目标导向的答案集编程下事件演算中的类芝诺行为*

Ondˇrej Vaší˘cek[1] JoaquinArias[2] Jan Fiedor[1,6] GopalGupta[3]Brendan Hall[4] BohuslavKˇ rena[1] Brian Larson[5] Tomáš Vojnar[1,7][1]布尔诺理工大学，捷克 [2]CETINIA，胡安·卡洛斯国王大学，西班牙 [3]达拉斯德克萨斯大学，美国[4]Ardent Innovation Labs，美国 [5]Multitude Corp.，美国 [6]霍尼韦尔国际有限公司，捷克 [7]马萨里克大学，捷克

有人认为，事件演算（EC）适合用于对安全关键型信息物理系统的高层规范进行建模。其主要优势在于，EC模型与以半形式化自然语言表述的需求之间的语义鸿沟相当小。此外，EC对连续时间和变量的使用，避免了离散化带来的不精确性。此前，我们已证明，目标导向的ASP系统可用于实现这些EC模型。然而，将时间精确地表示为可无限细分的连续量，会导致这类系统出现类芝诺行为和非终止。在本研究中，我们对文献中一些广为人知的示例问题进行建模，系统地研究会引发这类芝诺行为的各种自然EC建模模式，并提出应对方法。此外，我们还提出了一种自动检测所有此类情况的技术。

1 引言与动机

由于需求工程在医疗、航空航天等关键领域有着广泛的应用，因此它是一个广阔的研究领域；在这些领域中，验证信息物理系统（CPS）能否正确运行至关重要。系统开发始于系统需求的规范，许多人认为这是开发过程中最关键的环节。AVSI SAVI项目 [19] 的研究结果清楚地表明了这一点：70%的错误是在系统规范和设计阶段引入的，但其中大多数直到后来对系统具体实现进行测试时才被发现。错误未能及早发现的主要原因是，系统规范描述的是一整套可行方案，最终才会从中选出一个具体方案来实现。与在早期阶段需要验证的、可能规模庞大的方案集合相比，验证单个方案要容易得多，主要是因为我们可以做出一些假设，例如假设CPS在离散时间下运行。在推理所有可行方案时，我们需要尽可能贴近现实地表示其行为，这包括对系统在连续时间和空间中的行为进行建模。然而，这会引发诸如芝诺行为之类的问题：理论上，无限多个事件可能发生在一个有限区间内。这种不切实际的行为使建模变得困难。

正如 [18]，所论证的，将系统需求转换为事件演算 (EC) [12]，并利用其语义进行验证与确认，是一种颇有前景的推理方式。事件演算尤其适合此用途，因为其语义符合人类理解需求的方式，使事件演算与需求之间的语义鸿沟几乎为零。这使事件演算能够忠实地推理需求所定义的行为，而不受设计或实现决策的影响，例如

*本研究得到了捷克科学基金会项目 23-06506S 和 FIT BUT 内部项目 FIT-S-23-8151 的支持；grantVAE(TED2021-131295B-C33) 由 MCIN/AEI/10.13039/501100011033 和“欧盟 NextGenerationEU/PRTR”资助，COSASS 项目 (PID2021-123673OB-C32) 由 MCIN/AEI/10.13039/501100011033 和“欧洲区域发展基金：建设欧洲之道”资助；此外还得到了美国国家科学基金会 IIS 1910131 号拨款以及通过达拉斯德克萨斯大学应用人工智能与机器学习中心获得的行业资助。

M. Gebser、D.Inclezan、F. Ricca、M. Carro 和 M. Truszczynski：第 41 届国际逻辑编程会议（ICLP 2025），EPTCS 439，2026 年，第 496–510页，DOI：10.4204/EPTCS.439.34

© O.Vaší˘cek 等。本作品依据以下许可协议授权：知识共享署名许可协议。

像基于自动机的方法那样“设计”状态、转移或子系统分解。该方法的能力已在火车道口控制系统中得到展示 [17] ，也体现在我们此前的工作中 [18], ：我们成功发现了若干不一致之处，以及一项安全关键医疗设备规格说明中的安全属性违例。上述两项工作均使用*s(CASP)*——一种用于带约束的答案集编程（ASP）的目标导向、免接地推理器 [4]。免接地特性和对约束求解的支持，使 s(CASP) 能够对连续时间和连续量进行推理；这两者对于准确、真实地表示信息物理系统的行为都至关重要。其目标导向特性还使 s(CASP) 能够提供可解释的结果；当使用自动推理的结果作为认证证据时，这一点至关重要，例如构建保证论证时 [13]。然而，与 Prolog 类似，s(CASP) 可能会出现非终止，尤其是在连续时间推理时；上述两项工作都曾在这方面遇到困难。

在本研究中，我们识别并系统地分类了若干类一般性的类芝诺行为，这些行为会导致连续时间下目标导向推理无法终止。我们实现了文献中的所有相关示例，包括水箱等相当复杂的示例 [2] 以及现实中的安全关键系统 [18]。我们展示了如何处理已识别的各类类芝诺行为，从而避免非终止问题。更确切地说，类芝诺行为是指使推理探索无限多个事件，且这些事件之间的时间间隔趋于无穷小，但这些无限多个事件并未真正发生，因此这种行为只是类似于芝诺行为。这类行为由常见的事件演算建模模式引起，例如防止重复触发；其中一些模式是目标导向推理所特有的，例如终止特定轨迹。我们识别出这些行为共同具有的特征，即一条芝诺式递降事件链（第3节），并提出一种在推理过程中检测该特征的技术（第3.2节）。我们通过文献中的示例或其变体，逐一介绍各类一般性的类芝诺行为（第4 节和第5节），并为每一类提出解决方案。随后，我们讨论水箱示例；该示例将所有这些行为作为子问题组合在一起，并且还呈现了真正的芝诺行为（第6节）。此示例展示了如何组合此前提出的所有解决方案；解决类芝诺子问题也能同时处理真正的芝诺行为。

2 背景与相关工作

埃利亚的芝诺提出了芝诺运动悖论：他的赛跑悖论（或运动二分悖论）指出，跑步者永远无法到达终点，因为他们必须先跑完一半的距离，再跑完剩余距离的一半，如此往复。芝诺执行或芝诺运行是定时／混合系统和自动机中的一个众所周知的问题。芝诺行为的一种常见定义是：无限多个事件或状态转换发生在有限时间区间内。芝诺行为不会出现在真实的物理系统中，而是抽象化的副产品。根据 [20], ，许多研究仅仅施加非芝诺假设，例如 [7]。处理芝诺行为的一种方法是通过正则化等技术修改系统，以消除这些行为 [9]。还有研究提出了检测芝诺行为的技术 [6, 14], ，或提出了芝诺行为存在（或不存在）的充分条件 [20, 10, 8]。然而，据我们所知，目前尚未对事件演算中的芝诺行为进行系统性研究。例如，在第 4 节中，我们展示了银行账户示例 [12] 在连续时间中从根本上无解，导致目标导向推理中出现类芝诺行为。

2.1 s(CASP) 与事件演算

s(CASP)系统 [4] 基于稳定模型语义 [5], 扩展了ASP系统的表达能力，加入了谓词、非基变量之间的约束、无解释函数，以及最重要的自顶向下、查询驱动的执行策略。这使得返回带有非基变量的答案成为可能，并且能够通过仅返回必要的片段来计算部分模型

```
流(灯_亮)。
事件(打开_灯_)。
事件(关闭_灯_)。
启动(打开_灯_亮, 灯_亮, T)。
终止(turn_light_off,light_on, T)。
% 自动创建的 can_* 规则
can_启动(turn_light_on, 轻型_开启, T) 。
can_终止(turn_light_off,light_on, T)。
```

(a) 领域模型（ex1/model.pl）。

```
最初为(灯_亮)。
发生(打开_灯_, 10). 发生(打开_灯_关闭, 20)。
?- 在时刻成立(灯_亮, T). % expected: T #> 10,T #=< 20
```

(b) 叙事及具有预期答案的查询。

```
在时刻成立(流, T2) :-
    T1 #> 0, T1 #< T2,
    可以_启动(事件, 流, T1),
    发生(事件, T1),
    启动(事件, 流, T1),
    非_停止成立(T1, 流, T2)。

停止成立(T1, 流, T2) :-
    T1 #< T, T #< T2,
    可以_终止(事件, 流) ,
    发生(事件, T),
    终止(事件, 流变, T)。
非_停止成立(T1, 流变, T) :- 。 . .
```

(c) 位于 axioms/bec_scasp-small.pl的事件演算公理。

图 1：示例 1 (灯的开/关) 的 s(CASP) 编码。

一个稳定模型来支持该答案，其中可以包含完整的证明树，从而使答案完全可解释。在 s(CASP)中，得益于其构造性否定，`not p(X)`可以返回`X` 的绑定，而调用`p(X)` 本来会失败。得益于s(CASP)与约束求解器的接口，可以结合约束进行可靠的非单调推理。与其他ASP实现一样，但不同于Prolog，s(CASP)能够处理非分层否定，并返回相应的（部分）稳定模型。例如，对于`p :- not q. q :- not p.`，在稳定模型语义下，该偶环有两个可能的模型，其中`p`或`q` 为真。

事件演算（EC）是一种用于推理事件和变化的形式化方法 [16, 12]，它有多个版本。我们使用基本事件演算（BEC） [15, 11]。EC 有三个基本概念：(i) 事件是可能在世界中发生的动作；(ii) 流是世界的一种随时间变化的属性；(iii) 时间点是一个时间瞬间。事件可能在某个时间点发生。流变在任何时间点都有真值，并且可能与数量相关联。它们通过事件或连续的轨迹离散地变化。一个示例的 EC 表述由全称理论、领域模型和叙事构成。理论是事件演算公理的合取，领域模型由该领域的因果律构成，而叙事则提供事件发生的观测

以及流变的性质。考虑以下来自 [12]的示例：

示例 1（灯的开/关）： 一个描述灯可以打开和关闭的模型。首先，我们定义领域模型（图 1a），其中：(i) 灯的状态由一个流表示（第 1 行）；(ii) 打开/关闭灯由事件表示（第 2 行）；(iii) 事件效应在第 3–4 行声明，s(CASP) 生成第 6–7 行（详见第 2.1.1节）。其次，我们定义一个叙事（图 1b）。灯最初处于关闭状态（第 1 行），并在时间 10/20 打开/关闭（第 2 行）。我们查询灯处于开启状态的时间点，答案是 10 到 20 之间。代码可在 ex1/model.pl。[1]

本例相关的事件演算公理如图所示。1c 其实现位于 axioms/bec_scasp-small.pl。谓词 `holdsAt(F, T2)` 在时间 `T2` 成立的条件是，某个事件在更早的某个时刻 发生，该时刻为 `T1`，且该事件的效果是在 时刻 `T1` 使该流成立，并且该流在 *T1*与`T2`之间未被停止。如果某个事件在一个时间区间内发生并终止某个流，则该流在该区间内被停止。后续示例中使用的全部 7 条 BEC 公理，实现在 axioms/bec_scasp.pl 中。谓词 `not_stoppedIn/3` 的实现将在第 2.1.1 节讨论。

2.1.1 在 s(CASP) 中对事件演算公理建模

s(CASP) 能够对连续时间中的事件演算进行建模，主要得益于两个关键因素：执行过程中保留非基变量，以及与约束求解器集成。使用转换

[1] 所有文件均可在此处获取：https://github.com/ovasicek/iclp25-scasp-zeno/tree/master/examples

DOI 10.5281/zenodo.15737913

规则，可将 [3], BEC 公理转换为遵循逻辑编程约定的 s(CASP) 程序。然而，必须特别注意避免非终止。

首先，必须考虑happens/2 与 initiates/3 的相对顺序（terminates/3 与 releases/3也一样），它们在图 1c所示的 holdsAt/2公理中。当优先证明 happens/2时，推理可能会经过一个开销很大（或无法终止）的证明，确认某个事件发生之后，才发现它对该流没有任何效应。然而，优先证明 initiates/3，可能会导致推理尝试证明无限多个事件的效应，却不考虑这些事件是否发生[2]。在 [18], 中，我们提出通过预处理来定义谓词 can_initiates/3、can_terminates/3、can_releases/3和can_trajectory/4；使用 s(CASP) 的 --ec 命令行选项时，会自动执行此预处理。对于每个事实和/或规则initiates(E,F,T):- some_body ，都会定义一个事实 can_initiates(E,F,T)（其他谓词也同样处理），且不会产生重复项。也可以手动定义自定义规则。事实证明，这种方法在剪枝搜索空间的同时，能有效避免非终止。

其次，s(CASP) 通过生成not 关键字的对偶规则来实现该功能。生成过程不了解谓词的语义，因此会生成超出所需数量的规则，其中包括可能探索连续时间参数不现实的值组合的规则。我们使用谓词的自定义实现not_stoppedIn(T1, F, T2) 和not_startedIn/3，以避免非终止。该自定义实现（位于axioms/bec_scasp.pl第 169 行）利用谓词的语义提高效率，并防止非终止。

3 s(CASP) 中的类芝诺行为：芝诺式递降事件链

在我们的实验中，非终止的主要来源是触发事件。如果某个事件的发生可由领域模型中的规则推出，则该事件会被触发。在连续时间中，这类触发规则可能导致无限次发生，从而产生需要推理的无限次变化。

我们遇到过许多示例，它们在发生非终止时的推理树中呈现相同的模式。推理会通过事件触发规则陷入循环，其中每次迭代都包含一个happens(Event, TimeN) 谓词，其使用了一个新的时间变量；该变量严格小于前一次迭代中的时间变量，但除此之外，两者都受到同一区间的约束。因此，该树包含如下链：happens(e,T1)T2 < T1happens(e,T2)，...，TN <... < T2 < T1，happens(e,TN)。我们将这样的序列称为芝诺式递降事件链（或芝诺链）。一个关键方面是，各个T之间不存在时间差，因此时间点之间的距离可能是无穷小的。带有芝诺链的推理会导致在有限时间区间内探索无限次事件发生，这与芝诺行为的定义相符。然而，正如我们在全文中所展示的，即使在仅发生有限次事件的示例中，目标导向推理也可能遇到芝诺链。因此，并非所有芝诺链都表示芝诺行为，而只表示类芝诺行为。

在第3.2节中，我们提出一种自动检测芝诺链的技术，使我们能够提醒用户注意其存在并停止非终止推理。

3.1 芝诺式递降事件链示例

示例 2（简化银行账户）：一个简化版本的银行账户问题 [12] 描述单个银行账户及其取款操作。如果余额低于 1000 的最低限额，则会通过触发事件收取 10 的服务费。在此版本中，每个叙事中服务费只能收取一次（而不是每月一次）。图2 展示了该示例的 s(CASP) 编码，以及一个包含若干查询及其预期答案的示例叙事。代码位于ex2/model.pl。

[2]此类非 终止现象见于 ex1/logs/non_term_summary.log，该示例使用了轻量示例的修改版本 .

流(balance(M))。流(尚未收取服务费)。事件(取款(M))。事件(服务费)。3 4 启动(取款(X)，余额(NewB)，T) :- 5 NewB #=OldB - X，6 在时刻成立(余额(OldB)，T)。7 终止(取款(_)，余额(OldB)，T) :- 8 在时刻成立(余额(OldB)，T)。9 10 终止(服务费，尚未收取服务费，T)。11 启动(服务费，余额(NewB)，T) :- 12 NewB #=OldB - 10，% 服务费为 10 13 在时刻成立(余额(OldB)，T)。14 终止(服务费，余额(OldB)，T) :- 15 在时刻成立(余额(OldB)，T)。（a）领域模型（ex2/model.pl）。

```
发生(服务费, T):-
  在时刻成立(尚未收取服务费, T),
  B #< 1000,   % 最低余额为 1000
  在时刻成立(余额(B), T).
```

(b) 有问题的事件触发规则。

```
初始为(balance(10000)).
初始为(noServiceFeeYet).
发生(withdraw(8000),10).
发生(withdraw(1500),20).

?- 在时刻成立(balance(X), 5).   % 10000
?- 在时刻成立(余额(X), 15).  % 2000
?- 在时刻成立(余额(X), 25).  % 490
?- 发生(服务费,              T).   % 最小的 T > 20
```

(c) 具有预期答案的叙事和查询。

图 2：示例的 s(CASP) 编码2 （简化银行账户）。

1 发生(serviceFee, {T1~[T1 > 0, T1 =< 100]}) % T1 2 在时刻成立 (noServiceFeeYet, {T1~[T1 > 0, T1 =< 100]}) 3 初始为(noServiceFeeYet) 4 not_停止成立(0,noServiceFeeYet, {T1~[T1 > 0, T1 =< 100]}) 5 {T2~[T2 > 0, T2 < 100]} #< {T1~[T1 > 0, T1 =< 100]} % T2 < T1 6 发生 (serviceFee, {T2~[T2 > 0, T2 < 100]}) % T2 7 在时刻成立 (noServiceFeeYet, {T2~[T2 > 0, T2 < 100]}) 8 初始为 (noServiceFeeYet) 9 not_停止成立(0,noServiceFeeYet, {T2~[T2 > 0, T2 < 100]}) 10 {T3~[T3 > 0, T3 < 100]} #< {T2~[T2 > 0, T2 < 100]} % T3 < T2 11 发生 (serviceFee, {T3~[T3 > 0, T3 < 100]}) % T3

图 3： 例 2 简化轨迹中的芝诺式递降事件链（完整内容见 2，位于 ex2/logs/non_term.log）

图 2c 中的所有查询都会因 serviceFee 的触发规则而终止（图 2b）。图 3展示了非终止推理树的简化快照。芝诺链从第 6 行开始，并在第 11 行继续。推理进入芝诺链，是因为要证明费用在某个T2时发生，就必须考虑它是否可能更早在T3时已经发生；但接着又必须考虑更早的T4时的情况，以此类推。详情和解决方案将在第 4节讨论。

在接下来的章节中，我们将介绍并描述针对一类特定问题的解决方案，这类问题会通过芝诺链导致非终止。每类问题都会通过文献中的一个示例或其变体加以介绍。表 1 汇总了所有示例，以及其他未编号的示例（可在代码仓库中名为 other的文件夹下找到），并列出每个示例中出现的问题组合（芝诺式或芝诺行为）。

3.2 芝诺递减事件链的自动检测

我们提出了一种简单而有效的技术来检测芝诺链。自版本 0.25.06.25 起，该技术已作为 s(CASP) 的一项功能提供，网址为 https://gitlab.software.imdea.org/ciao-lang/sCASP。可以使用 --zeno_halt 标志启用此功能。检测到芝诺链时，s(CASP) 会停止并发出警告，以提醒用户（避免非终止）。该技术的工作方式如下：展开谓词 happens(E,T) 时，s(CASP) 会检查当前推导，查找谓词 happens(E,T1) 和 happens(E,T2)，并满足以下条件：(i) 三者具有相同的事件 E，变量 T1 和 T2 被约束在同一区间 (low < Tx < high) 内；(ii) happens(E,T1) 的展开早于 happens(E,T2)；(iii) T2 严格小于 T1，即存在约束 T2 < T1。例如，图 3 中第 6 至 11 行所示的芝诺链将会被检测到（完整的检测轨迹见 ex2/logs/zeno_halt.log）。

我们通过表中列出的每个示例，验证了该技术快速检测类芝诺行为的能力 1 （其检测轨迹可在代码仓库中查看）。

表1：示例及其问题构成概述。

		重复的触发	轨迹相关问题			芝诺行为	
			自我终止	循环	不相等时。	平凡悖论	
灯的开/关	例 1						
简化银行账户示例	2	*					
银行账户		*					
光触发关闭		*					
渐暗的光	示例 3		*				
火车道闸			*				
下落物体			*				
注水容器			*				
脉冲光	例 4		*	*			
无损耗水箱			*	*			
无损耗弹跳球			*	*			
简化水箱	例 5	*	*		*		
闪烁灯	例 A1	*				*	
水箱	例 6	*	*	*	*		*
弹跳球			*	*			*

4 防止事件重复触发

事件演算中的一个常见概念是防止事件反复触发 [12]。然而，在连续时间中进行推理会带来一些问题。请看上一节中的例2 。如果满足以下两个条件，就应收取服务费：(i) 余额低于某个最低值；(ii) 尚未收取该费用。在时刻 `T1`取款后，对于任意满足以下条件的 在时刻成立 (ballance(X),T2) `T2` ，只要 `T2 > T1`（回想图 1c中的公理）。如果取款导致余额低于最低值，那么条件 (i) 对所有`T2`都成立（形成一个连续区间）。若没有条件 (ii)，服务费就会被无限次收取。有了条件 (ii)，第一次发生 会阻止之后的任何 发生。然而，考虑连续时间时会遇到一个问题：由于下界不包含在内，无法找到第一次发生；在本例中，这意味着找不到满足以下条件的 `T2` `T2 > T1`。因此，在连续时间下，图2c中的查询本质上无法回答；s(CASP) 试图寻找最小的`T2`时会导致非终止（回想图 3）。为解决这个问题，我们提出两种方案：

解决方案 1（添加 epsilon）：在时间上引入一个极小正数，并在延迟后收取服务费（见图4a）。这通过使用一个新的谓词`holdsAt(F, T, EPS, E)` 实现（其编码见图4b）。第三个参数使该谓词仅在时间`T = T1 + EPS` 时可被证明，其中`T1` 是第四个参数指定的事件`E`发生的时间点，而该事件会启动流`F`。第四个参数并非必需，但有助于避免因探索无关事件而导致速度变慢。此解决方案确保触发规则先证明`happens(withdraw(X), T1)` ，然后才在某个`T2 = T1 + EPS`时触发`serviceFee`。由于`withdraw(X)` 不是触发事件，因此需要探索的`T1`只有有限多个。因此，不会再因这条规则而出现非终止，查询也会终止（见ex2/logs/fix-holdsAt4.log）。在此处引入极小正数并不等同于离散化，因为T1 和`T2` 仍然可以取连续域中的任意值（且不会影响推理时间）。

解决方案 2（重新建模行为）：考虑到连续时间，我们对示例进行重新建模，使费用与取款同时收取，并组合两个事件的效应（见图 4c）。费用在原先的下界时刻收取，而不是无限接近该时刻时收取，且查询会终止（见ex2/logs/fix-remodel.log）。在此示例中，服务费的效应是通过合并费用扣除的金额与导致该费用的事件，来发起新的余额。若费用在同一时刻收取，取款(X) 事件不会产生任何效应。此解决方案最接近原始行为，但仅适用于事件效应能够组合的示例。

```
发生(服务费, T2) :-
    精度 #= 1/1000000,        余额 #< 1000,
    在时刻成立(余额(B), T2, 精度, 取款(_)),
    在时刻成立(尚未收取服务费, T2).
```

(a) 解 1：添加极小正数（ex2/fix-holdsAt4.pl）。

```
在时刻成立(流, T2, 持续时间, 事件) :-
    T2 #=T1 + 持续时间, ..., 发生(事件, T1),
    启动(事件, 流, T1)              , ...
```

(b) 新公理（axioms/bec_scasp.pl，第 118 行）。

```
发生(服务费, T) :- 发生(取款(Amount), T),
    在时刻成立(尚未收取服务费, T)     ,   NewB #< 1000,
    NewB #=OldB - Amount, 在时刻成立(balance(OldB), T).
启动(取款(X), balance(NewB)          , T) :-
    not_发生(服务费, T), ...
终止(取款(_), 余额(OldB), T) :-
    非_发生(服务费, T), ...
启动(服务费, 余额(NewB), T) :-
    发生(取款(X), T),
    NewB#= (OldB - X) - 10, 在时刻成立(余额(OldB)     , T).
```

(c) 解法 2：重塑行为 (ex2/fix-remodel.pl)。

图 4：示例 2 中问题的两种解法（简化银行账户）。

使用上述解决方案，我们对其他具有相同问题的示例进行了建模：（i）**完整银行账户** 模型涵盖多个账户和交易，并为服务费添加了每月重置机制（编码位于 other/1-bank_账户）。(ii) **光触发关闭** 是例 1 的一个版本，当灯亮起时会自动关闭（编码位于 other/2-light_触发_关闭）。

5 轨迹相关问题

事件演算通过轨迹表示连续变化。轨迹通过以下方式定义：`trajectory(F1,T1,F2,T2):-body`，当某个事件在`T1`启动其`F1`控制流时，轨迹便开始，并持续活动，直到该流被终止。轨迹处于活动状态时，可用于确定连续流`F2`在任意`T2`时的值。下面我们将讨论遇到的三个与轨迹相关的非终止问题。

5.1 自终止轨迹问题

自动终止是一种常见的建模模式：轨迹根据其定义的连续值，通过触发事件来结束。我们将这类轨迹称为自我终止轨迹。在 s(CASP) 中对此进行建模可能导致非终止。请看以下示例：

示例 3（渐暗的光，续自示例 1）： 灯打开时，其亮度从 0 逐渐增强，直至达到最大亮度 10。流`brightness(X)`的连续变化由一个由以下流控制的轨迹建模：`fading_in`，并通过触发事件`fade_in_end`自动结束。当达到最大亮度时，该事件触发，并启动`brightness(10)`作为该流新的离散值。图5 展示了 s(CASP) 编码中的新增部分（与示例 1相比）以及包含若干查询的示例叙事。代码见ex3/model.pl。

图 5c 中的所有查询都会陷入循环而无法终止，在触发规则（图 5a，第 13 行）和轨迹公理（图 5b，第 1–4 行）之间反复循环。芝诺链如下：happens(fade_in_end,T1), holds-At(brightness(10),T1), T2<T1, can_initiates(fade_in_end,brightness(10),T2), happens(fa- de_in_end,T2)（见 ex3/logs/non_term_summary.log）。可以自动检测到这条链（见 ex3/logs/zeno_halt.log）。问题在于，事件的效果被当作证明其自身触发条件的一种方式。由于该示例本身并不包含任何芝诺（类）行为，这是目标导向推理的一项局限。我们提出一种解决方案：

解 1（限制 HoldsAt）：在推理轨迹何时结束时，我们假设该轨迹处于活动状态；否则就没有必要推理它的结束时间。我们限制`holdsAt(brightness(10),T)`只能通过由流`fading_`控制的轨迹来证明（见图 6a），并使用一个新的谓词`holdsAt(F,T,CF)`（图 6b），该谓词在证明某个流的值时限制控制流变项的选择。这避免了原有的循环，查询也随之终止（见ex3/logs/fix-holdsAt3.log）。

```
事件(渐_入_结束)。      流(灯光_打开)。
流(亮度(X))。                流(渐隐_入)。
启动(打开_灯光_，渐隐_入，T)。
释放(打开_灯_时，亮度(X)，T)。
终止(渐_入_结束，渐_入，T)。
启动(渐_入_结束，亮度(10)，T)。
终止(渐_入_结束，亮度(X)，T):- X #≠ 10。

轨迹(渐_入，T1，亮度(NewB)，T2) :-
    NewB #=OldB  + ((T2-T1) * 1)    ，
    在时刻成立(亮度(OldB), T1)。

发生(fade_in_end，T) :- 在时刻成立(亮度(10), T)。
```

(a) 领域模型新增内容（ex3/model.pl）。

```
在时刻成立(流 ，T2):-  T1 #> 0，  T1 #< T2  ，
  可以_轨迹(流, T1, 流, T2),
  可以_启动(事件, 流, T1)              ，
  发生(事件, T1),
  启动(事件, 流, T1),
  轨迹(流, T1, 流, T2),
  非_停止成立(T1, 流, T2)。
```

（b）轨迹公理（公理/bec_scasp.pl）。

```
初始为(亮度(0))。
发生(turn_light_on,       10).
?- 在时刻成立(亮度(X), 25)。 % 10
?- 发生(fade_in_end,      T).  % 20
```

(c) 叙事和一些查询。

图 5：示例的 s(CASP) 编码 3 （渐暗的光），示例的扩展 1 （灯的开/关）

```
发生（渐暗_开始_结束，T）:-
  在时刻成立(亮度(10), T, 渐暗_光)。
```

(a) 解：限制 HoldsAt（ex3/fix-holdsAt3.pl）。

```
在时刻成立(流, T2, 流) :- ...,
  轨迹(流, T1, 流, T2)，  .....
```

(b) 新公理（axioms/bec_scasp.pl，第 51 行）。

图 6：示例 3 中问题的解 （渐暗的光）。

本文其余部分中所有轨迹相关示例都存在上述问题。本节提出的解法默认应用于所有这些示例。此外，我们还建模了以下存在相同问题的示例：（i）**注水容器**[16]，它注满至最高水位后会停止注水并开始溢出（编码位于other/5-filling_vessel/）。（ii）**火车道闸**控制器 [1]，该控制器对一个铁路道口进行建模，并描述降低／抬起道口栏杆的轨迹（编码位于other/3-train_gate/）。（iii）**下落物体**示例 [12]，该示例对一个从某一高度落下并落到地面的物体进行建模（编码位于other/4-falling_object/）。

5.2 循环轨迹问题

本节介绍一个轨迹之间存在循环依赖 的问题——一个轨迹的结束会启动另一个轨迹，反之亦然。以下示例本身并不包含任何 Zeno（类）行为；然而，它对连续时间中的目标导向自顶向下推理构成了一个问题。

示例 4（脉冲光，续例 3）： 光会反复渐亮和渐暗。一旦光达到最大亮度，就会开始渐暗，直至亮度归零。一旦亮度归零，光就会再次开始渐亮，如此反复。图 7 展示了 s(CASP) 编码（与例 3相比）的修改，以及包含若干查询的叙事。代码可在 ex4/model.pl 中找到。

图 7b 中的所有查询都会陷入非终止。要证明某个轨迹在 T2 时产生的效果，我们首先需要找到它的开始时间 T1（见图 5b 中的公理）。然而，由于循环性，推理将按以下方式进行：(i) 尝试证明轨迹 A 在 TA2 时产生的效果；(ii) 需要证明其开始时间 TA1，使得 TA1 < TA2；(iii) 轨迹 A 可以通过较早轨迹 B 的结束而开始；(iv) 需要证明轨迹 B 在 TB2 时产生的效果，使得 TB 2 = TA1，这一步相当于 (i)，其中 TB2 < TA2。这个循环包含轨迹公理（图 5b 第 1–4 行）以及轨迹开始/结束事件的触发规则（图 7a 第 6–7 行，图 6a 第 1–2 行）。芝诺链如下所示：happens(fade_out_end,T1), holdsAt(brightness(0),T1), T2<T1, happens(fade_in_end,T2), holdsAt(brightness(10),T2), happens(fade_out_end,T3)（见 ex4/logs/non_term_summary.log）；并且可以自动检测到它（见 ex4/logs/zeno_halt.log）。问题在于，推理永远无法到达 trajectory/4 谓词的主体，这意味着系统不知道该轨迹的潜在持续时间；也就是说，推理正在探索无限多个无限短的轨迹。我们针对这个问题提出两种解决方案：

```
流(逐渐消退_)。                    事件(消退_完成_结束)。
终止(消退_)_结束，逐渐消退_，T)。
启动(消退_完成_结束，逐渐淡入_，T)。
轨迹(逐渐消退_，T1，亮度(NewB)，T2) :-
    新亮度 #=OldB - (T2-T1)，在时刻成立(亮度(OldB), T1)。
发生(淡出_出_结束, T) :-
    在时刻成立(亮度(0), T, 正在淡出_出)。
```

(a) 编码补充 (ex4/model.pl)。

```
初始为(亮度(0))。
发生(打开_灯_开,          10).

?- 发生(渐暗_入_结束,     T). % 20
                              % 和 40
?- 发生(渐暗_出_结束，T)。 % 30
```

(b) 叙事和一些查询。

图 7：s(CASP) 对示例 4 (脉冲光)的编码，是示例 3 (渐暗的光)的扩展。

```
发生(淡入_中_结束, T) :-        持续时间 #= 10,
    在时刻成立(亮度(10), T, 淡入_中, 持续时间)。
发生(淡出_中_结束, T) :-        持续时间 #= 10,
    在时刻成立(亮度(0), T, 淡出_中, 持续时间)。
```

（a）解 1：添加持续时间（ex4/fix-holdsAt4.pl）。

```
在时刻成立(流, T2, 流, 持续时间) :-
    T2 #=T1 + 持续时间, ...
    轨迹(流, T1, 流, T2), ...
```

（b）新公理（axioms/bec_scasp.pl，第 65 行）。

```
递增_事件（淡出 _in_结束）。
增加_事件（淡出_淡入_结束）。

可以_启动（淡入_结束，淡出_，持续时间）:-
    增加_发生（淡入_结束，持续时间）。

可以_启动（淡出_淡入_结束，淡入_，持续时间）:-
    增量式_发生(淡出_结束_, T)。
```

(c) 解法 2：增量推理（ex4/fix-incr.pl）。

图 8：示例 4 （脉冲光）问题的两种解法。

解 1（添加持续时间）：在事件触发规则中引入持续时间信息（见图 8a），使用一个新谓词 `holdsAt(F1,T,F2,Duration)`（见图 8b），该谓词基于示例 3 中的 holdsAt/3 谓词，并像示例 2 中的 holdsAt/4 谓词一样扩展了持续时间。它指定一个轨迹控制流`F2` 以及一个 持续时间，在此期间该流必须成立。这样便将轨迹的持续时间引入循环中，查询也就能够终止（见 ex4/logs/fix-holdsAt4.log）。这种方法适用于轨迹持续时间始终相同的情况；对于持续时间可变的情况，可以指定最小持续时间，或采用下一个解决方案。

解法 2（增量推理）：对轨迹采用正向推理，并能准确处理持续时间可变的轨迹。推理从第一条轨迹开始。一旦证明它已经结束，我们就可以开始推理它是否导致了某条第二轨迹的开始，依此类推。这是通过获取轨迹结束事件（图8c，第 1–2 行），并使用`增量式_happens(E, T)` 将其除终止轨迹以外的所有效应分离出来实现的（图 8c，第 4–8 行）。推理会反复执行查询，以确定是否有任何事件发生。如果在`E1` 处发现事件发生于`T1`，则会向知识库添加事实`增量式_happens(E1, T1)`。接下来的一轮增量推理将查询时间大于`T1`的事件。如果一轮增量推理中发现多个事件发生，则只保存最早发生的事件，因为它们可能会影响下一轮中较晚事件的发生。重复此过程，直到不再发现事件或达到时间限制。然后执行原始查询（参见 ex4/logs/fix-incr.log），并使用包含所有增量事件发生记录的知识库。此方法在 s(CASP) 中实现，可通过`--incremental` 命令行选项启用。

基于上述内容，我们又建模了两个示例：(i) **无损耗弹跳球** 是一个简化的弹跳球，每次弹跳都会回到原来的高度（编码见other/7-no_loss_bouncing_ball）。(ii) **无损耗水箱** 对两个缓慢排水的水箱进行建模，其中一个水箱会由进水臂轮流[57]补水，同时只考虑组合排水速率与输入速率相同的叙事（编码见 other/6-no_loss_water_tanks）。

5.3 基于不等式结束条件的自结束轨迹

在前面的轨迹示例中，我们考虑了结束事件由等式触发的轨迹，例如，当亮度等于 10 时，灯光停止渐亮。对于严格单调的

```
流(水_左侧(X))。
流(左侧_灌注)。          流(左侧_排水)。
事件(切换_左侧)。      事件(切换_右侧)。

启动(switch_左侧, 左侧_filling, T).
终止(switch_左侧, 左侧_draining, T).
终止(switch_右侧, 左侧_filling, T).
启动(switch_右侧, 左侧_draining, T).

轨迹(左侧_filling, T1,water_左侧(NewW), T2) :-
   新水位 #=旧水位 + ((T2-T1) * 10), % 速率 30 - 20
   在时刻成立(水_左侧(OldW), T1)。
轨迹(左侧_排水, T1, 水_左侧(NewW), T2) :-
   NewW #=OldW - ((T2-T1) * 20), % 流出速率 20
   在时刻成立(水_左侧(OldW)   , T1).

发生(开关_左侧, T):- W #=< 50, % 目标水位 50
   在时刻成立(水_左侧(W), T, 左侧_排水)。
```

（a）部分领域模型（完整版本见 ex5/model.pl）。

```
初始为（water_左侧（100））。
发生（开始（右侧）,        10).
发生（switch_右侧,         16.25）。

? - 发生（switch_左侧,     T）。 % 12.5,18.125
?- 在时刻成立(water_左侧(X), 12.5).      % 50
?- 在时刻成立(water_左侧(X), 16.25). % 87.5
?- 在时刻成立(water_左侧(X), 18.125).% 50
?- 在时刻成立(water_左侧(X)  , 19.5)。    % 50.625
```

(b) 工作叙事 (ex5/model-nar1.pl).

```
初始为(water_左侧(0)).
发生(开始(左侧),           10).
发生(switch_右侧,          13).

?- 在时刻成立(water_左侧(X)  , 13)。% 30（终止）
?- 发生(开关_左侧, T)。          % 最小 T>13
?- 在时刻成立(水_左侧(X), 15)。% 50
```

(c) 非终止叙事 (ex5/model-nar2.pl).

图9：s(CASP) 示例编码5（简化水箱）。

```
发生(切换_左侧, T) :- W #= 50,          % = 目标水位
   在时刻成立(水_左侧(W), T, 左侧_排水).
发生(切换_左侧, T):-                持续时间 #= 1,
   W #< 50,    % < 目标水位
   在时刻成立(water_左侧(W), T, 左侧_draining, 持续时间)。
```

(a) 解 1：添加持续时间 (ex5/fix-split_holdsAt4.pl)。

```
终止(switch_右侧, 左侧_filling, T) :-
   X #> 50, % > 目标水位 50
   在时刻成立(water_左侧(X), T, 左侧_filling)。
启动(切换_右侧、左侧_排水, T) :-
   X #> 50, % > 目标水位 50
   在时刻成立(水_左侧(X), T, 左侧_填充)。
```

(b) 解法 2：重塑行为 (ex5/fix-split_否_start.pl)。

图 10：示例 5 (简化水箱)。

轨迹，亮度只会在一个时间点恰好为 10，因此结束事件只能发生一次。然而，有时可能需要使用带不等式的结束条件。在这种情况下，我们可能会遇到例 2 中的重复触发问题：某个事件被定义为在一个非包含边界附近无限接近地发生。请看以下示例：

示例 5（简化水箱）： 著名的两个漏水水箱示例的简化版本 [2], 其中，两个水箱共用一个输入臂，该输入臂会在两者之间来回切换，试图使水位保持在目标水位以上。我们的简化版本仅对左侧水箱进行建模，所有切换到右侧水箱的情况都仅作为叙事中的事实。左侧水箱有一条以排水速率（20 个单位）排水的轨迹，以及另一条通过结合输入速率（30 个单位）和排水速率实现注水的轨迹。触发事件切换_左侧会使输入臂切换到左侧水箱。当水箱正在排水且当前水位等于或低于目标水位（50 个单位）时，该事件会被触发——排水轨迹具有一个带不等式的结束条件。图 9 展示了此示例的大部分 s(CASP) 编码，以及两个包含若干查询的示例叙事。代码可在以下位置获取：ex5/model.pl。

第一个叙事（图 9b）中的查询均会终止。[3] 然而，第二个叙事（图 9c）中的第二个和第三个查询不会终止。在第一个叙事中，机械臂一开始位于右侧水箱中，这意味着水箱最初从起始水位 100 开始排水，在时间 12.5 时降至目标水位 50，并触发一个 切换_左侧事件。在时间 16.25 时发生一个 切换_右侧 事件，此时水箱已经注水至 87.5（高于目标水位）。在第二个叙事中，水箱起初是空的，正在注水。在时间 13 时发生一个切换_右侧 事件，而左侧水箱的水位仅为 30，仍低于目标水位。注水轨迹在时间 13 时结束，但在某个T > 13时，会尽可能早地触发一个 切换_左侧 事件。然而，这会造成一个问题

[3]How更复杂的实现方式需要用来表示_停止成立/3 （位于公理/bec_scasp-interval_not.pl，第 19 行） 4).

，因为这会产生一条无限短的排水轨迹。由于试图寻找一个无限接近非包含下界的事件，推理无法终止（参见 ex5/logs/non_term_summary.log以及ex5/logs/zeno_halt.log中的检测结果）。在连续时间中无法找到这样的事件，因此不可能回答该查询。这是在轨迹语境下，例 2中的重复触发问题的一种表现。我们提出两种解决方案，与例 2中的方案类似。这两种方案都将推理分为两种情况：（a）轨迹开始时，其结束条件尚未满足（第一个叙事）；（b）轨迹开始时，其结束条件已经满足（第二个叙事）。在情况（a）中，轨迹先达到等式所要求的条件，并在不等式适用之前结束，因此不需要不等式。同样，在情况（b）中，等式所要求的条件永远不会达到（因为已经超过），所以也不需要等式。

解 1（添加持续时间）：案例 (a) 使用基于等式的触发条件；案例 (b) 则利用例 4 中的 `holdsAt/4` 谓词为轨迹引入最小持续时间（见图 4 （见图 10a）。固定持续时间可防止出现无限短的轨迹，从而消除非终止问题（见 ex5/logs/fix-split_holdsAt4.log）。本例中本就需要该持续时间，并且不会导致离散化时可能产生的不准确性。

解决方案 2（重新建模行为）：案例 (a) 同样使用基于等式的触发条件，但案例 (b) 采用了不同的方法。无限短的轨迹只会造成无限小的差异，因此不应开始。只需将 `switch_右侧`事件的所有效应限制在案例 (a) 中（见图 10b），即可实现这一点，查询也会终止（见 ex5/logs/fix-split_no_start.log）。

6 真正的芝诺行为

我们说，前面的示例包含类似芝诺的行为，因为它们并不完全符合芝诺行为的定义——推理过程考虑了在有限时间内发生无限多个事件，但这些事件从未需要实际发生。本节介绍水箱中的 [2] 真正的芝诺行为，其中由于事件之间的延迟不断缩短，会发生无限多个事件。此外，我们还建模了另外两个示例：(i)**弹跳球**[9]，它在每次弹跳时都会损失能量；(ii)**闪烁灯**，它以无穷小的延迟闪烁，展示了平凡的芝诺行为（详见A）。

示例 6（水箱，续示例5）：水箱的完整版本 [2]，，其中水箱和机械臂开关通过触发事件建模，这些事件基于对两个水位的推理。芝诺行为 [2, 20, 9] 表现为：输入流量高于每个水箱的输出流量，但低于两者组合方案的输出流量——机械臂能够注满每个水箱，但系统中的总水量却在减少。机械臂会瞬时切换，因此，当两个水箱都越来越接近其目标水位时，它会通过持续时间越来越短的切换，始终防止两个水箱的水位低于目标水位。图11 概述了 s(CASP) 编码（完整代码见ex6/model.pl）。

图 11b 中所示的查询不会终止（见 ex6/logs/non_term_summary.log），这是由于迄今为止我们所介绍的问题共同造成的：(a) 真正的芝诺行为，(b) 示例 2 中在轨迹内表现出的重复触发问题，(c) 示例 3中的自结束轨迹问题，(d) 示例 4中的循环轨迹问题，以及 (e) 示例 5中的不等式终止条件问题。除 (a) 外，所有子问题都会通过芝诺链导致非终止，因此可以被检测到（ex6/logs/zeno_halt.log）。与前两个示例一样，初始编码已经通过`holdsAt/3`谓词解决了问题 (c)。我们进一步提出三种解决方案：

部分解决方案 1（增量推理）：该解决方案针对由推理限制导致的问题 (d)，并终止于问题 (a)。我们通过增量推理解决问题 (d)（见图 12a）。增量事件是用于向左侧和右侧切换机械臂的事件。由于

```
流(右侧_填充)。 % 重命名流(左侧_排水)
流(水_右侧(X))。

轨迹(右侧_填充, T1, 水_右侧(NewW), T2) :-
    。 . . % 与水的轨迹类似_左侧
轨迹(右侧_填充, T1, 水_右侧(NewW), T2) :-
    。 ..... % 与水的轨迹类似_左侧

发生(切换_右侧, T):-
    当前水位 #=< 50,     % 目标水位
    在时刻成立(水_右侧(CurrW), T, 左侧_注水)。
```

(a) 领域模型补充（ex6/model.pl）。

```
初始为(water_左侧(100))。
初始为(water_右侧(100))。
发生(开始(右侧),                10).
?- T #=< 19.5,
    发生(switch_左侧, T)。 % 12.5,18.125
?- T #=< 19.5,
    发生(switch_右侧, T)。% 16.25,19.0625
?- 在时刻成立(water_左侧(X)    ,   12.5)。   % 50
?- 在时刻成立(water_右侧(X), 16.25)。 % 50
?- 在时刻成立(water_右侧(X), 19.5)。      % 54.375
?- 在时刻成立(水_左侧(X),          19.5)。   % 50.625
```

(b) 叙事和一些查询。

图 11：示例6 （水箱）的 s(CASP) 编码，示例5的完整版本。

```
incr_事件（开关_左侧）。 增加_事件（切换_右侧）。
可以_启动（切换_左侧，左侧_注水，T）:-
    增加_发生（切换_左侧，T）。
可以_启动（切换_右侧，右侧_注水，T）:-
    增加_发生（切换_右侧，T）。
?- !增加_最大_时间（19.5），.....
```

(a) 部分解决方案 1：增量式 (ex6/partfix-incr.pl).

```
发生(switch_右侧, T):-                    最小持续时间 #= 1,
    持续时间 #>=最小持续时间,    水位 #=< 50,    % 目标水位 50
    在时刻成立(水位_右侧(持续时间), T, 左侧_填充, 持续时间)。
% 与 happens(switch_left, T) :- ... 相同
```

(b) 解法 2：添加持续时间(ex6/fix-holdsAt4.pl)。

```
% 与解 1 相同
递增_事件(...)。
可以_启动(...) :- 递增_发生(...)。

% 与解 2 类似，但拆分触发规则
发生(switch_右侧, T):-                    最小持续时间 #= 1,
    持续时间 #>=最小持续时间,    水位 #= 50,    % 目标水位 50
    在时刻成立(water_右侧(水位), T, 左侧_填充, 持续时间)。
发生(switch_右侧, T):-                    持续时间 #= 1,
    水位 #< 50,    % 目标水位 50
    在时刻成立(water_右侧(水位), T, 左侧_填充, 持续时间)。
% 对 happens(switch_left, T) :- ... 也是如此
```

(c) 方案 3：组合方案（ex6/fix-incr_holdsAt4.pl）。

图 12：示例6（水箱）中的问题的三种解法。

芝诺行为，典型叙事不会遇到问题 (e) 和 (b)，因为水箱中的水位从不会低于目标水位（参见 ex6/logs/partfix-incr.log）。

解法 2（添加持续时间）：该解法处理问题 (b) 和 (e) 中的类芝诺行为，以及问题 (a) 中的芝诺行为。问题 (b) 是问题 (e) 的一部分，因此我们只需关注后者。我们通过使用 *holdsAt/4* 谓词为轨迹设定最小持续时间（见图 12b）来解决问题 (e)。这会在切换事件之间引入延迟，同时也解决了问题 (a)，凸显了芝诺行为与类芝诺行为的相似性。此外，通过设定最小持续时间来解决问题 (e)，也为问题 (d) 提供了另一种解法，这意味着不再需要增量推理来确保终止（见 ex6/logs/fix-holdsAt4.log）。然而，最小持续时间的值越小，对推理时间的负面影响越大，因为这会允许循环执行更多次。

方案 3（组合方案）：此方案结合了解 1 和解 2（见图12c和 ex6/logs/fix-incr_holdsAt4.log）。解 1 只是部分解决方案，而结合这两种方案能带来更好的性能（从 30 分钟缩短至 10 秒）。

请注意，其他芝诺行为可能还需要其他解决方案。例如，对于**弹跳球**的轨迹设定最小持续时间，会导致它穿过地板落下。在这种情况下，一个合适的解决方案是：如果最后一次弹跳的时间短于最小持续时间，就让它停止弹跳；或者在每次弹跳时引入固定的速度损失（在other/8-bouncing_球中建模）。

7 结论与总结

我们实现了一种新颖的技术，并将其作为 s(CASP) 的一项功能提供。该技术能够根据是否存在芝诺式递降事件链，自动检测类芝诺行为。我们还进一步识别并描述了一系列由类芝诺行为引起的非终止问题，这些问题可能出现在连续时间下常见的事件演算建模任务中，并提出了四种

表 2：所提出的解决方案及其对所述问题的适用性总结 s.

		重复触发	轨迹相关问题			芝诺行为	
			自我终止	循环	存在不等关系	平凡	悖论
自动检测	（第 3.2节）	✓	✓	✓	✓	✓	
限制 HoldsAt	（例 3、4、5、6）		✓				
增量推理	（例 4、6）			✓			
添加时间 ε／持续时间	（例 2、4、5、6、A1）	✓		✓	✓	✓	✓
重塑行为	(例 2，5)	✓			✓		✓

可用于解决这些问题而无需离散化的解决方案类型。我们认为，这一系列问题（及其解决方案）足以涵盖广泛的实际应用场景，例如安全关键系统 [18]。

如表 2所示，除了非平凡芝诺行为外，该检测技术适用于所有类别的问题，因为非平凡芝诺行为具有不断递减的事件链，其中事件之间的延迟越来越短（而非无穷小）。此外，该表还总结了所提出的解决方案、这些方案应用于哪些示例，以及每种方案解决的问题类别。限制 HoldsAt 方案利用示例的外部知识，通过限制搜索规则来选择谓词 `holdsAt`**增量推理**方案则通过正向推理来处理循环性；由于事件演算需要重建历史，这种方法尤其适用于事件演算。**添加时间 ε／持续时间** 方案在循环中引入时间极小正数，以取代无穷小时间步长，同时避免产生不准确的行为（离散化时便会出现这种情况）。**重塑行为** 方案则调整示例模型，使其能够在连续时间中进行推理，同时避免无穷小时间步长，并保留预期行为。

参考文献

[1] Rajeev Alur & David L. Dill (1994)：《定时自动机理论》。理论计算机科学 126，第183–235页，doi:10.1016/0304- 3975(94)90010-8。[2] Rajeev Alur & Thomas A. Henzinger (1997)：《定时系统与混合系统的模块化》。载于：并发理论国际会议，LNCS 1243，施普林格，第74–88页，doi:10.1007/3-540-63141-0_6。[3]Joaquín Arias、Manuel Carro、Zhuo Chen & Gopal Gupta (2022)：《使用目标导向约束答案集编程进行事件演算建模与推理》。TPLP 22(1)，第51–80页，doi:10.1017/S1471068421000156。[4] Joaquín Arias、Manuel Carro、Elmer Salazar、Kyle Marple & Gopal Gupta (2018)：《无基化的约束答案集编程》。TPLP 18(3-4)，第337–354页，doi:10.1017/S1471068418000285。[5] Michael Gelfond & Vladimir Lifschitz：《逻辑编程的稳定模型语义》。载于：国际逻辑编程会议，第1070–1080页，doi:10.2307/2275201。[6] Rodolfo Gómez & Howard Bowman (2007)：《定时自动机中芝诺运行的高效检测》。载于：FORMATS，LNCS 4763，施普林格，第195–210页，doi:10.1007/978-3-540-75454-1_15。[7] Thomas A. Henzinger (1996)：《混合自动机理论》。载于：逻辑与计算机科学国际研讨会，IEEE计算机学会，第278–292页，doi:10.1109/LICS.1996.561342。[8] Frédéric Herbreteau & B Srivathsan (2013)：《粗粒度抽象使芝诺行为难以检测》。计算机科学中的逻辑方法，第9卷，第1期：6，doi:10.2168/LMCS-9(1:6)2013。[9] Karl Henrik Johansson、Magnus Egerstedt、John Lygeros & Shankar Sastry (1999)：《芝诺混合自动机的正则化》。系统与控制快报 38(3)，第141–150页，doi:10.1016/S0167-6911(99)00059-6。[10] Andrew G. Lamperski & Aaron D. Ames (2013)：《芝诺稳定性的李雅普诺夫理论》。IEEE自动控制汇刊 58，第100–112页，doi:10.1109/TAC.2012.2208292。

[11] Erik T. Mueller (2008)：第17章《事件演算》。载于：《知识表示手册》，人工智能基础 3，爱思唯尔，第671–708页，doi:10.1016/S1574-6526(07)03017-9。[12] Erik T. Mueller (2014)：《常识推理：一种基于事件演算的方法》。Morgan Kaufmann，doi:10.1016/B978-0-12-801416-5.00002-4。[13]Anitha Murugesan、Isaac Hong Wong、Joaquín Arias & 等 (2024)：《使用目标导向ASP自动分析系统保障案例的语义》。TPLP 24(4)，第805–824页，doi:10.1017/S1471068424000425。[14] Jonas Rinast & Sibylle Schupp (2012)：《基于同步矩阵和两种数据变量启发式方法对UPPAAL网络中的芝诺运行进行静态检测》。载于：FORMATS，施普林格，柏林、海德堡，第220–235页，doi:10.1007/978-3-642-33365-1_16。[15] Murray Shanahan (1996)：《机器人学与常识信息情境》。载于：ECAI，Wiley，第684–688页。[16] Murray Shanahan (1997)：《解决框架问题：对常识惯性定律的数学探究》。麻省理工学院出版社。[17] Sarat Chandra Varanasi、Joaquín Arias、Elmer Salazar & 等 (2022)：《使用事件演算和s(CASP)对实时系统进行建模与验证》。载于：PADL，Springer Iternational，doi:10.1007/978-3- 030-94479-7 12。[18]Ondrej Vasicek、Joaquin Arias、Jan Fiedor & 等 (2024)：《使用事件演算和答案集编程对高级系统需求进行早期确认》。TPLP 24(4)，第844–862页，doi:10.1017/S1471068424000280。[19] Don Ward (2012)：《AVSI的系统架构虚拟集成计划：将SAVI推向发射台》。载于：第15届NDIA年度系统工程会议。可查阅：https://ndia.dtic.mil/wp-content/ uploads/2012/system/ttrack914871.pdf。[20] Jun Zhang、Karl Henrik Johansson & 等 (2001)：《芝诺混合系统》。IJRNC 11(5)，第435–451页，doi:10.1002/rnc.592。

一种平凡芝诺行为：零延迟循环事件

本附录介绍一个未能完全纳入论文的示例。该示例展示了平凡的芝诺行为，但这是一种在事件演算中很容易遇到的建模模式。在连续时间中进行推理时，很容易引入事件触发规则，导致有限时间区间内发生无限多个事件。事件没有持续时间，其效应是瞬时的，除非我们明确引入某种持续时间或延迟。请看以下示例：

示例 A1（闪烁灯，续前例1）： 开灯和关灯这两个事件都被改成了触发事件。触发规则规定，灯亮时应将其关闭，反之亦然。然而，该编码并未定义闪烁的频率。触发事件被定义为在灯亮起/熄灭时立即将其熄灭/点亮。图13 展示了该示例的编码补充（与示例1相比），以及包含若干查询的示例叙事。代码见apx1/model.pl。

图中的所有查询 13b 都会无法终止，而且预期答案也未知。这里有两个问题：(i) 两个重复触发问题实例（来自例 2），每个触发事件对应一个；(ii) 平凡芝诺行为，因为灯被定义为以无限快的速度闪烁。问题 (ii) 意味着，当灯闪烁时，我们无法判断它是亮着还是灭着。这个问题表现为两个触发事件之间存在循环依赖，且没有任何延迟（见apx1/logs/non_term_summary.log），并且可以自动检测到（见apx1/logs/zeno_halt.log）。

我们在图 14 中给出解决方案。为了解决问题 (i)，我们采用了例 2 中第一个解决方案的简化版本。第二个方案不适用于这里，因为开灯和关灯事件的效应相反。解决问题 (i) 也就解决了问题 (ii)。

解决方案（图14a）使用新的谓词为事件触发规则添加了延迟 `holdsAt/3`（图 14b），它与`holdsAt/4` 在示例2 中引入的谓词相同，只是没有第四个参数，因为这里不需要它（每条规则只需考虑一个事件）。

为了解决问题（ii），即芝诺行为，我们需要为闪烁灯定义一个闪烁频率。请注意，问题（i）的解决方案已经在事件之间引入了延迟，因此也解决了这个问题。这表明了芝诺行为与本文所介绍的类芝诺行为之间的相似性。

1 happens(turn_light_off, T) :- happens(turn_light_on, 10). 2 holdsAt(light_on, T). ?- holdsAt(light_on, 15). % ?（未知） 3 happens(turn_light_on, T) :- ?- happens(turn_light_off, T). % ?（未知） 4 not_holdsAt(light_on, T). ?- happens(turn_light_on, T). % 10 和 ?（未知） （a）领域模型补充内容（apx1/model.pl）。（b）叙事和答案未知的查询。图 13：基于示例 1（灯的开/关）的示例 A1（闪烁灯）的 s(CASP) 编码。

1 happens(turn_light_off, T) :- holdsAt(light_on, T, 1). 1 holdsAt(Fluent, T2, Dur) :- T2 #= T1 + Dur, 2 happens(turn_light_on, T) :- not_holdsAt(light_on, T, 1). 2 ..., happens(Event, T1), ... (a)Solution(apx1/fix-holdsAt3.pl). (b) 新公理 (axioms/bec_scasp.pl, 第 80 行)。

图 14：示例中问题的解A1 （闪烁灯）。