由 Φ 通过展开公式并引入所需的计时器来计算。这由 introExp 函数完成。子公式$[100]\neg$g 展开为 $\neg$g $\wedge \square[t_0^{101}]\neg$g，反映了算子$[100]$的语义。这会引入一个持续时间为 101、索引为 0 的计时器 t_0^{101}。子公式 $[10](r \to \Diamond[100]$g) 展开为$\bigcirc[t_0^{10}](r \to \Diamond[100]$g)，由此为 $[10]$引入计时器 t_0^{10}。计时器的持续时间 101 和 10 分别对应于$[100]$ 和$[10]$,中的界，而索引 0 是相应持续时间下当前未使用计时器的最小索引。此步骤不会为$[100]$ 引入计时器，因为它受一个算子约束。因此，初始状态就是展开后的公式 $\varphi_0 = \neg$g $\wedge(\square[t_0^{101}]\neg$g) $\wedge \bigcirc[t_0^{10}](r \to \Diamond[100]$g)。

确定转移 $\delta(\varphi_0, \emptyset, \{g\}, \emptyset) = (\varphi_1, e_1)$

我们应用 tree($\varphi_0, \emptyset, \{g\}$)，它计算当前步骤中输入 $\emptyset$ 和输出{g} 对公式产生的作用，因此将 φ_0中的 g 替换为 $\top$。这会得到 tree($\varphi_0, \emptyset, \{g\}$) $= \bot$，这意味着此转移会到达位置 $\bot$。确定转移 $\delta(\varphi_0, \emptyset, \emptyset, \{t_0^{10}\}) = (\varphi_2, e_2)$

再次，我们首先计算 tree($\varphi_0, \emptyset, \emptyset$) $= (\square[t_0^{101}]\neg g) \wedge \bigcirc[t_0^{10}](r \to \Diamond[100]g)$，此时它将 $\bot$ 替换为 g。然后，对结果应用处理超时的 to 函数，此处为 $\{t_0^{10}\}$，这意味着计时器 t_0^{10}在当前步超时。因此，子公式 $[t_0^{10}](r \to \Diamond[100]g)$ 被替换为 $r \to \Diamond[100]g$，这表示公式 $r \to \Diamond[100]g$ 成为下一步义务的一部分，因为计时器 t_0^{10}已超时。由此，我们得到 to($\{t_0^{10}\}$, $(\square[t_0^{101}]\neg g) \wedge \bigcirc[t_0^{10}](r \to \Diamond[100]g)) = (\square[t_0^{101}]\neg g) \wedge (r \to \Diamond[100]g)$。之后，我们应用挤压函数 squeeze，以处理超时后可能变得未使用的计时器。这体现在作用 e_2 上，它会重置当前公式中未出现的所有计时器。因此，在 e_2 中，刚刚超时的计时器 t_0^{10}被映射为 RESET，而仍然存在的计时器 t_0^{101}则映射为其自身。最后一步是应用 introExp 函数，对当前公式进行展开，并引入可能需要的任何新计时器。子公式 $[t_0^{101}]\neg$g 展开为 $\neg$g $\wedge \square[t_0^{101}]\neg$g。子公式 $r \to \Diamond[100]$g 展开为 $r \to (g \vee \Diamond[t_1^{101}]$g)，并为 $[100]$引入计时器 t_1^{101}。请注意，由于公式中已经包含持续时间为 101 的计时器 t_0^{101}，新引入的

计时器 t_1^{101}的索引为 1。函数 to 和 $squeeze$ 确保相同持续时间的计时器索引顺序，能够体现这些计时器超时的先后顺序。计算 $introExp((\,[t_0^{101}]\neg g)\ \wedge(r\ \rightarrow [100]g))$ 后，我们得到 $\varphi_2 = \neg g\ \wedge(\ \ [t_0^{101}]\neg g)\ \wedge(r\ \rightarrow (g\ \vee\Diamond[t_1^{101}]g))$.

构造 我们通过从 L_0 探索 L 的可达部分，构造位置、计时器和转移集合。我们描述了几种剪枝机制，以保持可达位置集合较小。

构造不变量。为确保正确性并提高博弈生成效率，我们为每个可达位置维持以下不变量：

1. 对于每个可达位置 φ，(1.a) 所有未受“下一步”算子约束的文字和有界算子都出现在布尔顶层，且 (1.b) 所有顶层有界算子都实例化了一个计时器。2. 对于每个持续时间 d，计时器的值按索引排序，即 $t_0^d < t_1^d < \ldots t_j^d = \ldots t_d^d = d$。对于值不为 d 的计时器，排序是严格的。3. 在位置 φ 中，对于任意 d 和 $i > 0$，若 $t_i^d \in$ Timers(φ)，则 $t_{i-1}^d \in$ Timers(φ)。

不变式 (1) 对于正确性是必需的，同时还要确保当前步骤中所有相关的文字都被考虑，并且所有相关的有界算子都由计时器跟踪。不变式 (2) 确保我们所需的计时器数量不会超过可用的 d 个。这是因为计时器在运行时严格排序，而一旦我们引入 t_{d+1}^d，t_0^d 就已经超时。此外，对计时器进行排序还会减少超时的可能组合。不变式 (3) 防止按照上述顺序在已使用的计时器之间存在未使用的计时器，从而减少等价位置的可能组合。

tree 函数：输入和输出的选择。函数 tree(φ, i, o) 计算当前步骤中输入 i 和输出 o 对公式的作用。根据不变式 (1)，只需考虑布尔顶层上的文字，即不是时序算子子公式的文字。按照 i 和 o 在 φ 中赋值文字时，我们利用正在生成博弈这一事实，剪枝并选择一些可以立即确定的“显然选择”。这种剪枝是我们方法的重要组成部分，因为在实践中，它可以剪去相当一部分可能的位置。tree 函数递归地应用一组规则。下面我们按照每个递归步骤中应用这些规则的顺序进行描述。图 1 给出了形式化描述。

1. 若顶层析取项 c 是输出文字，则系统通过使公式 $\top$成立而获胜。系统的另一种选择可以安全地剪枝。2. 若顶层合取项 u 是输入文字，则环境通过使公式 $\bot$成立而获胜。另一种选择可以安全地剪枝。3. 如果某个输出命题只以正极性或只以负极性出现，那么系统只需选择相应极性的文字，因为另一种选择生成的公式会被蕴含。4. 如果某个输入命题只以正极性或只以负极性出现，那么只需考虑环境选择取反文字的情况，因为这种情况的实现难度严格更高（即一个公式蕴含另一个公式），而针对这种情况的每种策略也适用于另一种情况。

$$tree(c \vee \psi, i, o) := [\![c \in o]\!] \tag{1}$$
$$tree(u \wedge \psi, i, o) := \bot \tag{2}$$
$$tree(\psi, i, o) := \begin{cases} tree(\psi[c/\top]_T) & \text{if } c \in o \\ \bot & \text{if } c \notin o \end{cases} \quad c \in ActL(\psi), \neg c \notin ActL(\psi) \tag{3}$$
$$tree(\psi, i, o) := tree(\psi[u/\bot]_T) \quad u \in ActL(\psi), \neg u \notin ActL(\psi) \tag{4}$$
$$tree(\psi, i, o) := \psi[u/[\![u \in i]\!]]_T \quad u, \neg u \in ActL(\psi) \tag{5}$$
$$tree(\psi, i, o) := \psi[c/[\![c \in o]\!]]_T \quad c, \neg c \in ActL(\psi) \tag{6}$$

图 1：设 $u \in \mathcal{I}$ 和 $c \in \mathcal{O}$。为简化说明，我们省略交换和结合的情形以及负文字。ActL(ψ) 表示出现在 ψ的布尔顶层的文字集合。公式 $\psi[\mathsf{ap}/v]_T$由 ψ 得到：将布尔顶层所有出现的 ap 替换为 $v \in \{\top, \bot\}$，但仅限于该层。每次替换后，我们通过常量折叠来简化公式。$[\![x \in X]\!]$ 在 $x \in X$ 时为 $\top$，在 $x \notin X$时为 $\bot$。

5. 如果无法做出“提前决策”或“最坏情况决策”，我们就采用环境选择，因为在博弈中环境先行动。
6. 如果没有剩余的环境选择，我们就生成系统的分支。

to 函数：处理超时。不变式 (2) 的一个推论是，由于计时器已排序，只有索引为 0 的计时器（即形式为 t_0^d 的计时器）会超时。此外，未出现在公式中的计时器不应超时（这由挤压保证），我们稍后将对此进行说明。请注意，这不适用于持续时间为 1 的计时器，因为它们会立即超时。我们将不可能发生的超时导向$\top$，因为这些超时不会发生。因此，to(T, φ) := $\top$ 当且仅当存在某个 $t_i^d \in T$ 使得$i \neq 0$，或者 $d > 1$ 且 $t_i^d \notin$ Timers(φ)。否则，to(T, φ) 的定义是：对 φ 的所有子公式应用以下变换，并使计时器$t \in T$超时：我们变换 $[t]\psi \rightsquigarrow \bot$、 $\bigcirc[t]\psi \rightsquigarrow \psi$和 $\phi \mathcal{W}[t]\psi \rightsquigarrow \top$。应用 to 后，我们执行常量折叠，因为公式的某些部分可能会变得无关紧要。

squeeze 函数：确定计时器的重新映射和重置。在应用树函数时，某些计时器可能会变得未使用。因此，我们必须确保不变式 (3) 成立，并如上一段所述，重置所有未出现在公式中的计时器。我们将 squeeze(φ) := (e, ψ) 定义如下：对于每个持续时间 d，令 $t_{i_j}^d \in$ Timers(φ) 中索引为 $i_0 < i_1 < i_2 < \ldots$的计时器为剩余计时器，并按索引 i_j排序。然后设 $e(t_j^d) := t_{i_j}^d$，若 i_j 存在；否则设 e (t_j^d) := 重置。通过将计时器 $t_{i_j}^d$ 替换为 t_j^d，即可得到 ψ。

introExp 函数：展开与计时器引入。introExp 函数执行公式展开，并在必要时引入新的计时器。该展开可保证之后不变式 (1) 成立。引入新计时器时，还必须维持不变式 (2) 和不变式 (3)。具体做法是：对于每个关联持续时间为 d 的界 b，将计时器赋予下一个尚未使用的索引，即 $t_j^d \notin$ Timers(φ)，其中 $t_0^d, \ldots, t_{j-1}^0 \in$ Timers(φ)。令 $I(d) := \max\{i \mid t_i^d \in \text{Timers}(\varphi)\} + 1$ 为下一个尚未使用的索引。此外，由于根据不变式 (2)，不存在满足 $i > d$ 的计时器 t_i^d，因此生成这些计时器的展开式将被重定向

至 $\top$。因此，我们定义 introExp(φ) := rd(iE $_I(\varphi)$)，其中 rd (φ) := $\top$ 若存在某个 $i > d$ 使得 $t_i^d \in$ Timers(φ)，则为 φ，否则为 $= \varphi$。执行展开的函数 iE$_I$ 定义如下：

iE $_I(l) := l$ iE $_I(\varphi\ o\ \psi) :=$ iE $_I(\varphi)\ o$ iE $_I(\psi)$ iE $_I([n]\varphi) :=$ iE $_I(\varphi) \vee \Diamond[t_I^{n+1}(n+1)]\varphi$ iE$_I([t]\varphi) :=$ iE $_I(\varphi) \vee \Diamond[t]\varphi$ iE $_I(\ \ [n]\varphi) := \bigcirc[t_{I(n)}^{n}\Diamond]\varphi$ iE $_I(\ \ [t]\varphi) := \bigcirc[t]\varphi$ iE $_I(\varphi \mathcal{W} \textcircled{o}[n]\psi) :=$ iE $_I(\psi) \vee$ iE $_I(\varphi) \wedge$ iE $_I(\varphi \mathcal{W}[t]\psi) :=$ iE$\bigcirc_I(\psi) \vee$ iE $_I(\varphi)$ ($\varphi \mathcal{W}[t_I^{n+1}{}_{n+1}()]\psi$) $\wedge(\varphi \mathcal{W}[t]\psi)$ iE $_I(\varphi \mathcal{W} \psi) :=$ iE $_I(\psi) \vee$ iE $_I(\varphi) \wedge(\varphi \mathcal{W} \psi)$,

其中 $l \in \{$ap $\neg$ap$\}$, $o \in \{\wedge, \vee\}$, $n \in \mathbb{N}$ 以及 $t \in \mathcal{T}$。

函数 opt：公式化简。函数 opt 通过化简公式来避免引入无穷多个逻辑等价的公式，从而确保所构造的位置集合 L 是有限的。由于我们必须维持不变式，这种化简并不保证模等价意义下的唯一性。不过，它确保了 L 的有限性并进行了优化。

不安全集合的定义与正确性 为了完成倒计时器博弈的构造，我们将不安全位置集合定义为 UNSAFE$_L$ ={$\bot$}。下文正确性定理的证明见附录 B.1。

定理 1. 设 $\Phi \in$ SafeLTL$_B$，且 G 是如上所述由 Φ 构造的倒计时器游戏结构。那么，存在一个实现 $\mathcal{L}(\Phi)$ 的系统，当且仅当该系统在倒计时器博弈 (G, UNSAFE$_L$) 中获胜。

我们通过加入若干扩展来改进该构造的效率并扩大其适用范围。例如，我们将显式展开与基于计时器的隐式展开相结合，从而能够直接处理诸如单个 $\bigcirc\psi$ 这样的算子。我们还使用近似来处理形如 ψ 的简单假设，其中 $\mathcal{W}$是完全有界的，即不含 $\bigcirc$。详细信息见附录 A。 □

5 求解倒计时定时器博弈

现在我们描述综合算法的第二阶段，即求解由 SafeLTL$_B$ 规约生成的倒计时器博弈。在倒计时器博弈中，计时器的持续时间（对应于规约中时序算子的界）采用二进制编码。因此， $\mathcal{V}$ 计时器赋值的集合以及第 3 节中定义的安全博弈都会随着倒计时器博弈规模的增大而呈指数增长。由于我们的目标是高效地求解持续时间较大的倒计时器博弈，因此不希望显式构造并求解语义安全博弈。然而需要指出的是，在最坏情况下无法避免这种爆炸式增长。下一定理对此进行了说明，其证明见附录 B.2。

定理 2. 求解倒计时定时器博弈是 *EXPTIME*-完备的。

这意味着，要高效求解倒计时定时器博弈，就需要采用一种以符号方式处理计时器赋值集合的方法，从而尽可能避免显式枚举。我们提出了一种用于求解倒计时定时器博弈的符号算法，该算法还采用了逐步细化的近似。此方法适用于计时器赋值集合的通用符号表示。我们展示了该方法的一种具体实现，其表示由计时器值区间和计时器上的偏序组成。

符号博弈求解　求解安全博弈的标准方法是计算环境能够强制到达不安全状态的状态集合，并检查该集合是否与初始状态集合相交。若相交，则环境赢得博弈；否则，系统获胜。

对于博弈 (G, 不安全)，其中 $G = (S, S_0, \mathcal{I}, \mathcal{O}, \rho)$，环境能够强制到达不安全状态的状态集合称为环境吸引子，定义为 AttrE_G(不安全) $= \{s \in S \mid \exists \pi$: 环境策略.$\forall \sigma$: 系统策略.$\exists k \in \mathbb{N}$. 结果$(s, \pi, \sigma)_k \in$ 不安全$\}$。当且仅当 AttrE_G(不安全) $\cap S_0 \neq \emptyset$ 时，环境赢得安全博弈。

我们通过计算环境玩家对不安全位置的吸引集的符号表示，来求解倒计时器博弈。假设存在计时器赋值空间 $2^{\mathcal{V}}$ 的符号表示 Rep。对于每个$R \in$ Rep，我们用 $[\![R]\!] \subseteq \mathcal{V}$ 表示由 R 表示的 $\mathcal{V}$ 子集。我们使用函数来表示语义安全博弈的状态空间子集 $L \times \mathcal{V}$：函数从$L \to$映射到 Rep，其中 $U \in (L \to \text{Rep})$ 表示 $\{(l, v) \mid v \in [\![U(l)]\!]\}$。

环境的符号可强制前驱 CPreE^{symb} : $(L \to \text{Rep}) \to (L \to \text{Rep})$ 定义如下。对于 $U \in (L \to \text{Rep})$，我们令

$$CPreE_{symb}(U) := \lambda l. \bigcup_{i \subseteq \mathcal{I}} \bigcap_{o \subseteq \mathcal{O}} \bigcup_{T \subseteq \mathcal{T}} symTrans(\delta(l, i, o, T), T, U), \text{ where}$$

$$symTrans((l', e), T, U) := inc(effTO(T, remap(e, effReset(e, U(l')))))$$

是将转移 $\delta(l, i, o, T)$ 反向作用于目标集$[\![U(l')]\!]$的符号化操作。symTrans 所需的操作按从后到前的顺序如下。

– inc：Rep $\to$ Rep 对计时器执行反向递增，形式化地，$[\![\text{inc}(R)]\!] = \{\lambda t.\ v(t) + 1 \in \mathcal{V} \mid v \in [\![R]\!]\}$。– effTO：$2^{\mathcal{T}} \times$ Rep $\to$ Rep 表示超时的作用：$[\![\text{effTO}(T, R)]\!] = \{v \in [\![R]\!] \mid \forall t \in \mathcal{T}.(t \in T \to v(t) = 0) \wedge (t \notin T \to v(t) \in [1, d(t)])\}$。– remap：$\mathcal{E} \times$ Rep $\to$ Rep 表示重新映射的作用：$[\![\text{remap}(e, R)]\!] = \{v \in \mathcal{V} \mid \exists v' \in [\![R]\!].\forall t \in \mathcal{T}$使得 $e^{-1}(t)$ 已定义. $v(t) = v'(e^{-1}(t))\}$。– effReset：$\mathcal{E} \times$ Rep $\to$ Rep 表示计时器重置的作用：$[\![\text{effReset (e, R)}]\!] = \{v \in [\![R]\!] \mid \forall t \in \mathcal{T}.e(t) =$ 重置 $\to v(t) = d(t)\}$。注意，$e^{-1}(t)$（由作用 e 映射到 t的计时器）是唯一的，因为该作用对于不同于重置的值是单射的，因此在有定义时可以求逆。

我们还要求能够执行集合运算 $\cup$、$\cap$，并对 Rep 中的元素进行相等性检查，以便完成计算。

我们采用符号可强制前驱算子 CPreE^{symb}，按照以下方式计算环境吸引子 AttrE^{symb} 的符号表示：

如下。我们令 $\mathrm{AttrE}^0_{symb} := (\lambda l.\text{若 } l \in \text{不安全 } _L \text{则 } \mathcal{V} \text{否则 } \emptyset)$，然后对于$n \in \mathbb{N}$，令 $\mathrm{AttrE}^{n+1}_{symb} := \mathrm{AttrE}^{\ n}_{symb} \cup \mathrm{CPreE}^{\ symb}(\mathrm{AttrE}^{\ n}_{symb})$。

命题 1. 如果 $(G_T, \text{不安全}_L)$ 是一个带有 $G_T = (\mathcal{T}, d, L, L_0, \mathcal{I}, \mathcal{O}, \delta)$ 的倒计时器博弈，且安全博弈 $(G, \text{不安全}_L \times \mathcal{V})$ 具有 $G = (L \times \mathcal{V}, L_0 \times \{\lambda t.d(t)\}, \mathcal{I}, \mathcal{O}, \rho)$ 并且是其语义，那么对于上述计算出的符号吸引集，对每个 $l \in L$ 都有 $[\![\mathrm{AttrE}^{\ symb}(l)]\!] = \{v \in \mathcal{V} \mid (l, v) \in \mathrm{AttrE}_G\}$。

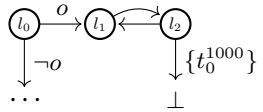
计时器赋值的近似 由于上述描述的符号化状态空间仍可能导致探索大量集合，我们对所探索状态的吸引集进行过近似和欠近似。

我们使用阈值 $k \in \mathbb{N}$ 来控制抽象的精度。直观地说，在对 $t \in \mathcal{T}$ 进行近似时，我们希望精确处理“边界”上的计时器值，即 $[0, k]$ 和 $[d(t) - k, d(t)]$, 中的计时器值，因为这些值与超时和重置有关。我们的过近似 $\mathrm{over} : \mathrm{Rep} \to \mathrm{Rep}$ 和欠近似 $\mathrm{under} : \mathrm{Rep} \to \mathrm{Rep}$ 将中间值 $[k, d(t) - k]$ 视为单个值块。对于 (R) 的过近似，如果 R 中有一个值位于 $[k, d(t) - k]$ 内，则加入所有中间值；而对于 (R) 的欠近似，如果 R 中有一个值不在其中，则移除所有中间值。形式化地说：

$$
\begin{aligned}
&approx_k(t, I) := (I \cap [k, d(t) - k] \neq \emptyset) \wedge ([k, d(t) - k] \not\subseteq I) \\
&[\![over(R)]\!] \quad := \left\{ \lambda t. \begin{cases} v(t) \cup [k, d(t) - k] & \text{if } approx_k(t, v(t)) \\ v(t) & \text{otherwise} \end{cases} \;\middle|\; v \in [\![R]\!] \right\} \\
&[\![under(R)]\!] \quad := \left\{ \lambda t. \begin{cases} v(t) \setminus [k, d(t) - k] & \text{if } approx_k(t, v(t)) \\ v(t) & \text{otherwise} \end{cases} \;\middle|\; v \in [\![R]\!] \right\}
\end{aligned}
$$

现在按如下方式计算吸引集：我们从 $k := 1$ 开始。对于当前的 k，我们在计算过程中的每个符号状态上，分别使用欠近似和过近似计算一次环境吸引子。如果环境在欠近似中获胜，那么它就在具体游戏中获胜。如果系统在过近似中获胜，那么它就在具体游戏中获胜。如果两者都不成立，我们就令 $k := 2 \cdot k$，然后重复此过程。该过程总会终止，因为对于 $k > d(t)/2$，这些近似会变得精确，因此必有一方获胜。

例 2。考虑一个倒计时器博弈，其中一些转移如图 2a 所示。在图示的转移中，只有从 l_2 到$\bot$ 的转移具有非空的超时集合 $\{t_0^{1000}\}$。由于计时器 t_0^{1000}的持续时间为 1000，



(a) 倒计时器博弈，$UNSAFE_L = \{\bot\}$。

	0	1	2	3	4	...	7
l_1	$\emptyset$	$\emptyset$	$\{1\}$	$\{1\}$	$\{1\}, [3, 997]$	...	$\{1\}, [3, 997], \{999\}$
l_2	$\emptyset$	$\{0\}$	$\{0\}$	$\{0\}, \{2\}$	$\{0\}, \{2\}$	...	$\{0\}, [2, 998], \{1000\}$

(b) 近似吸引子计算过程中的集合。

图 2：展示计时器赋值近似作用的示例。

要精确计算位置 l_1 和 l_2 的 AttrE^{symb} ，需要迭代 1000 次。另一方面，采用阈值为 $k = 3$的过近似，如图 2b 所示，只需迭代 7 次即可达到不动点。这在图 2a 所示博弈这样的情形中很有帮助：位置l_0 处转移的选择由系统控制（通过输出 o）。在此，过近似使求解算法能够迅速判定，选择转移至 l_1 会导致失败，而系统可以通过另一条转移获胜。

使用盒子的符号表示 作为符号域，我们选择了计时器上的偏序增强的区间表示 Rep : $2^{PartialOrder(\mathcal{T} \times 2^{Rec}} = \{ i \in \mathcal{T} \to \mathbb{N} \times \mathbb{N} \mid \forall t \in \mathcal{T} \ a \ b \ = i \ t \ .0 \leq)$其中 Rec : ( ) ,(,) ( )$a \leq b \leq d(t)\}$ 是超立方体形式的区间。直观来说，我们有一组偏序，对于每个偏序，又有一组超立方体。形式化地：

$$[\![R]\!] := \bigcup_{(p,C)\in R} \left(\{v \in \mathcal{V} \mid \forall (t_1 \sim t_2) \in p : v(t_1) \sim v(t_2)\} \cap \bigcup_{r\in C} \lambda t.[r(t)_1, r(t)_2] \right)$$

其中 $r(t)_i$ 是 $r(t)$ 的第 i 个投影。接下来需要定义必要的操作：inc、effReset、effTO 和 remap。这些操作大多可根据定义直接实现，因为可以逐个修改和检查所有区间，或仅重新排列计时器。此外，effReset 利用偏序推导与被重置计时器相关的计时器的界。effTO 会细化偏序，因为在超时 T 时， T 中的所有计时器都小于 $\mathcal{T}\backslash T$。此外，也可以逐点对区间进行近似，因为近似后的区间仍然是区间。我们采用语法相等作为相等性判定，也就是说，必须避免冗余状态，以防止状态空间爆炸。通过应用德摩根定律，再计算区间和偏序的并集，可得到 $\cap$ 。 $\cup$ 可通过标准的集合并计算得到。我们合并相同的偏序和相邻的超立方体，以避免冗余状态。虽然这无法消除所有冗余，但在实践中效果足够好。

我们选择这个域，是因为它简单，同时由于采用了偏序，也非常适合我们正在解决的这类问题。我们的求解算法具有通用性，也能适用于其他更复杂的域。

6 评估

我们实现了[1] 并对我们的方法进行了评估。我们将原型实现与 [9] 中介绍的 ebr-ltl-synth 进行比较；该工具针对线性时序逻辑$_{\mathsf{EBR}}$进行综合。我们还与最先进的线性时序逻辑综合工具 strix 21.0 版0 [19, 22]进行了比较。下文将介绍我们使用的基准测试、实验及结果。所有实验均在配备 16GB 内存的 Intel Core i7-1165G7 处理器上运行，并且仅使用一个核心。所有时间均为墙钟时间。基准测试的详细说明见附录D。

[1] 可通过以下地址获取： https://github.com/phheim/lisynt

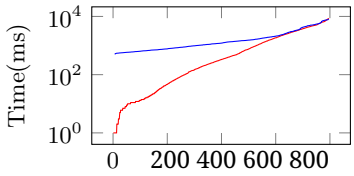


图3：基准测试 [9] 的执行时间（毫秒）。

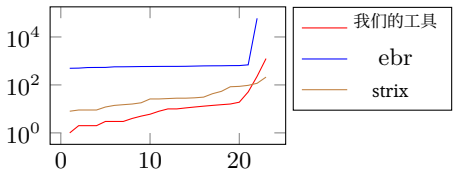


图4：$\mathrm{LTL}_{\mathsf{EBR}}$ SYNTCOMP 基准测试上的执行时间（毫秒）。

有界响应基准 在第一组实验中，我们使用来自$_{\mathsf{EBR}}$ 的线性时序逻辑 [9], 公式，以及 23 个属于线性时序逻辑$_{\mathsf{EBR}}$ 的 SYNTCOMP 2021 基准测试[2]对这些工具进行评估； [9]中也使用了这些基准进行类似比较。图 3 和图 4 分别展示了超时时间设为一分钟时的运行时间。不幸的是，对于 [9] 中的大约一半基准测试，strix 因输入公式过长而无法接受，因为输入时必须显式展开有界算子。因此，我们在这项比较中未纳入 strix。图 3 显示，对于 [9] 中的基准测试，忽略不同的启动时间后，我们的实现与 ebr-ltl-synth 的运行时间大致相同。图 4 显示，在选定的 SYNTCOMP 基准测试中，三种工具的表现相当。

这些实验评估了我们的实现对相关基准测试的表现，其中有些基准测试的设计初衷与我们方法所针对的问题并不完全一致。结果表明，我们的实现与现有工具不相上下。

实时基准的改编 在第二组实验中，我们从 [13] 中选取了 MTL 综合问题，并将其改编为 SafeLTL$_B$ 公式。基准测试包括一条传送带（conv-belt）、一个机器人摄像头（robo-cam），以及多铁路道口控制器（rail）的若干参数化实例。我们对实时界限进行了离散化。这些基准测试最多使用 19 个命题和 16 个有界算子，界限介于 60 和 4000 之间。详细结果见表1。由于我们必须使用假设（无法通过 $\mathrm{LTL}_{\mathsf{EBR}}$ 片段中的规约表示）来建模定时环境，因此 ebr-ltl-synth 不适用于这些基准测试。

这些实验表明，通过恰当的离散化，SafeLTL$_B$ 能够表达实时领域中的有趣需求。我们没有直接与 [13], 中的工具进行比较，因为其底层建模形式体系不同，因此我们对基准测试做了调整。不过，对比 [13] 中的结果可以看出，我们的工具表现良好（在某些情况下甚至更优）。此外，在这些基准测试中，我们的工具明显优于 strix。

办公室机器人基准 我们最后一组实验考察了自行创建的基准测试。这些基准测试由若干规约组成，用于描述机器人在一栋有四个房间的办公楼中执行的任务。基准测试的参数是需要服务的房间数量。它们最多使用 11 个命题和 14 个有界时序算子，界在 10 到 21600 之间。详细

[2] https://github.com/SYNTCOMP/benchmarks

Name	$\|L\|$	$\|\mathcal{T}\|$	τ_{Gen}	k	胜利	τ_Σ	$\tau_{\texttt{strix}}$
$Clean$(1)	8	2	0.01	1	S	**0.01**	3.56
$Clean$(2)	16	3	0.02	1	S	**0.03**	7.99
$Clean$(3)	41	4	0.06	8	S	**0.33**	21.4
$Clean$(4)	123	5	0.22	8	S	**1.45**	97.3
$Clean_C$(1)	10	4	0.03	1	S	**0.05**	189
$Clean_C$(2)	22	5	0.08	16	S	**617**	TO
$Clean_C$(3)	61	6	0.32	-	-	TO	TO
$Clean_C$(4)	205	7	1.30	-	-	TO	TO
$Coffee$(1)	14	4	0.03	1	S	**0.04**	TO
$Coffee$(2)	44	5	0.12	2	S	**0.33**	TO
$Coffee$(3)	175	6	0.55	2	S	**3.53**	TO
$Coffee$(4)	418	7	1.34	2	S	**10.2**	TO
conv-belt	9	3	0.01	1	S	**0.02**	F
robo-cam	22	5	0.04	1	S	**0.19**	F
rail(2,2)	647	6	2.60	1	S	**3.93**	TO
rail(2,4)	647	6	2.58	1	S	**4.05**	TO
rail(2,8)	647	6	2.62	1	S	**3.97**	TO
rail(4,4)	647	7	2.67	1	S	**4.10**	TO

Name	$\|L\|$	$\|\mathcal{T}\|$	τ_{Gen}	k	胜利	τ_Σ	$\tau_{\texttt{strix}}$
$Clean_H$(1)	3	2	0.02	512	E	**0.07**	1.61
$Clean_H$(2)	3	2	0.02	512	E	**0.07**	2.63
$Clean_H$(3)	3	2	0.02	512	E	**0.07**	4.99
$Clean_H$(4)	3	2	0.02	512	E	**0.07**	5.64
$Clean_N$(1)	23	4	0.07	1	S	**0.12**	TO
$Clean_N$(2)	32	4	0.10	1	S	**0.27**	TO
$Clean_N$(3)	48	4	0.15	8	S	**7.47**	TO
$Clean_N$(4)	75	4	0.26	8	S	**13.7**	TO
$Coffee_C$(1)	46	6	0.16	1	S	**0.88**	F
$Coffee_C$(2)	151	7	0.59	1	S	**5.51**	F
$Coffee_C$(3)	613	8	2.73	1	S	**62.9**	F
$Coffee_C$(4)	1634	9	6.82	1	S	**191**	F
rail(4,8)	647	7	2.53	1	S	**3.96**	TO
rail(8,8)	647	7	2.60	1	S	**4.03**	TO
rail(1,1,1)	3111	7	27.8	-	-	TO	TO
rail(2,1,1)	9179	9	89.1	1	S	**220**	TO
rail(2,2,2)	9179	9	93.7	1	S	**225**	TO

表1：办公室机器人和改编后的实时基准测试结果。$|L|$ 和$|\mathcal{T}|$ 分别是生成的倒计时器博弈中的位置数和计时器数。τ_{Gen} 是博弈生成的运行时间，单位为秒。k 是求解终止时所采用的近似阈值。胜利方显示系统（S）或环境（E）是否获胜。τ_Σ 是包含博弈生成和求解在内的总运行时间，其中 TO 表示 15 分钟后超时。$\tau_{\texttt{strix}}$ 是 strix 的运行时间。对于某些基准测试，strix 会因输入过长而拒绝处理（F）；这是因为使用 strix 时，有界算子会被展开。

结果见表1。由于使用了假设（4 个基准测试），ebr-ltl-synth 不适用；另有 25 个基准测试超时。

结果表明，SafeLTL$_B$ 能够表达有意义的综合任务，而且我们的方法适用于求解这些任务。此外，结果也表明，我们的方法确实达到了预期目的：对于时序算子要求较大界限的规约，我们的方法明显优于最先进的工具。

总体分析：表1显示，与求解相比，倒计时游戏生成的效率非常高。我们预计可以通过更先进的符号技术改进求解，因此也认为，基于倒计时游戏的方法能够适用于更复杂的性质。在大多数情况下，求解器在近似阈值较低时便终止了，这表明近似很有用。根据我们的经验，若不采用近似，使用当前技术求解界限较大的基准测试将变得不可行。

7 结论

我们提出了一种新的合成方法，用于处理以安全线性时序逻辑扩展形式表示、且包含有界时序算子的规约。我们方法的一个显著特点是，专门针对高效求解包含有界时序算子的规约的综合问题，尤其是那些界限较大的规约。评估结果表明，我们的技术在一系列具有此类时序要求的基准测试中表现出色。成功的关键在于，我们提出了一种新颖的方法，将规约转换为界限以符号方式表示的安全博弈，

其效率得益于有效的剪枝技术。评估结果表明，我们求解所生成博弈的方法切实可行。不过，通过采用性能更优的符号表示和抽象技术，该方法仍有进一步改进的空间。

数据可用性声明

本研究期间生成和/或分析的数据集可在 Zenodo 存储库中获取：https://doi.org/10.5281/zenodo.7505914。

参考文献

1. Rajeev Alur、Kousha Etessami、Salvatore La Torre 和 Doron A. Peled。“模型测量”的参数时序逻辑。 *ACM Trans. Comput. Log.*, 2(3):388–407, 2001。2. Rajeev Alur、Tom´as Feder 和 Thomas A. Henzinger。放宽精确时性的益处。 *J. ACM*, 43(1):116–146, 1996。3. Gerd Behrmann、Agn`es Cougnard、Alexandre David、Emmanuel Fleury、Kim Guldstrand Larsen 和 Didier Lime。Uppaal-tiga：博弈对局的时刻！载于 Werner Damm 和 Holger Hermanns 编， *Computer Aided Verification, 19th Inter-national Conference, CAV 2007, Berlin, Germany, July 3-7*, 2007, *Proceedings*, *Lecture Notes in Computer Science*第4590卷，第121–125页。施普林格，2007。4. Patricia Bouyer、Laura Bozzelli 和 Fabrice Chevalier。针对 MTL 规约的控制器综合。载于 Christel Baier 和 Holger Hermanns 编， *CONCUR2006 - Concurrency Theory, 17th International Conference, CONCUR 2006, Bonn, Germany August 27-30 2006 Proceedings* 4137 *Lecture Notes in Com-* , , , , 第 卷, *puter Science* 450 464 2006 页码 – 。施普林格，。5. Thomas Brihaye、Morgane Esti´evenart、Gilles Geeraerts、Hsi-Ming Ho、Benjamin Monmege 和 Nathalie Sznajder。实时综合难度很高！载于 Martin Fr¨anzle 和 Nicolas Markey 编， *Formal Modeling and Analysis of Timed Systems - 14th International Conference, FORMATS 2016, Quebec, QC, Canada, August 24-*26, 2016, *Proceedings*, *Lecture Notes in Computer Science*第9884卷，第105–120页。施普林格，2016。6. Peter E. Bulychev、Alexandre David、Kim G. Larsen 和 Guangyuan Li。针对 MTL 0∞ *Acta Informatica* 51 3 4 165 192, 片段的高效控制器综合，。, (-): – 。7. Franck Cassez。面向部分可观测定时博弈的高效在线算法。载于 Jean-Fran¸cois Raskin 和 P. S. Thiagarajan 编， *Formal Modeling and Analysis of Timed Systems, 5th International Conference, FORMATS 2007, Salzburg, Austria October 3-5 2007 Proceedings* 4763 *Lecture Notes in Computer* , , , , 第 卷, *Science* 5 24 2007 页码 – 。施普林格，。8. Alonzo Church。逻辑、算术与自动机。载于 *International congress of mathematicians* 23 35, 1962 ，第 – 页。

9. Alessandro Cimatti、Luca Geatti、Nicola Gigante、Angelo Montanari 和 Stefano Tonetta。基于扩展有界响应线性时序逻辑规范的反应式综合。载于 *2020 Formal Methods in Computer Aided Design, FMCAD 2020, Haifa, Israel, September 21-24 2020* 83 92 2020 , , 第 – 页。电气电子工程师学会，。

10. Alexandre David、Peter Gjøl Jensen、Kim Guldstrand Larsen、Marius Mikucio-nis 和 Jakob Haahr Taankvist。Uppaal stratego。载于 Christel Baier 和 Cesare Tinelli 编，*Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18* 9035 *Lecture Notes in Computer Science*，*2015. Proceedings*，第 卷，页码206–211。施普林格，2015。11. Laurent Doyen、Gilles Geeraerts、Jean-Fran˛cois Raskin 和 Julien Reichert。实时逻辑的可实现性。载于 Jo¨el Ouaknine 和 Frits W. Vaandrager 编，*Formal Modeling and Analysis of Timed Systems, 7th International Conference, FORMATS 2009 Budapest Hungary September 14-16* ，，，，*2009. Proceedings*，*Lecture Notes in Computer Science*第5813卷，第133–148页。施普林格，2009。12. Deepak D'Souza 和 P. Madhusudan。针对外部规约的定时控制器综合。载于 Helmut Alt 和 Afonso Ferreira 编，*STACS 2002, 19th Annual Symposium on Theoretical Aspects of Computer Science, Antibes - Juan les Pins, France March 14-16 2002 Proceedings* 2285 *Lecture Notes in Computer*，，，，第 卷，页码 – 。施普林格，。13. Till Hofmann 和 Stefan Schupp。Tacos：用于 MTL 控制器综合的工具。载于 Radu Calinescu 和 Corina S. Pasareanu 编，*Software Engineering and Formal Methods - 19th International Conference, SEFM 2021, Virtual Event, Decem-ber 6-10,* 2021, *Proceedings*，*Lecture Notes in Computer Science*第13085卷，第372–379页。施普林格，2021。14. Marcin Jurdzinski、Jeremy Sproston 和 Fran˛cois Laroussinie。使用一个或两个时钟对概率定时自动机进行模型检测。 *Log. Methods Comput. Sci.*, 4 3 2008( ), 。15. Ron Koymans。使用度量时序逻辑描述实时性质。 *Real Time Syst.* 2 4 255 299, 1990 , ( ): – 。16. Hadas Kress-Gazit、Georgios E. Fainekos 和 George J. Pappas。基于时序逻辑的反应式任务与运动规划。 *IEEE Trans. Robotics*, 25(6):1370–1381, 2009。17. Orna Kupferman、Nir Piterman 和 Moshe Y. Vardi。从活性到及时性。*Formal Methods Syst. Des.*, 34(2):83–103, 2009。18. Guangyuan Li、Peter Gjøl Jensen、Kim Guldstrand Larsen、Axel Legay 和 Danny Bøgsted Poulsen。针对 mtl0 的实用控制器综合 ∞。载于 Hakan Erdogmus 和 Klaus Havelund 编，*Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software, Santa Barbara, CA USA 2017* 102 111 2017 ，，*July 10-14,* ，第 – 页。美国计算机协会，。19. Michael Luttenberger、Philipp J. Meyer 和 Salomon Sickert。通过奇偶博弈从线性时序逻辑规约实用地综合反应式系统。 *Acta Informatica*, 57 1 2 3 36, 2020 (-): – 。20. Oded Maler、Dejan Nickovic 和 Amir Pnueli。根据有界响应性质综合控制器。载于 Werner Damm 和 Holger Hermanns 编，*Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany July 3-7 2007 Proceedings* 4590 *Lecture Notes in Computer*，，，，第 卷，页码 – 。施普林格，。21. Oded Maler、Amir Pnueli 和 Joseph Sifakis。定时系统离散控制器的综合（扩展摘要）。载于 Ernst W. Mayr 和 Claude Puech 编，*STACS 95, 12th Annual Symposium on Theoretical Aspects of Com-puter Science, Munich, Germany, March 2-4,* 1995, *Proceedings*, *Lecture Notes in Computer Science*第900卷，第229–242页。施普林格，1995。

22. Philipp J. Meyer、Salomon Sickert 和 Michael Luttenberger。Strix：显式反应式综合强势回归！载于 Hana Chockler 和 Georg Weissenbacher 编，*Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17*，2018，*Proceedings, Part I*，第 10981 卷，*Lecture Notes in Computer Science*，第 578–586 页。施普林格，2018。

23. Dejan Nickovic 和 Nir Piterman。从 MTL 到确定性定时自动机。载于 Krishnendu Chatterjee 和 Thomas A. Henzinger 编，*Formal Modeling and Analysis of Timed Systems - 8th International Conference, FORMATS 2010, Klosterneuburg Austria September 8-10* 6246 *Lec- , , , ture Notes in Computer Science* 152 167 2010 *2010. Proceedings*，第 卷，，第 – 页。施普林格，。24. Amir Pnueli。程序时序逻辑。载于 *18th Annual Symposium on Foundations of Computer Science Providence Rhode Island USA 31 October - , , , , ,* 第 – 页。*1 November 1977* 46 57 1977 IEEE计算机学会，。25. Shufang Zhu、Lucas M. Tabajara、Jianwen Li、Geguang Pu 和 Moshe Y. Vardi。安全线性时序逻辑综合的符号化方法。载于 Ofer Strichman 和 Rachel Tzoref-Brill 编，*Hardware and Software: Verification and Testing - 13th InternationalHaifa Verification Conference, HVC 2017, Haifa, Israel, November 13-15*，2017，*Proceedings* 10629 *Lecture Notes in Computer Science* 147 162 ，第 卷，，第 – 页。施普林格，2017。

A 扩展技术与分析

现在，我们概述几种用于生成倒计时计时器的方法的技术；这些技术对该方法在实践中的效率至关重要。

基于计时器偏序的剪枝。我们通过跟踪不同持续时间计时器之间的偏序来改进生成过程。如果两个计时器 t_a 和 t_b 及其d（t_a） $<$ d（t_b）在同一时刻引入，我们就知道 t_b 不可能先于 t_a超时。我们在函数中利用这一信息，以剪枝去除更多虚假的超时。这项技术可以显著缩小生成的倒计时器博弈。例如，在公式（$u \rightarrow$（ $[100]c \vee \Diamond[500]\neg c$））且 $u \in \mathcal{I}$， $c \in \mathcal{O}$ 时，如果不采用这项技术，界为 500 的计时器可能会先于界为 100 的计时器超时。重复这一过程会导致界为 100 的所有计时器都被使用，尽管实际上只需要一个。

复杂度简述。最坏情况下，对于某些公式 φ，我们可能必须为所有可能的索引生成计时器，而索引的数量相对于公式 $|\varphi|$的大小呈指数级。因此， $|L|$ 可能达到 $|\varphi|$的三重指数级，而 $|\mathcal{V}|$ 可能达到 $|\varphi|$的双重指数级。语义安全博弈的状态空间 $L \times \mathcal{V}$ 相对于 $|\varphi|$呈三重指数级，因此最坏情况下的求解时间也如此。对于生成的计时器最大索引相对于其持续时间以对数方式有界的公式 φ， $|\mathcal{V}|$ 最坏情况下仅为 $|\varphi|$ 的指数级。

混合展开。为某个界生成所有计时器可能会对求解所得博弈的性能产生负面影响，因为计时器数量过多。解决方法是将显式展开与基于定时器的隐式展开结合起来。为此，我们允许顶层时序算子带有数值界限，而不仅仅是计时器。随后需要修改 introExp。例如

$$iE'_I(\bigcirc[n]\varphi) := \begin{cases} \bigcirc[t^n_{I(n)}]\varphi & \neg Expandexplicit \\ \bigcirc[n-1]\varphi & n > 1, Expandexplicit \\ iE'_I(\varphi) & n = 1, Expandexplicit \end{cases}$$

此外，这样就可以直接处理诸如 single 这样的算子。根据上述复杂度分析，我们对那些生成的计时器数量多于其对数的界进行显式展开。这不会改变复杂度类，但会使 $|\mathcal{V}|$ 指数级减少，同时对 L 的大小影响有限，从而在实践中提升整体性能。

SafeLTL 中的假设$_B$。 由于 $SafeLTL_B$ 不包含无界的“最终”算子，因此我们无法表示 形式的假设 ψ。因此，我们使用近似来处理 ψ形式的简单假设，其中 ψ 是完全有界的，即不含 $\mathcal{W}$。其思路是将这些假设与其他公式一起分别展开。如果它们变得等价于 $\bot$，我们就能确定违规已经发生，系统获胜。否则，它们不会影响博弈结果。这种近似是可靠的，即如果系统在采用近似的情况下获胜，那么它在不采用近似时也会获胜；但它并不完备，也就是说，如果系统在近似情况下输掉博弈，我们无法据此作出任何判断。

B 证明

B.1 第 4 节的证明

定理 1. 设 $\Phi \in \text{SafeLTL}_B$ 和 G 是根据第 4 节所述，由 Φ 构造的倒计时器博弈结构。那么，当且仅当系统在倒计时器博弈（G，不安全$_L$）中获胜时，存在一个系统实现 $\mathcal{L}(\Phi)$。

证明。证明的思路是将给定倒计时游戏的语义安全博弈与 Φ 的展开式对应起来，其中 Φ 是 SafeLTL_B 公式。利用这些展开式，我们可以证明安全博弈与 $\mathcal{L}(\Phi)$ 一致。

设 $G = (\mathcal{T}, d, L, L_0, \mathcal{I}, \mathcal{O}, \delta)$，并设 $(S, S_0, \mathcal{I}, \mathcal{O}, \rho)$ 为根据定义 2 得到的 G 的语义安全博弈的博弈结构。

注意，S 具有 $S = \text{PositiveBooleanCombinations}(\text{cl}(\Phi)) \times \mathcal{V}$ 的形式。我们将 S 中的每个状态与一个 SafeLTL_B 公式相关联，该公式通过将计时器值作为相应的界插入而得到。直观地说，我们对计时器的抽象进行逆向操作。我们定义 $\langle \cdot, \cdot \rangle : \text{SafeLTL}_B^t \times \mathcal{V} \to \text{SafeLTL}_B$ 如下

$$
\begin{aligned}
\langle l, v\rangle &:= l && \text{for } l \in \{ap, \neg ap\}\\
\langle \varphi \ o \ \psi, v\rangle &:= \langle \varphi, v\rangle \ o \ \langle \psi, v\rangle && \text{for } o \in \{\wedge, \vee\}\\
\langle \bigcirc[t]\varphi, v\rangle &:= \bigcirc[v(t)]\langle \varphi, v\rangle &&\\
\langle \Diamond[t]\varphi, v\rangle &:= \Diamond[v(t)-1]\langle \varphi, v\rangle && \text{if } v(t) > 0\\
\langle \Diamond[t]\varphi, v\rangle &:= \bot && \text{if } v(t) = 0\\
\langle \varphi \,\mathcal{W}[t]\psi, v\rangle &:= \langle \varphi, v\rangle \,\mathcal{W}[v(t)-1]\langle \psi, v\rangle && \text{if } v(t) > 0\\
\langle \varphi \,\mathcal{W}[t]\psi, v\rangle &:= \top && \text{if } v(t) = 0\\
\langle \varphi \,\mathcal{W}\, \psi, v\rangle &:= \langle \varphi, v\rangle \,\mathcal{W}\langle \psi, v\rangle &&
\end{aligned}
$$

此外，我们的论证基于 SafeLTL_B 公式的显式展开。因此，我们将单步 expand 定义为展开和插入集：$\text{expand} : \text{SafeLTL}_B \times (\mathcal{I} \cup \mathcal{O}) \to \text{SafeLTL}_B$ as

$$
\begin{aligned}
expand(ap, m) &:= \begin{cases} \top & \text{if } ap \in m \\ \bot & \text{if } ap \notin m \end{cases}\\
expand(\neg ap, m) &:= \begin{cases} \bot & \text{if } ap \in m \\ \top & \text{if } ap \notin m \end{cases}\\
expand(\varphi \ o \ \psi, m) &:= cfold(expand(\varphi, m) \ o \ expand(\psi, m)) && o \in \{\wedge, \vee\}\\
expand(\bigcirc[0]\varphi, m) &:= expand(\varphi, m)\\
expand(\bigcirc[n]\varphi, m) &:= \bigcirc[n-1]\varphi && \text{if } n > 0\\
expand(\Diamond[0]\varphi, m) &:= expand(\varphi, m)\\
expand(\Diamond[n]\varphi, m) &:= cfold(expand(\varphi, m) \vee \Diamond[n-1]\varphi) && \text{if } n > 0\\
expand(\varphi \,\mathcal{W}[0]\psi, m) &:= cfold(expand(\psi, m) \vee expand(\varphi, m))\\
expand(\varphi \,\mathcal{W}[n]\psi, m) &:= cfold(expand(\psi, m) \vee expand(\varphi, m) \wedge (\varphi \,\mathcal{W}[n-1]\psi)) && \text{if } n > 0\\
expand(\varphi \,\mathcal{W}\, \psi, m) &:= cfold(expand(\psi, m) \vee expand(\varphi, m) \wedge (\varphi \,\mathcal{W}\, \psi))
\end{aligned}
$$

其中，常量折叠定义为

$$\begin{aligned} cfold(\top \wedge \psi) &= cfold(\psi \wedge \top) = \psi \\ cfold(\bot \wedge \psi) &= cfold(\psi \wedge \bot) = \bot \\ cfold(\bot \vee \psi) &= cfold(\psi \vee \bot) = \psi \\ cfold(\top \vee \psi) &= cfold(\psi \vee \top) = \top \end{aligned}$$

此外，对于轨迹 $\tau \in \left(2^{\mathcal{I} \cup \mathcal{O}}\right)^{\omega}$，截至步骤 k 的展开定义如下：

$$\begin{aligned} &expand^0(\varphi, \tau) := \varphi \\ &expand^k(\varphi, \tau) := expand(expand^{k-1}(\varphi, \tau), \tau_{k-1}) \qquad \text{for } k > 0 \end{aligned}$$

对于展开，我们可以证明如下结论（证明稍后给出）。

引理 1. 对于任意轨迹 $\tau \in \left(2^{\mathcal{I} \cup \mathcal{O}}\right)^{\omega}$ 和公式 $\varphi \in \text{SafeLTL}_B$:

$$\tau \not\models_0 \varphi \Leftrightarrow \exists k \in \mathbb{N}. expand^k(\varphi, \tau) = \bot$$

现在，我们将展开与生成的博弈联系起来。因此，暂且假设 tree 不进行剪枝，即它只根据布尔顶层文字的取值进行替换，并执行常量折叠。稍后我们将说明剪枝为何可靠。

设 $\equiv$ 定义布尔顶层上公式的等价关系，则

引理 2。对于任意 $v \in \mathcal{V}$、可达的 $\varphi \in L$、 $u \in 2^{\mathcal{I}}$ 和 $c \in 2^{\mathcal{O}}$

$$\langle \rho((\varphi, v), u, c) \rangle \equiv expand(\langle \varphi, v \rangle, u \cup c)$$

证明将在后文给出。

我们将 ρ 在 $s_0 \in S$、 $i \in \left(2^{\mathcal{I}}\right)^{\omega}$ 和 $o \in \left(2^{\mathcal{O}}\right)^{\omega}$ 上针对 $k \in \mathbb{N}$ 的 k 次迭代定义如下：

$$\begin{aligned} &\rho^0(s_0, \tau) := s_0 \\ &\rho^k(s_0, \tau) := \rho(\rho^{k-1}(s_0, \tau), i_{k-1}, o_{k-1}) \qquad \text{对于 } k > 0 \end{aligned}$$

对于任意 $v \in \mathcal{V}$、可达的 $\varphi \in L$、 $i \in \left(2^{\mathcal{I}}\right)^{\omega}$ 和 $o \in \left(2^{\mathcal{O}}\right)^{\omega}$，反复应用引理 2 可得

$$\langle \rho^k((\varphi, v), i, o) \rangle \equiv expand^k(\langle \varphi, v \rangle, i \cup o)$$

对于 $k \in \mathbb{N}$，其中 $i \cup o$ 是逐点定义的。请注意，这仅在 ρ 和 expand 保持语义时成立，也就是说，将它们应用于等价公式会得到等价公式。结合引理 1 可得

$$\begin{aligned} i \cup o \notin \mathcal{L}(\langle \varphi, v \rangle) &\Leftrightarrow i \cup o \not\models_0 \langle \varphi, v \rangle \\ &\Leftrightarrow \exists k \in \mathbb{N}. \langle \rho^k((\varphi, v), i, o) \rangle = \bot \\ &\Leftrightarrow \exists k \in \mathbb{N}. \rho^k((\varphi, v), i, o) \in \{\bot\} \times \mathcal{V} \\ &\Leftrightarrow \exists k \in \mathbb{N}. \rho^k((\varphi, v), i, o) \in \mathit{UNSAFE} \end{aligned}$$

请注意， $L_0 = \{\text{optintroExp}(\Phi)\}$ ，因此 S_0 是单元素集合 $S_0 = \{s_0\}$ ，其中 $s_0 =$(opt introExp(Φ) $\lambda t.\ d(t)$)。根据 introExp 和 opt 的构造， Φ 等价于 $\langle s_0 \rangle$，即 $\mathcal{L}(\Phi) = \mathcal{L}(\langle s_0 \rangle s_0)$，因为 introExp 只应用展开定律并插入计时器，而这些计时器随后会由 $\langle \cdot, \cdot \rangle$ 相应地重新替换。因此，

$$i \cup o \notin \mathcal{L}(\Phi) \Leftrightarrow \exists k \in \mathbb{N}.\ \rho^k(s_0, i, o) \in \mathit{UNSAFE}$$

由于 i 和 o 都是全称量化的，我们只需将这一论述推广到系统和策略的语境中。我们通过证明当且仅当不存在满足 $\mathcal{L}(\Phi)$ 的实现时，环境才会在博弈中获胜，来证明主要结论。如果不存在满足 $\mathcal{L}(\Phi)$ 的实现，那么对于所有系统，都存在一个输入序列 $i \in \left(2^{\mathcal{I}}\right)^{\omega}$ ，使得由此产生的输出序列 $o \in \left(2^{\mathcal{O}}\right)^{\omega}$ 满足：对于 $t \in \mathbb{N}$, $i \cup o \notin \mathcal{L}(\Phi)$，有 $(c_{t+1}, o_t) = \gamma(c_t, i_t)$。根据上述等价关系， $\exists k \in \mathbb{N}.\ \rho^k(s_0, i, o) \in$ 不安全，即任何实现都会在安全博弈中落败。由于安全博弈可解当且仅当存在无记忆策略使其可解，因此环境总能赢得安全博弈。由此，等价关系成立。

现在我们来证明两个引理:

证明（引理 1）。设 $\tau[n] := \lambda t.\ \tau_{t+n}$ 是从位置 n 开始的 τ。我们证明以下更一般的命题

$$\forall t \in \mathbb{N}.\ \tau \not\models_t \varphi \ \Leftrightarrow\ \exists k \in \mathbb{N}. \mathit{expand}^k(\varphi, \tau[t]) = \bot$$

通过对 φ 进行结构归纳:

ap:

$$ap \not\models_t \tau \ \Leftrightarrow\ ap \notin \tau_t \ \Leftrightarrow\ \mathit{expand}(ap, \tau[t]) = \bot \ \Leftrightarrow\ \mathit{expand}^1(ap, \tau[t]) = \bot$$

$\neg ap$:

$\neg$ap $\not\models_t \tau \ \Leftrightarrow$ap $\in \tau_t \ \Leftrightarrow$expand $(\neg$ap $\tau[t]) = \bot \ \Leftrightarrow$expand $^1(\neg$ap $\tau[t]) = \bot\varphi \wedge \psi$:

$$\begin{aligned}
& \tau \not\models_t \varphi \wedge \psi \\
\Leftrightarrow\ & (\tau \not\models_t \varphi) \vee (\tau \not\models_t \psi) \\
\Leftrightarrow\ & (\exists k \in \mathbb{N}.\ \mathit{expand}^k(\varphi, \tau[t]) = \bot) \\
& \vee (\exists k \in \mathbb{N}.\ \mathit{expand}^k(\psi, \tau[t]) = \bot) \quad \text{by IH} \\
\Leftrightarrow\ & \exists k \in \mathbb{N}.\ \mathit{expand}^k(\varphi \wedge \psi, \tau[t]) = \bot \ \text{due to } \mathit{cfold}
\end{aligned}$$

$\varphi \vee \psi$:

$$\begin{aligned}
& \tau \not\models_t \varphi \vee \psi \\
\Leftrightarrow\ & (\tau \not\models_t \varphi) \wedge (\tau \not\models_t \psi) \\
\Leftrightarrow\ & (\exists k \in \mathbb{N}.\ \mathit{expand}^k(\varphi, \tau[t]) = \bot) \\
& \wedge (\exists k \in \mathbb{N}.\ \mathit{expand}^k(\psi, \tau[t]) = \bot) \ \text{by IH} \\
\Leftrightarrow\ & \exists k \in \mathbb{N}.\ \mathit{expand}^k(\varphi \vee \psi, \tau[t]
\end{aligned}$$

请注意，最后几步成立，因为当第一个析取项变为 $\bot$ 时，常量折叠会将其移除。析取展开为 $\bot$ 之前的界，是两个析取项中较大的那个。

$\bigcirc[n]\psi$:

$$\begin{aligned}
& \tau \not\models_t \bigcirc[n]\psi \\
\Leftrightarrow\ & \tau \not\models_{t+n} \psi \\
\Leftrightarrow\ & \exists k \in \mathbb{N}.\ expand^k(\psi, \tau[t+n]) = \bot \ \text{ by IH} \\
\Leftrightarrow\ & \exists k \in \mathbb{N}.\ expand^k(\bigcirc[n]\psi, \tau[t]) = \bot \ \text{ by iterating rule for } \bigcirc[\cdot] \text{ in } expand
\end{aligned}$$

$\Diamond[n]\psi$:

$$\begin{aligned}
& \tau \not\models_t \Diamond[n]\psi \\
\Leftrightarrow\ & \forall i \leq n.\ \tau \not\models_{t+i} \psi \\
\Leftrightarrow\ & \forall i \leq n.\ \exists k \in \mathbb{N}.\ expand^k(\psi, \tau[t+i]) = \bot \ \text{ by IH} \\
\Leftrightarrow\ & \exists k \in \mathbb{N}.\ expand^k(\Diamond[n]\psi, \tau[t])
\end{aligned}$$

最后一步成立，因为 expand 会将 $[n]\psi$ expand 为所有 n步骤上的析取项。若且唯若所有子部分都变为 $\bot$，则可将其折叠为 $\bot$。

$\varphi\,\mathcal{W}\,\psi$:

$\tau \not\models_t \varphi\,\mathcal{W}\,\psi \Leftrightarrow\ \exists j.(\tau \not\models_{t+j} \varphi)\ \wedge \forall j \leq j.(\tau \not\models_{t+i} \psi) \Leftrightarrow\ \exists j.(\exists k_1 \in \mathbb{N}.\text{expand}^{k_1}(\varphi, \tau[t+i]))\ \wedge \forall j \leq j.(\exists k_2 \in \mathbb{N}.\text{expand}^{k_2}(\psi, \tau[t+i]))$

注意

$$expand^k(\varphi\,\mathcal{W}\,\psi, \tau) = a_0 \vee b_0 \wedge (a_1 \vee b_1 \wedge (\ldots(\varphi\,\mathcal{W}\,\psi)))$$

其中 a_i:=expand $^{k-i}(\psi, \tau)$ 和 b_i =expand$^{k-i}(\varphi, \tau)$ 是 φ和 ψ 在不同展开步骤中的展开式。现在注意到，只有在存在某个展开步骤 m，使得 b_m 为 $\bot$ 且对于所有 $i \leq m$， a_i 也为 $\bot$时，该项才能折叠为 $\bot$。因此，上述项等价于

$$\exists k \in \mathbb{N}.\ expand^k(\varphi\,\mathcal{W}\,\psi, \tau[t]) = \bot$$

注意， k 是 $k_1 + i$ 和所有 $k_2 + j$中的最大值。

$\varphi\,\mathcal{W}[n]\psi$: 与前一种情况类似，但这里有一个界。如果不满足该条件，那么在界内，要么 φ 不满足，要么 ψ 从未满足。当展开式中的相应项变为 $\bot$ 时， $\bot$ 常量折叠会将所有内容折叠为 $\bot$。

□

证明（引理 2）。 $\rho((\varphi, v)\ u, c)$ 定义为 (l', v')，其中

– T:= $\{t \mid v(t)\ - 1 \leq 0\}$– (e, ψ) := 挤压(至(T, 树(φ, u, c))) – l' := 优化(introExp(ψ)). –

$$v' := \lambda t.\ \begin{cases} v(e(t)) & \text{if } e(t) \in \mathcal{T} \\ d(t) & \text{if } e(t) = RESET \end{cases}$$

首先注意，squeeze 只会重新排序计时器，因此在特定条件下，$\langle \cdot, \cdot \rangle$在 v' 中所做的更改下保持不变式。由于 opt 只修改顶层布尔层级，接下来只需证明 introExp、tree 和 to 在 $\langle \cdot, \cdot \rangle$ 下的行为与 expand 相同。首先注意，to 的超时行为对应于 expand 中带界算子的三种情况，因为在定时器游戏中，我们使用的计时器持续时间比相应的界增加了。随后，to 只需应用神经元元素，该元素会通过常量折叠移除。introExp 按照展开定律展开公式，与其他情况中的 expand 一样。根据构造，引入计时器在 $\langle \cdot, \cdot \rangle$ 下是等价的。tree 在与 expand 相同的情况下为文字赋值，即只针对顶层布尔文字。□

接下来需要证明，树所做的剪枝是可靠的，也就是说，不会改变谁获胜。对于剪枝规则 (1) 和 (2)，这显而易见，因为在这些情形下，玩家可以强制进入各自的获胜状态。对于剪枝规则 (3)，系统被迫选择后继 ψ_G 而不是 ψ_B，使得 $\mathcal{L}(\psi_B) \subseteq \mathcal{L}(\psi_G)$。因此，任何会使 ψ_G 输掉的策略，也会使 ψ_B 输掉；也就是说，这不会改变环境是否获胜。类似地，对于剪枝规则 (4)，如果系统会因规则选出的选项 ψ_S 而输，那么环境只需选择该选项；否则，系统无论如何都会获胜，因为对于环境的另一种选择 ψ_N，$\mathcal{L}(\psi_S) \subseteq \mathcal{L}(\psi_N)$。□

B.2 第 5 节的证明

定理 2. 求解倒计时定时器博弈是 *EXPTIME*-完备的。

证明。EXPTIME 包含性源于构建并求解语义安全博弈结构 G 所需时间相对于倒计时器游戏结构 G_T 的大小呈指数级。为此，首先注意到博弈 G 中状态 $n_s = |L| \cdot |\mathcal{V}|$ 的数量可由下式界定

$$n_s = |L| \cdot |\mathcal{V}| \leq |L| \cdot (\max d(t) \ : \ t \in \mathcal{T})^{|\mathcal{T}|} \leq |G_T| \cdot (2^{|G_T|})^{|G_T|} \leq 2^{poly(|G_T|)},$$

其中 $|G_T|$ 是计时器博弈的编码长度，数值采用二进制编码。G 中转移的数量可由下式界定

$$n_t = n_s \cdot |2^{\mathcal{I}}| \cdot |2^{\mathcal{O}}| \leq 2^{poly(|G_T|)}.$$

由于博弈结构 G 的大小为 $n_s + n_t + |2^{\mathcal{I}}| + |2^{\mathcal{O}}| \leq 2^{poly(|G_T|)}$，且求解安全博弈所需时间与博弈图的大小呈线性关系，因此求解 (G_T, 不安全) 可在指数时间内完成。

我们通过从 [14] 中引入的倒计时博弈进行归约，证明了 EXPTIME-困难性。倒计时博弈是一种双人博弈，其中一名玩家若能在恰好经过 c 个时间单位后进行转移，无论另一名玩家如何行动，就能获胜。[14] 中证明了，判定倒计时博弈中哪名玩家获胜的问题是 EXPTIME 完全的。

形式上，倒计时博弈是一个二元组 $C = (S_C, T_C)$，其中 S_C 是状态集，$T_C \subseteq S_C \times (\mathbb{N} \setminus \{0\}) \times S_C$ 是转移关系。对于 $(s, \Delta, s') \in T$，Δ 称为该转移的持续时间。C 的一个配置是一个二元组 (s, c)，其中 $s \in S_C$ 且 $c \in \mathbb{N}$。博弈过程如下。在配置 (s, c) 中，玩家1选择一个数 Δ，使得 $0 < \Delta \leq c$ 且对于某个状态 $s' \in S_C$，有 $(s, \Delta, s') \in T_C$。然后，玩家 2 选择一条持续时间为 Δ 的转移 $(s, \Delta, s') \in T_C$。后继配置为 $(s', c - \Delta)$。有两种类型的终端配置 (s, c)，在这些配置中无法进行任何移动：

– 如果 $c = 0$，那么 (s, c) 是终端配置，且玩家1获胜。– 如果对于所有转移 $(s, \Delta, s') \in T_C$，我们都有 $\Delta > c$，那么 (s, c) 是终端配置，且玩家 2 获胜。

倒计时博弈的求解问题是：给定一个倒计时博弈 $C = (S_C, T_C)$ 和一个配置 (s, c)，其中 C 中所有转移的持续时间以及数字 c 均以二进制给出，判断玩家1是否有策略能够从配置 (s, c) 出发，确保到达一个获胜配置。判定倒计时博弈中的获胜者是 EXPTIME 完全问题 [14]。我们将展示如何把这个问题编码为倒计时器博弈的求解问题。

设 $C = (S_C, T_C)$ 是一个倒计时博弈，(s, c) 是其初始配置。不失一般性，我们假设所有持续时间都大于或等于 2（这可以通过将所有持续时间和 c 乘以 2 来实现）。我们构造一个定时器游戏 $(G_T, \text{不安全}_L)$，其中 $G_T = (\mathcal{T}, d, L, L_0, \mathcal{I}, \mathcal{O}, \delta)$，使得当且仅当玩家1在倒计时博弈中获胜时，环境玩家在 $(G_T, \text{不安全}_L)$ 中获胜。博弈结构 G_T 定义如下：

– $\mathcal{T} := \{g\} \cup \{t_\Delta \mid \exists s, s'. (s, \Delta + 1, s') \in T_C\}$ 持续时间为 $d(g) := c$ 且 $d(t_\Delta) = \Delta$。计时器 g 用于表示配置中的全局倒计时值。其他计时器用于表示转移的持续时间。请注意，其持续时间会减一，这是因为我们必须引入辅助转移。

– $L := S_C \cup T_C \cup \{\text{SAFE、UNSAFE}\}$。
  • 位置 S_C 是倒计时游戏中对应的状态。
  • 位置 T_C 是辅助状态。在这些辅助状态中，通过在相应状态中停留所需的时间（使用一些计时器 t_Δ），g 的值会按倒计时游戏中相应转移的持续时间递减。
  • 位置 SAFE 和 UNSAFE 表示博弈的终止状态（并用于避免转移关系中出现不合理的选择）。

– 我们令 $\mathcal{I} :=$ 命题(C)，其中命题(C)对应于 C 中持续时间的二进制编码。因此，

$$2^{\mathcal{I}} \supseteq \{\Delta \mid \exists s, s'. (s, \Delta, s') \in T_C\},$$

而 $2^{\mathcal{I}}$ 中的元素代表 C 中玩家1的动作。通过重定向至 SAFE，δ 处理这些虚假选择。

– 我们令 $\mathcal{O} := \text{Propositions}(T_C)$，其中 $\text{Propositions}(T_C)$ 是与转移 T_C 的二进制编码相对应的命题。因此，

$$2^{\mathcal{O}} \supseteq T_C,$$

并且， $2^{\mathcal{O}}$ 中的元素表示 C中玩家 2 的动作。伪造的选择通过将其重定向至不安全状态来由 δ 处理。

在 G_T 中，我们使用以下效果：

$$\begin{aligned} ea &:= \lambda t.\ RESET \\ eg &:= \lambda t.\ \begin{cases} g & \text{if } t = g \\ RESET & \text{otherwise} \end{cases} \\ e_\Delta &:= \lambda t.\ \begin{cases} g & \text{if } t \in \{g, t_\Delta\} \\ RESET & \text{otherwise} \end{cases} \end{aligned}$$

转移关系 δ 定义如下：

$$\delta(s, i, o, TO) \quad := (s, ea) \qquad \text{if } s \in \{SA\text{FE，不安全状态}\}$$

$$\delta(s, \Delta, t_C, TO) := \begin{cases} (SAFE, ea) & \text{if } \neg\exists s' \in S.\ (s, \Delta, s') \in T_C \\ (UNSAFE, ea) & \text{otherwise if } \neg\exists s' \in S_C.\ t_C = (s, \Delta, s') \\ (t_C, eg) & \text{otherwise if } g \notin TO \\ (SAFE, ea) & \text{otherwise if } g \in TO \end{cases} \quad \text{if } s \in S_C$$

$$\delta(t_C, i, o, \text{至}) := \text{let } (s, \Delta, s') := t_C \text{ in} \qquad \begin{cases} (t_C, e_\Delta) & \text{if } g, t_{\Delta-1} \notin TO \\ (s', eg) & \text{if } g \notin TO \wedge t_{\Delta-1} \in TO \\ (SAFE, ea) & \text{if } g \in TO \wedge t_{\Delta-1} \notin TO \\ (UNSAFE, ea) & \text{if } g, t_{\Delta-1} \in TO \end{cases} \quad \text{if } t_C \in T_C$$

转移关系将 C 中玩家 1 和玩家 2 的行动分别建模为 G_T 中环境玩家和系统玩家的选择。

上述归约是多项式时间的。根据构造，当且仅当玩家 1 赢得 C 时，环境玩家赢得 $(G_T, \text{不安全})$。因此，我们可以得出结论：求解倒计时定时器博弈是 EXPTIME 困难的。

这就完成了 EXPTIME 完全性的证明。

C 示例：倒计时器博弈

示例 3. 图 5 描绘了一个倒计时器博弈，它对一个 SafeLTL$_B$ 公式进行编码，该公式规定了打印机应满足的一些简单要求。唯一的输入命题是 j （表示“作业”），输出命题是 i （表示“空闲”）、 w （表示“已预热”）和 p （表示“已打印”）。该规约为

$$\varphi = i \wedge \square\Big(\neg(w \wedge p) \wedge \big((\neg j \wedge i) \rightarrow \bigcirc i\big) \wedge \big((j \wedge i) \rightarrow \bigcirc((\square[9]w)) \wedge (\neg i\, \mathcal{U}[299]p \wedge i)\big)\Big).$$

该规约指出，打印机不能同时处于预热状态

并且已经完成打印；没有打印任务时，打印机保持空闲。如果打印机处于空闲状态且有打印任务，它将在至多 299 个时间步内完成打印，且预热阶段至少持续 9 个时间步。□

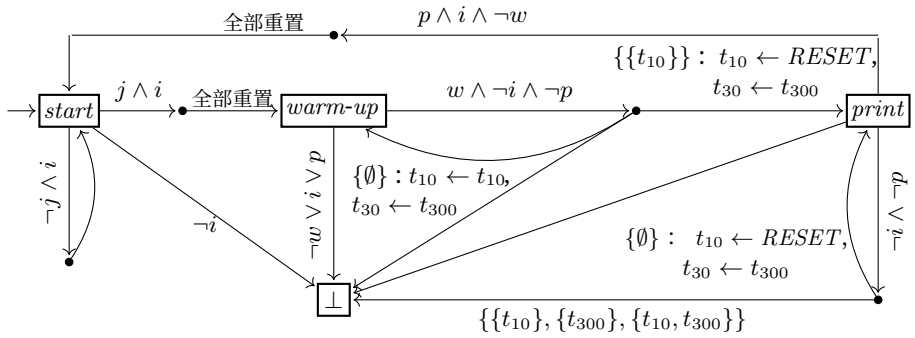


图 5：包含 $L = \{$start、warm-up、print$\}$ 的倒计时器博弈。为方便起见，我们将每个转移拆分为两部分。第一部分标记为关于 $\mathcal{I} \cup \mathcal{O}$ 的布尔公式 α。第二部分标记为一对 $T: e$，其中 $T \subseteq 2^{\mathcal{T}}$ 是在给定步骤中超时的计时器集合所组成的集合，$e \in \mathcal{E}$ 是一个作用。连接位置 l 和 l' 的一对标签 α、$T: e$ 表示存在满足 α 和 $\tau \in T$ 的 $(i, o) \in 2^{\mathcal{I}} \times 2^{\mathcal{T}}$，使得 $\delta(l, i, o, \tau) = (l', e)$。我们将重置全部表示为 reset-all $:= 2^{\mathcal{T}}: e_{reset}$，其中对于所有 $t \in \mathcal{T}$，$e_{reset}(t) =$ 重置。为简单起见，我们省略指向不安全状态 $\bot$ 的转移中无关的标签元素。

D 基准测试

D.1 办公室机器人基准测试

适用于 $N \in \{1, 2, 3, 4\}$ 个办公室的清洁机器人

- $\mathcal{I} = \emptyset$– $\mathcal{O} = \{$走廊，办公室$_1$，办公室$_2$，办公室$_3$，办公室$_4\}$–
  $\Phi_{Clean(N)} := \varphi_{start} \wedge \varphi_{position} \wedge \varphi_{mutex} \wedge \varphi_{clean-duration} \wedge \varphi^N_{clean}$

  - 机器人从走廊开始。φ_{start} :=走廊

  - 机器人至少位于一个位置。$\varphi_{position}: =$ (走廊 $\vee$办公室$_1 \vee$ 办公室$_2 \vee$办公室$_3 \vee$ 办公室$_4$)

□

- 机器人至多位于一个位置。

$$\begin{aligned}\varphi_{mutex} := &\square(corridor \rightarrow (\neg office_1 \wedge \neg office_2 \wedge \neg office_3 \wedge \neg office_4))\\ \wedge\ &\square(office_1 \rightarrow (\neg corridor \wedge \neg office_2 \wedge \neg office_3 \wedge \neg office_4))\\ \wedge\ &\square(office_2 \rightarrow (\neg office_1 \wedge \neg corridor \wedge \neg office_3 \wedge \neg office_4))\\ \wedge\ &\square(office_3 \rightarrow (\neg office_1 \wedge \neg office_2 \wedge \neg corridor \wedge \neg office_4))\\ \wedge\ &\square(office_4 \rightarrow (\neg office_1 \wedge \neg office_2 \wedge \neg office_3 \wedge \neg corridor))\end{aligned}$$

- 开始清洁办公室时，至少清洁 10 分钟。

$$\varphi_{clean-duration} := \bigwedge_{i=1}^{4} \square\left(corridor \rightarrow \bigcirc(office_i \rightarrow \square[10] office_i)\right)$$

- 每 12 小时（720 分钟）开始清洁办公室 $1 \ldots N$。

$$\varphi_{clean}^{N} := \bigwedge_{i=1}^{N} \square\Diamond[720] office_i$$

清洁和充电机器人适用于 $N \in \{1, 2, 3, 4\}$ 间办公室

– $\mathcal{I} = \emptyset$– $\mathcal{O} = \{$走廊，办公室$_1$,办公室$_2$,办公室$_3$,办公室$_4$,充电$\}$– $\Phi_{Clean_C(N)}$：

$= \Phi_{Clean(N)} \wedge \varphi_{charge-duration} \wedge \varphi_{charge}$

- 公式 $\Phi_{Clean(N)}$ 如上所述。
- 充电需要20分钟，充电站位于走廊。

$$\begin{aligned}\varphi_{charge-duration} := &\square\left(\neg charge \rightarrow \bigcirc(charge \rightarrow \square[20] charge)\right)\\ \wedge\ &\square(charge \rightarrow corridor)\end{aligned}$$

- 每6小时（360分钟）内充电一次。

$$\varphi_{charge} := \square\Diamond[360] charge$$

有人类在场时，清洁机器人可服务于 $N \in \{1, 2, 3, 4\}$间办公室

– $\mathcal{I} = \{$人类$_1$,人类$_2$,人类$_3$,人类$_4\}$– $\mathcal{O} = \{$走廊,
办公室$_1$,办公室$_2$,办公室$_3$,办公室$_4\}$– $\Phi_{Clean_H(N)}$：

$= \Phi_{Clean(N)} \wedge \varphi_{no-interfere}$

- 公式 $\Phi_{Clean(N)}$ 如上所述。
- 如果有人在办公室，机器人就不应待在办公室。

$$\varphi_{no-interfere} := \bigwedge_{i=1}^{4} \square(human_i \rightarrow \neg office_i)$$

机器人夜间清洁 $N \in \{1, 2, 3, 4\}$ 个办公室

– $\mathcal{I} = \{$夜间$\}$– $\mathcal{O} = \{$走廊、办公室$_1$, 办公室$_2$, 办公室$_3$, 办公室$_4\}$– $\Phi_{Clean_N}(N)$: $= \varphi_{night} \rightarrow (\varphi_{clean-night} \wedge \varphi^{N}_{clean-start-night})$ • 假设：每隔至少 12 小时会进入一次夜间，且夜间持续 12 小时。$\varphi_{night} :=$ (¬夜间 $\rightarrow \Diamond$[720]夜间) $\wedge$ (¬夜间 $\rightarrow$ (夜间 $\rightarrow \Box$[720]夜间))

- 公式 $\varphi_{start}, \varphi_{position}, \varphi_{mutex}, \varphi_{clean-duration}$ 与之前相同。
- 只在夜间清洁。

$$\varphi_{clean-night} := \bigwedge_{i=1}^{4} \Box(\neg night \rightarrow \neg office_i)$$

- 在夜幕降临时开始清洁。

$$\varphi^{N}_{clean-start-night} := \bigwedge_{i=1}^{N} \Box(\neg night \rightarrow \bigcirc(night \rightarrow \Diamond[60] office_i))$$

机器人配送咖啡至 $N \in \{1,2, 3, 4\}$ 个办公室

– $\mathcal{I} = \{$请求$_1$,请求$_2$,请求$_3$,请求$_4\}$– $\mathcal{O} = \{$走廊，办公室$_1$, 办公室$_2$, 办公室$_3$, 办公室$_4$, 制作咖啡$\}$– $\Phi_{Coffee(N)} = \varphi_{start} \wedge \varphi_{position} \wedge \varphi_{mutex} \wedge \varphi_{corridor} \wedge \varphi_{coffee-making} \wedge \varphi^{N}_{coffee-delivery}$: 公式 φ_{start}、$\varphi_{position}$、φ_{mutex} 同前。

- 通过走廊至少需要 120 秒。

$$\varphi_{corridor} := \Box(\neg corridor \rightarrow \bigcirc(corridor \rightarrow \Box[120] corridor))$$

- 咖啡机位于办公室$_1$，制作咖啡需要 180 秒。$\varphi_{coffee-making}$ ： = （制作咖啡 $\rightarrow$办公室$_1$）$\wedge$ $\Box$(¬制作咖啡 $\rightarrow \bigcirc$（制作咖啡 $\rightarrow \Box$[180]制作咖啡））
- 咖啡请求必须在 10 分钟内处理。

$$\varphi^{N}_{coffee-delivery} := \bigwedge_{i=1}^{N} \Box(request_{5-i} \rightarrow (\Diamond[600] makeCoffee) \wedge (\Diamond[600] office_{5-i}))$$

机器人配送咖啡至 $N \in \{1,2, 3, 4\}$ 个办公室并充电

– $\mathcal{I} = \{$请求$_1$,请求$_2$,请求$_3$,请求$_4\}$– $\mathcal{O} = \{$走廊，办公室$_1$, 办公室$_2$, 办公室$_3$, 办公室$_4$, 制作咖啡、充电$\}$– $\Phi_{CoffeeC(N)} = \Phi_{Coffee(N)} \wedge \varphi_{charge-coffee}$ ： 公式 $\Phi_{Coffee(N)}$ 如上。

- 充电需要20分钟，且必须在6小时后进行。$\varphi_{charge-coffee} :=$ (¬充电 $\rightarrow$ (充电 $\rightarrow$ ($\Box$[1200] 充电) $\wedge$($\Box\bigcirc\Box$[1200]充电))) $\wedge$ (充电 $\rightarrow$ (¬充电 $\rightarrow \Diamond$[21600]¬充电)) $\wedge$ ($\Box\Diamond\bigcirc$[21600] 充电))) $\wedge$ (充电 $\rightarrow$ 走廊)

D.2 改编自 [13] 的基准测试

传送带 这是对 [13] 中示例 3（传送带）的改编。该规约以毫秒为单位进行离散化。

– $\mathcal{I} = \{$释放算子、卡住、恢复$\}$– $\mathcal{O} = \{$移动、停止$\}$– SafeLTL$_B$规约：$\Phi_{\text{conv-belt}} := \varphi_{luggage} \rightarrow (\varphi_{move} \wedge \varphi_{stop} \wedge \varphi_{release} \wedge \varphi_{resume})$

- 假设： 释放算子 当且仅当行李解除卡住状态时为真。$\varphi_{luggage}$: $= \square\big((\text{卡住} \wedge (\neg\text{卡住})) \leftrightarrow (\text{释放算子})\big)$
- 传送带要么移动，要么停止。φ_{move} : $= (\text{移动} \vee \text{停止}) \wedge (\neg(\text{移动} \wedge \text{停止}))$
- 如果行李被卡住，传送带就必须停止。

$$\varphi_{stop} := \square(stuck \rightarrow stop)$$

- 释放后，传送带必须保持停止 2 秒。$\varphi_{release}$:= (释放 $\rightarrow \square[2000]$停止)
- 传送带释放后，我们希望在 3 秒后恢复运行。（备注：原始来源中未明确具体行为。）φ_{resume} := (恢复 $\wedge \neg$卡住 $\rightarrow \Diamond[3000]$(移动 $\mathcal{W}$ 卡住))

机器人摄像头 这是对 [13] 中例 2(robot) 的改编。该规约以毫秒为单位进行离散化。

– $\mathcal{I} = \{$拾取、放置、移动$\}$– $\mathcal{O} = \{$开启、关闭 $\}$– SafeLTL$_B$规约：$\Phi_{\text{robo-cam}} := \varphi_{robot} \rightarrow (\varphi_{on-off} \wedge \varphi_{switch-off} \wedge \varphi_{switch-on})$

- 假设：pick、put、move 中最多只有一个成立，并且它们满足定义它们的假设。φ_{robot} := (¬(pick $\wedge$ put)) $\wedge$ (¬(pick $\wedge$ move)) $\wedge$ (¬(put $\wedge$ move)) $\wedge\wedge$ $\square\square($ $\rightarrow \bigcirc\ \square[3000]$ $\wedge \Diamond[3001]$ $\square)$ pick ((move) $\wedge\ \square($ $\rightarrow \bigcirc\ \square[3000]$ $\wedge \Diamond[3001]$ $)$ put) put ((move) pick)$\wedge$ pick

- 摄像头要么处于开启状态，要么处于关闭状态。

$$\varphi_{on-off} := \square(\neg(on \leftrightarrow off))$$

- 摄像头不会连续开启超过 4 秒。

$$\varphi_{switch-off} := \square(on \rightarrow \Diamond[4000]\neg on)$$

- 1 在拾取或放置前一秒，必须打开摄像头。

$$\varphi_{swicth-on} := \square((\Diamond[1000]pick) \rightarrow on) \wedge \square((\Diamond[1000]put) \rightarrow on)$$

有三个道口的铁路（$\boldsymbol{T1}$、 $\boldsymbol{T2}$、 $\boldsymbol{T3}$）这是对 [13] 中示例 1（铁路）的改编。

$\mathcal{I} = \{$在$_1$,在$_2$,在$_3$,行驶，关闭 $_1$，打开$_1$,通行^1'– 关闭 $_2$，打开$_2$,通行$_2$,关闭 $_3$，打开 $_3$，通行$_3\}$– $\mathcal{O} = \{$打开$_1$,关闭$_1$,打开$_2$,关闭$_2$,打开$_3$,关闭$_3\}$– SafeLTL$_B$规约:

$$\Phi_{\text{rail(T1,T2,T3)}} := (\varphi_{train(T1,T2,T3)} \wedge \varphi_{gates}) \rightarrow (\varphi_{close} \wedge \varphi_{open})$$

- 假设：列车在 $T1$ 分钟后通过道口 1，在$T2$ 分钟后通过道口 2，在 $T3$ 分钟后通过道口 3。通过道口后，列车继续行驶。

$$\begin{aligned}\varphi_{train(T1,T2,T3)} := & (\Box[60 \cdot T1]\neg in_1) \\ \wedge & \ (\Box[60 \cdot T2]\neg in_2) \\ \wedge & \ (\Box[60 \cdot T3]\neg in_3) \\ \wedge & \ (\Box[60 \cdot T3 + 120]\neg travel) \\ \wedge & \ opened_1 \wedge opened_2 \wedge opened_3 \\ \wedge & \ \Box(travel \rightarrow \bigcirc travel) \\ \wedge & \ \Box(travel \rightarrow \neg(in_1 \vee in_2 \vee in_3)\end{aligned}$$

- 假设：道闸的行为。

$$\begin{aligned}\varphi_{gates} := \textstyle\bigwedge_{i=1}^{3} \Big(& \ \Box(\neg(opened_i \wedge closed_i)) \\ & \wedge\Box(\neg(transit_i \wedge closed_i)) \\ & \wedge\Box(\neg(transit_i \wedge opened_i)) \\ & \wedge\Box(\neg transit_i \rightarrow \bigcirc(transit_i \rightarrow \Box[60]transit_i)) \\ & \wedge\Box(opened_i \wedge \neg close_i \rightarrow \bigcirc opened_i) \\ & \wedge\Box(opened_i \wedge close_i \rightarrow \bigcirc(transit_i \wedge \Diamond[60] \bigcirc closed_i)) \\ & \wedge\Box(closed_i \wedge \neg open_i \rightarrow \bigcirc closed_i) \\ & \wedge\Box(closed_i \wedge open_i \rightarrow \bigcirc(transit_i \wedge \Diamond[60] \bigcirc opened_i))\Big)\end{aligned}$$

- 列车在道闸内时，道闸处于关闭状态 d.

$$\varphi_{close} := \bigwedge_{i=1}^{3} \Box(in_i \rightarrow closed_i)$$

- 列车离开后，道闸应打开。

$$\varphi_{close} := \bigwedge_{i=1}^{3} \Box(travel \rightarrow \Diamond[120]opened_i)$$

有两个道口的铁路（$\boldsymbol{T1, T2}$）这是对 [13] 中例1（铁路）的改编。

– $\mathcal{I} = \{$进$_1$,进$_2$,行驶，关闭$_1$，开启$_1$，过境$_1$,关闭$_2$，开启$_2$，过境 $_2\}$

– $\mathcal{O} = \{$打开$_1$,关闭$_1$,打开$_2$,关闭$_2\}$–
SafeLTL$_B$规约：

$$\Phi_{\text{rail(T1,T2)}} := (\varphi_{train(T1,T2)} \wedge \varphi'_{gates}) \rightarrow (\varphi'_{close} \wedge \varphi'_{open})$$

- 假设：列车在经过 $T1$ 分钟后通过道口 1，在经过 $T1+T2$ 分钟后通过道口 2。通过道口后，列车继续行驶。

$$\begin{aligned}\varphi_{train(T1,T2)} := & (\square[60 \cdot T1]\neg in_1) \\ \wedge\ & (\square[60 \cdot (T1+T2)]\neg in_2) \\ \wedge\ & (\square[60 \cdot (T1+T2) + 120(\star)]\neg travel) \\ \wedge\ & opened_1 \wedge opened_2 \\ \wedge\ & \square(travel \rightarrow \bigcirc travel) \\ \wedge\ & \square(travel \rightarrow \neg(in_1 \vee in_2)\end{aligned}$$

- 假设：道闸的行为。

$$\begin{aligned}\varphi_{gates} := \bigwedge_{i=1}^{2} \Big(& \square(\neg(opened_i \wedge closed_i)) \\ & \wedge\square(\neg(transit_i \wedge closed_i)) \\ & \wedge\square(\neg(transit_i \wedge opened_i)) \\ & \wedge\square(\neg transit_i \rightarrow \bigcirc(transit_i \rightarrow \square[60] transit_i)) \\ & \wedge\square(opened_i \wedge \neg close_i \rightarrow \bigcirc opened_i) \\ & \wedge\square(opened_i \wedge close_i \rightarrow \bigcirc(transit_i \wedge \diamondsuit[60] \bigcirc closed_i)) \\ & \wedge\square(closed_i \wedge \neg open_i \rightarrow \bigcirc closed_i) \\ & \wedge\square(closed_i \wedge open_i \rightarrow \bigcirc(transit_i \wedge \diamondsuit[60] \bigcirc opened_i))\Big)\end{aligned}$$

- 列车进入道口时，道闸关闭。

$$\varphi_{close} := \bigwedge_{i=1}^{2} \square(in_i \rightarrow closed_i)$$

- 列车离开后，道闸应打开。

$$\varphi_{close} := \bigwedge_{i=1}^{2} \square(travel \rightarrow \diamondsuit[120] opened_i)$$

$\star$ 对于（$T1 = 2,\ T2 = 4$），此处应为 100，而不是 120，这是因为我们发现错误时已经太晚了。

D.3 来自 [9]的基准测试

具体的基准测试取自 https://gitlab.fbk.eu/lgeatti/ksy-tool.git。它们由四种公式类型组成：$\varphi_A(N)$ $\varphi_B(N)$ $\varphi_C(N)$ $\varphi_D(N)$，其中 $N \in \{2, \ldots, 200\}$。φ_A, φ_B 是可实现的，而 φ_C, φ_D 是不可实现的。

$\mathcal{I} := \{u_0, \ldots, u_N\}$ and $\mathcal{O} := \{c0, \ldots, cN\}$.

$$\varphi_A(N) := \bigwedge_{i=0}^{N-1} (\bigcirc[1+2+\cdots+i]\,\Box\, c_i) \wedge \bigcirc[1+2+\cdots+N]\,\Box(c_N \vee u)$$

$$\varphi_B(N) := \bigwedge_{i=0}^{N} (\bigcirc[1+2+\cdots+i]\,\Box(c_i \vee u_i)$$

$$\varphi_C(N) := (\Box\, c_0) \wedge \bigvee_{i=0}^{N} \Box \left(\bigwedge_{j=0}^{i} u_j \right)$$

$$\varphi_D(N) := c_0 \wedge \bigwedge_{i=0}^{N} \bigcirc[i](u_i \vee u_{i+1})$$