复杂事件识别中用于聚合的形式化查询语言与自动机模型

Pierre Bourhis[1] Cristian Riveros[2] Amaranta Salas[2]

[1]法国里尔大学，法国国家科学研究中心
pierre.bourhis@univ-lille.fr[2]智利天主教
大学′{cristian.riveros, afsalas}@uc.cl

arXiv:2601.00967v1 [cs.DB] 2 Jan 2026

摘要

复杂事件识别（CER）系统用于识别事件流中的复杂模式，例如股票市场、传感器网络及其他类似应用中的模式。这些模式中的一项重要任务是聚合，即使用代数函数（如最大值、求和或平均值等）将一组值汇总为单个值。尽管这项任务十分重要，复杂事件识别中的查询语言通常仅以受限的语法形式支持聚合，其语义通常也未明确定义。

在这项工作中，我们迈出了形式化复杂事件识别聚合查询语言的第一步。我们提出扩展复杂事件逻辑（CEL）——一种用于复杂事件识别的形式化查询语言——加入聚合运算。这项工作需要重新审视 CEL 的语义，并采用一种基于元组多重集而非位置集合的新语义。随后，我们介绍 CEL 的一种扩展，称为聚合复杂事件逻辑（ACEL），它为任意交换幺半群运算引入了聚合算子。该算子可以与原有的 CEL 算子自由组合，使用户能够定义复杂的查询和模式。我们展示了若干实际查询，证明 ACEL 能够自然地对其进行描述。在计算模型方面，我们提出了一种新颖的自动机模型，称为聚合复杂事件自动机（ACEA）；它在原有复杂事件自动机（CEA）的基础上扩展了聚合和过滤功能。此外，我们证明了 ACEL 中的每个查询都可以用 ACEA 表达，展示了该计算模型的有效性。最后，我们从 ACEL 的角度研究 ACEA 的表达能力，并表明该自动机模型的表达能力强于 ACEL。

1 引言

复杂事件识别（CER）系统是一类数据流管理系统，用于实时检测满足特定模式的特殊事件，这些事件称为复杂事件；检测时会考虑它们的位置、顺序以及彼此之间的其他约束 [17, 12]。其应用示例包括海事监控 [27], 网络入侵检测 [25], 工业控制系统 [21], 以及实时分析[32]。文献中，人们提出了多种系统和查询语言，采用不同形式化方法来处理复杂事件，例如基于自动机、基于逻辑、基于树的方法，或这些方法的组合 [17]。在学术界和工业界开发的复杂事件识别系统包括 SASE [36], EsperTech [1], 和 CORE [9, 8], 等。

复杂事件识别系统的一个问题是，用于声明复杂事件的查询语言在语法和语义方面都定义不足，这令人遗憾。正如先前研究所指出的 [38, 15, 11, 6], ，系统中的 CER 查询语言通常缺乏简单、组合式且具有指称性的语义。一般而言，其语义是通过示例间接定义的 [2, 10, 23], ，或通过转换为求值模型来定义 [26, 33, 35]。近年来，研究者对复杂事件识别系统中的这一问题进行了更深入的研究，而一种成功定义了若干 CER 运算符语义的查询语言是复杂事件逻辑(CEL) [19, 20], ，与之配套的计算模型称为复杂事件自动机(CEA)，该模型基于有限状态转换器和符号自动机理论。

形式化 CER 查询语言面临的一个未决问题是，实践中的若干有趣查询包含聚合，而此前的提案尚未涉及这一点。聚合指查询中任何将多个（通常为数值）值组合并合并为单个值的子过程 [18], ，例如求一组值的平均值、总和或最大值。使用聚合的复杂事件识别系统包括 SASE [14, 37], EsperTech [1], GLORIA [24], GRETA [30], 等 [17, 12]。为说明一种典型的聚合查询，下面给出 SASE 中的一个（简化）查询[37, 第3]页：

Q1：PATTERN seq(JobStart a, Mapper+ b[ ]，JobEnd c)
WHERE a.job_id = b[i].job_id and a.job_id = c.job_id
RETURN 平均值(b[ ].period), 最大值(b[ ].period)

直观地说，前述查询旨在从映射器运行时间列表中获取平均值和最大值。为此，它在“PATTERN”子句中查找流中与以下模式匹配的事件：一个 JobStart 有类型事件，后跟一个或多个 Mapper 事件，最后是一个 JobEnd 事件。接着，它使用“WHERE”子句确保与该模式匹配的每个事件具有相同的 id 属性，最后返回满足条件的事件的平均值和最大值。

以往形式化 CER 查询语言的方案不包含此类带聚合的查询，因为它们不仅必须检测和检索复杂事件，还必须生成新的事件和值。具体而言，聚合查询无法通过 CEL 等逻辑或 CEA 等计算模型来定义，也无法通过当前定义的任何其他 CER 形式化方法来定义。这些问题意味着，带聚合的 CER 查询语言难以比较，查询如何组合也不明确，并且难以求值（即，在不了解查询实际含义的情况下）。此外，用于编译带聚合查询的计算模型尚未得到充分理解，而系统通常依赖适用于特定查询和模式的临时求值策略。

在本文中，我们提出对逻辑 CEL 及其对应计算模型 CEA 的扩展，以表达带聚合的查询，并分别将其称为聚合复杂事件逻辑 (ACEL)和聚合复杂事件自动机 (ACEA)。我们的主要目标是设计一种逻辑及计算模型，并为其赋予形式语义，以形式化复杂事件识别（CER）中的聚合，同时为所有 CER 语言奠定基础。

要为 CEL 扩展聚合功能，我们需要重新审视其语义。当前 CEL 语义面临的首要问题之一是：CEL 公式会返回触发复杂事件的流中位置，却不允许创建新的值或事件。因此，我们提出一种新的、等价的 CEL 语义，使其返回事件而非位置。此外，由于聚合还需要保留重复项，我们将语义扩展为使用事件多重集而非集合。随后，我们证明新语义与先前的语义等价。有趣的是，新语义使我们能够为复杂事件识别定义新的相关算子，例如属性投影。

为了形式化复杂事件识别中的聚合，我们考虑一种基于聚合函数 [22, 18]的通用聚合设定；这类函数将值多重集映射为单个值，旨在概括信息（如计数、求和等）。通过采用这一通用的聚合函数框架，我们为复杂事件识别中的聚合提供了一个通用框架。此外，我们引入了 $\mathrm{Agg}^{Y(\mathbf{b}\leftarrow\otimes X(\mathbf{a}))}$ 算子，用于名称为 X 和 Y 的变量、名称为$\mathbf{a}$ 和$\mathbf{b}$的属性，以及聚合函数 $\otimes$。该算子对存储在 X 和 $\otimes$中的事件多重集的相应属性$\mathbf{a}$进行操作，并将结果存储在另一个变量 Y中某个事件的另一个属性$\mathbf{b}$ 中。我们在第5节中形式化地定义其语法和语义。该定义的一个优点是，我们可以将 Agg 算子与 CEL 中的其他算子组合使用。我们证明，以往工作中大多数包含聚合的 CER 查询都可以用 ACEL 表达。

CEL 的一个优势是，可以用所谓的复杂事件自动机（CEA）来刻画其表达能力；具体而言，对于每个 CEL 公式，都存在一个与之等价的 CEA，反之亦然。这一结果的实际意义在于，CEL 便于用户定义查询，而 CEA 则便于系统对查询进行求值。在这项工作中，我们旨在实现类似的等价结果，因此下一步是寻找一种能够扩展 CEA 并对 ACEL 进行形式化定义的机器模型。我们为 ACEL 引入了一种带聚合的自动机模型，称为聚合复杂事件自动机 （ACEA），它是 CEA 的扩展，带有用于聚合值的寄存器。这种扩展采用了与代价寄存器自动机 [4] 相同的概念，即在转移中操作值并维护寄存器。具体而言，在每次转移中，ACEA 都会读取一个事件，并用这些新值更新其寄存器。随后，它根据赋值执行操作，检查是否满足谓词，最后，

它会生成一个包含聚合值的新元组。我们的主要结果之一是，可以将每个 ACEL 公式编译为一个 ACEA；也就是说，我们可以证明 ACEL 的表达能力是 ACEA 表达能力的子集。

概要。我们在第 2 节介绍预备知识。在第3节中，我们讨论 CEL 语义所需的变更，并介绍一个新的算子: 按属性投影。在第 4 节中，我们讨论聚合函数的设置。在第5节中，我们正式引入 ACEL，并在第 6 节中引入 ACEA，以研究将 CEL 公式编译为 ACEA 的过程及其与复杂事件自动机的等价性。最后，我们在第 7 节总结并讨论未来工作。

2 预备知识

集合、区间与映射。给定一个集合 A，我们用 $\mathcal{P}(A)$ 表示 A 的所有有限子集构成的集合。我们用 N 表示自然数。给定 $n, m \in$ N 且 $n \leq m$，我们用 $[n]$ 表示集合 $\{1, \ldots, n\}$，用 $[n..m]$ 表示 N 上的区间 $\{n, n+1, \ldots, m\}$。按照惯例，我们写作 $f\colon A \to B$，表示一个函数f，其定义域为集合 A，值域为 B，且 A中的每个元素都有一个像。映射 M 是一个偏函数，它将 A 中的有限多个元素映射到 B中的元素。我们写作 $M\colon A \mapsto B$，表示一个映射M，从 A 映射到 B。我们用 dom(M) 表示 M 的定义域（即所有使得 $M(a)$ 有定义的 $a \in A$），用 img(M) 表示 M 的像。通常，我们使用记法 $[a_1 \mapsto b_1, \ldots, a_k \mapsto b_k]$来定义映射$M$，其 dom($M$) 为 $= \{a_1, \ldots, a_k\}$，img(M) 为 $= \{b_1, \ldots, b_k\}$，且对每个 $i \in [k]$，都有 $M(a_i) = b_i$。此外，对于映射 M及 $a \notin$dom(M)，我们写作 $[M, a \mapsto b]$ 来指定一个新的映射 M'，它将 a映射到 b，并扩展映射 M。

多重集。多重集或多重集（带有自身的幺元） B 是一个映射 $B : I \mapsto U$，其中 $I =$dom(B) 是标识符（或 id）的有限集，$U =$img(B) 是该多重集的底集。对于任意多重集 B，我们分别将这些分量称为 $I(B)$ 和 $U(B)$。例如，多重集 $B = \{\!\{a, a, b\}\!\}$（其中 a 重复两次）可以用映射 $B_0 = [1 \mapsto a, 2 \mapsto a, 3 \mapsto b]$ 表示，其中 $I(B_0) = \{1, 2, 3\}$ 且 $U(B_0) = \{a, b\}$。一般而言，我们将使用多重集的标准记法 $\{\!\{a_1, \ldots, a_n\}\!\}$，表示标识符为 $I(B) = \{1, \ldots, n\}$ 且对每个 $i \in I(B)$ 均有 $B(i) = a_i$ 的多重集 B。我们将用 $\uplus$ 表示多重集的并集。此外，我们将 $\mathcal{P}_{bags}(A)$ 定义为可以由集合A构成的所有有限多重集的集合。

计算模型。我们假设采用均匀成本度量的随机存取机（*RAM*）模型，并将加法和减法作为基本运算 [3]。例如，这意味着访问查找表（即按键索引的表）只需常数时间。这些是该领域文献中常见的假设 [9, 34]。

3 重新审视复杂事件逻辑的语义

本节中，我们重新审视 CEL [20] 的语义，并介绍一种基于元组而非位置的新语义。随后，我们证明这两个版本之间的等价性。例如，这种新语义允许为 CEL 定义一种称为属性投影的新算子，而旧语义无法定义这种算子。此外，在下一节中引入 CEL 的聚合时，新语义也至关重要。

事件与流。我们固定一个可数无限的属性名称集合 A，以及一个可数无限的数据值集合 D（例如整数、字符串）。一个（无类型）事件 e 是一个有序对（M，i），其中 M： A $\mapsto$ D 将 A 中的属性名称映射到 D 中的数据值，而 $i \in$N 是事件的时间[16]（我们倾向于使用离散时间，这已足以满足我们的需求）。直观地说，M 定义了事件的数据（即一个元组）。我们用 e(a) $\in$ D 表示属性 a $\in$ A 经由 M 赋予的值（即 e(a) $= M$(a)）。如果e 未定义属性 a 的值，则记作 e(a) =NULL。此外，为简化起见，我们也记作 e(time) $= i$（但请注意，time 并非属性）。我们将 Att(e) 定义为 e的属性集合，即 Att(e) $=$ dom(M)。我们用 E 表示由属性名称 A 和数据值 D 构成的所有事件的集合。通常，我们用粗体字母 a、b 和 c 表示 A 中的属性名称，用普通字母 a、b和 c 表示 D 中的数据值。

现在固定一个有限的事件类型$\mathbf{T}$，并假设$\mathbf{T} \subseteq \mathbf{D}$ 且 NULL $\in \mathbf{D}$。在本文中，我们假设存在一种特定属性类型，用于定义事件的类型。具体而言，

令 type 为一个满足 type $\in \mathbf{A}$ 的属性。对于每个事件 e，我们假设 type $\in$ Att(e) 且e(type) $\in \mathbf{T} \cup \{\text{NULL}\}$ 是 e 的类型。请注意， e 可以是有类型的（即， e(type) $\in \mathbf{T}$），也可以是无类型的，此时有 e(type) = NULL。模式 Σ 是一个函数 Σ:$\mathbf{T} \to \mathcal{P}(\mathbf{A}_\Sigma)$，其中 $\mathbf{A}_\Sigma \subseteq \mathbf{A}$ 是一个有限属性集。我们称事件 e 满足模式 Σ，当且仅当 e 是有类型事件，且 Att(e) $= \Sigma(e(\text{type})) \cup \{\text{type}\}$。特别地，根据定义，无类型事件不满足任何模式。

设 $\Sigma : \mathbf{T} \to \mathcal{P}(\mathbf{A}_\Sigma)$ 为一个模式。模式 Σ 上的流 是一个（长度任意的）有类型事件序列 $\mathcal{S} = e_1 e_2 \dots e_n$，满足对每个 $i \in [n]$，均有 e 满足 Σ，且 e_i(时间) $= i$。换言之，流由符合 Σ 的有类型事件组成，且每个事件的时间就是它在流中的位置。请注意，我们定义事件的类型和时间，是为了后续在 CEL 的语义中使用。前者使我们在将 CEL 编译为自动机模型时能够获知每个事件的属性，后者则使我们能够通过添加事件在流 [7]中的来源 来区分元组。

示例 3.1. 作为一个贯穿全文的示例，假设我们有一个流 $\mathcal{S}_{\text{Stocks}}$，它会发出特定股票的买入和卖出事件 [9]。这里，我们假定有一个模式 Σ_{Stocks}，其中包含名称和价格属性，分别表示股票的名称（例如英特尔的 INTL）及其价格（例如 80美元）。我们有两种类型，称为 BUY 和 SELL，并且 Σ_{Stocks}(BUY) $= \Sigma_{\text{Stocks}}$(SELL) $= \{$名称，价格$\}$。一种可能的流$\mathcal{S}_{\text{Stocks}}$ 如下：

	0	1	2	3	4	5	6	7	8	9
$\mathcal{S}_{\text{Stocks}}$:	SELL MSFT 101	SELL MSFT 102	SELL INTL 80	BUY INTL 80	SELL AMZN 1900	SELL INTL 81	BUY AMZN 1920	BUY MSFT 101	BUY INTL 79	SELL INTL 80

请注意，每个事件都包含一个类型（即，买入或卖出）、其属性值（即，名称和价格）以及时间（即，事件上方的位置）。此外，每个事件都满足 Σ_{Stocks}。

事件的重命名这一概念在本文中很有用（例如，参见第 6 节）。形式化地，我们将重命名 r 定义为映射 r : A $\mapsto$A。我们称事件 e 与重命名 r一致，当且仅当 r(a) $= r$(b) 时， e(a) $= e$(b)。给定一个与 r一致的事件 e，我们定义重命名后的事件 $r(e)$，使得 $[r(e)]$(时间) $= e$(时间)，并且对每个属性 a $\in$dom(r)，都有 $[r(e)](r(\text{a})) = e$(a)。换句话说，r 将 e 的每个属性 a 重命名为 r(a)。我们用 Ren 表示 A 上所有重命名的集合。

事件谓词。谓词 是一个可能无限的事件集合 P。例如， P 可以是满足条件 e(a) ≤ 20 的所有事件 e 的集合。在我们的例子中，我们将使用记法 a $\sim a$，其中 a $\in$**A**，$a \in$**D**，且 $\sim$ 是 **D** 上的二元关系，以表示谓词 $P = \{e \mid e(\text{a}) \sim a\}$。我们称事件 e满足谓词 P，记作 $e \models P$，当且仅当 $e \in P$。我们将此记法从事件推广到事件多重集 E： 当且仅当对于每个 $e \in E$， $E \models P$成立时， $e \models P$成立。

在本文中，我们假设有一个固定的谓词集合 **P** ，它对交集、否定和重命名封闭，即 $P_1 \cap P_2 \in \mathbf{P}$、$\mathbf{E} \setminus P \in \mathbf{P}$，并且对于每个 $P, P_1, P_2 \in \mathbf{P}$ 和 $r \in$Ren，$r(P) = \{r(e) \mid e \in P \wedge e$ 与 $r\}$ 一致；此外，所有事件的集合 **E** 是**P** 中的一个谓词，我们通常将其记作 TRUE。

复杂事件。在本文中，我们将采用略有不同的复杂事件定义：我们会将事件存储在赋值中，而不是像在 [9]中那样存储位置。形式上，固定一个由变量构成的有限集 X，其中包含所有事件类型（即 $\mathbf{T} \subseteq$ X）。设 $\mathcal{S}$ 为长度为 n的流。 $\mathcal{S}$上的一个复杂事件是一个三元组 (i, j, μ)，其中 $i, j \in [n]$, $i \leq j$，且 μ : X $\to \mathcal{P}_{bags}$(E) 是一个从变量映射到有限事件多重集的函数。直观地说， i 和 j 标记复杂事件发生区间的起点和终点，而 μ 则存储区间 $[i..j]$ 中触发该复杂事件的事件。下文中，我们通常用 C 表示 $\mathcal{S}$ 上的一个复杂事件 (i, j, μ)；如果上下文已明确流是什么，就省略 $\mathcal{S}$。我们将用 time(C)、start(C) 和 end(C) 分别表示 C的区间 $[i..j]$、起点 i和终点j 。此外，稍微滥用一下记号，我们也会用 $C(X)$表示 $X \in$X，以指代 C的多重集 $\mu(X)$。

下文将贯穿全文使用以下关于复杂事件的操作。我们将复杂事件 C_1 与 C_2的并集定义为 $C_1 \uplus C_2$，记作复杂事件 C'，使得 start(C') =min{start(C_1), start(C_2)}, end(C') =max{end(C_1), end(C_2)}，且对每个 $X \in$ X，均有 $C'(X) = C_1(X) \uplus C_2(X)$。此外，我们将复杂事件 C在 $L \subseteq$X 上的投影定义为 $\pi_L(C)$，记作复杂事件 C'，使得 time(C') =time(C)，并且只要 $X \in L$，就有 $C'(X) = C(X)$；否则， $C'(X) = \emptyset$。最后，我们用 $(i, j, \mu_\emptyset)$ 表示具有平凡函数 $\mu_\emptyset$ 的复杂事件，使得对每个 $X \in$X，均有$\mu_\emptyset(X) = \emptyset$ 。

$$[\![R]\!](\mathcal{S}) = \{(i,i,\mu) \mid i \in [k] \wedge e_i(\text{类型}) = R \wedge \mu(R) = \{\!\{e_i\}\!\} \wedge \forall Y \neq X.\, \mu(Y) = \emptyset\}[\![\varphi \text{ AS } X]\!]$$
$$(\mathcal{S}) = \{C \mid \exists C' \in [\![\varphi]\!](\mathcal{S}).\ \text{time}(C) = \text{time}(C') \wedge C(X) = \biguplus_Y C'(Y) \wedge \forall Z \neq X.\, C(Z) = C'(Z)\}$$
$$[\![\varphi \text{ FILTER } X[P]]\!](\mathcal{S}) = \{C \mid C \in [\![\varphi]\!](\mathcal{S}) \wedge C(X) \models P\}[\![\pi_L(\varphi)]\!](\mathcal{S}) = \{\pi_L(C) \mid C \in [\![\varphi]\!](\mathcal{S})\}$$
$$[\![\varphi_1 \text{ 或 } \varphi_2]\!](\mathcal{S}) = [\![\varphi_1]\!](\mathcal{S}) \cup [\![\varphi_2]\!](\mathcal{S})[\![\varphi_1 \text{ 与 } \varphi_2]\!](\mathcal{S}) = [\![\varphi_1]\!](\mathcal{S}) \cap [\![\varphi_2]\!](\mathcal{S})[\![\varphi_1 : \varphi_2]\!](\mathcal{S})$$
$$= \{C_1 \uplus C_2 \mid C_1 \in [\![\varphi_1]\!](\mathcal{S}) \wedge C_2 \in [\![\varphi_2]\!](\mathcal{S}) \wedge \text{end}(C_1) + 1 = \text{start}(C_2)\}[\![\varphi_1;\ \varphi_2]\!](\mathcal{S})$$
$$= \{C_1 \uplus C_2 \mid C_1 \in [\![\varphi_1]\!](\mathcal{S}) \wedge C_2 \in [\![\varphi_2]\!](\mathcal{S}) \wedge \text{end}(C_1) < \text{start}(C_2)\}[\![\varphi\oplus]\!](\mathcal{S}) = [\![\varphi]\!](\mathcal{S}) \cup [\![\varphi :$$
$$\varphi\oplus]\!](\mathcal{S})[\![\varphi+]\!](\mathcal{S}) = [\![\varphi]\!](\mathcal{S}) \cup [\![\varphi;\ \varphi+]\!](\mathcal{S})$$

图 1：定义在流 $\mathcal{S} = e_1 e_2 \ldots e_n$ 上的 CEL 语义，其中每个 e_i 都是一个事件。

CEL 的新语义。在本文中，我们使用 [20]中引入并在 CORE [9] 中实现的 复杂事件逻辑 （CEL）作为 CER 的基本查询语言。然而，为了扩展其聚合功能，我们重新审视了其语义。具体而言，我们采用与 [20] 中相同的 CEL 语法，其文法如下：

$\varphi :=$ R （事件类型选择）| φ AS X （变量绑定）| φ FILTER $X[P]$ （谓词过滤）| $\pi_L(\varphi)$ （变量投影）| φ OR φ （析取）| φ AND φ （合取）| $\varphi :\ \varphi$ （连续序列）| $\varphi;\ \varphi$ （非连续序列）| $\varphi\oplus$ （连续迭代）| $\varphi+$ （非连续迭代）

其中， R 是事件类型， $X \in \mathbf{X}$ 是变量， $P \in \mathbf{P}$ 是谓词， $L \subseteq \mathbf{X}$是变量的有限集。类似于 [20, 9],，我们递归地将流 $\mathcal{S} = e_1 e_2 \ldots e_n$上的 CEL 公式 φ的语义定义为关于 $\mathcal{S}$的一组复杂事件。主要区别在于复杂事件的概念：现在其中包含的是事件，而不是位置。在图 1 中，我们按照 [9, 20]中的方式定义每个 CEL 算子的语义。给定公式 φ，语义 $[\![\varphi]\!](S)$ 定义了一组复杂事件。请注意， $[\![\varphi]\!](S)$ 具有集合语义，而复杂事件则存储事件的多重集。

接下来，我们通过一个示例说明如何使用 CEL 的语法和语义从流中提取复杂事件（另见第5节）。在此示例中，我们在过滤条件中使用合取和析取，可以将其读作：

$$\varphi \text{ FILTER } (X[P_1] \wedge Y[P_2]) \equiv (\varphi \text{ FILTER } X[P_1]) \text{ FILTER } Y[P_2]$$
$$\varphi \text{ FILTER } (X[P_1] \vee Y[P_2]) \equiv (\varphi \text{ FILTER } X[P_1]) \text{ OR } (\varphi \text{ FILTER } Y[P_2])$$

对于每个 CEL 公式 φ、变量 X， $Y \in \mathbf{X}$ 和谓词 P_1， P_2。

示例3.2 （来自 [9]）。 考虑示例 3.1 中的流 $\mathcal{S}_{\text{Stocks}}$。假设我们对所有由三个卖出事件组成的序列感兴趣：第一个是微软售价超过 100 美元的销售，第二个是英特尔的销售（价格不限），第三个是亚马逊售价低于 2000 美元的销售。那么，我们可以用以下 CEL 公式指定此模式：

$\varphi_2 =$ (标记为卖出的 msft; 标记为卖出的 intel; 标记为卖出的 amzn) FILTER
([=] ∧ [> 100] ∧ [=] msft 名称“MSFT”
∧ =
msft 价格 intel 名称“INTC” amzn[名称“AMZN”] ∧ amzn[价格 < 2000]).

直观而言，表达式 (SELL AS msft ; SELL AS intel ; SELL AS amzn) 指定了我们希望看到三个 SELL 事件，并分别将它们命名为变量 msft、intel 和 amzn。分号算子 (;) 表示它们之间采用非连续序列，也就是说，它们之间可以出现其他事件。最后，过滤子句要求事件数据满足必要的限制。

如前所述，在本文中，我们将 [20, 9] 使用的语义改为使用事件而非位置，并在此称之为基于事件的 语义。CEL 的旧语义称为基于位置的 语义，它通过输出形式为 $C = (i, j, \mu_{\text{index}})$ 的复杂事件得到，其中 μ_{index} 是一个映射，满足 $\mu_{\text{index}} : \mathbf{X} \mapsto \mathcal{P}(\mathbb{N})$。也就是说， μ_{index} 包含参与构成 C 的事件的位置。容易看出，CEL 的基于事件的语义等价于基于位置的语义，只需将其中的 i 位置替换为流中的第 i 个事件即可。换言之，我们有如下等价关系。

定理 3.1。 CEL 的（旧）基于位置的语义等价于 CEL 的（新）基于事件的语义。

一种用于投影属性的新算子。提供新的基于事件的语义的一个优势在于，可以使用新的算子扩展 CEL，例如聚合算子，我们将在后续章节中讨论这一点。更有趣的是，我们可以引入用于管理复杂事件的新型自然算子，而这些算子无法使用 [20] 中的旧语义定义。在本文中，我们使用事件而非位置，这使得可以使用<style id='4'>属性投影算子扩展 CEL 语法；该算子用于投影复杂事件中的元组。形式化地说，我们使用以下算子扩展 CEL 公式的语法：

$$\varphi \quad := \quad \pi_{X(\mathbf{a}_1,\ldots,\mathbf{a}_k)}(\varphi) \qquad \text{(tuple projection)}$$

其中， φ 是任意 CEL 公式， X 是 $\mathbf{X}$ 中的一个变量，而 $\mathbf{a}_1$、.....、$\mathbf{a}_k$ 是 $\mathbf{A}$ 中的属性列表。直观地说，这意味着在变量 X 中的事件里，只考虑属性 $\mathbf{a}_1$、.....、$\mathbf{a}_k$。

我们递归地定义属性投影算子 $\pi_{X(\mathbf{a})}$ 的形式语义如下。对于属性列表 $\mathbf{a}_1, \ldots, \mathbf{a}_k$ 和事件 e，我们将 $\pi_{\mathbf{a}_1,\ldots,\mathbf{a}_k}(e)$ 定义为新事件 e'，使得 $\text{Att}(e') = \text{Att}(e) \cap \{\mathbf{a}_1, \ldots, \mathbf{a}_k\}$， $e(\text{time}) = e'(\text{time})$，并且只要 $\mathbf{a}_i \in \text{Att}(e')$，就有 $e'(\mathbf{a}_i) = e(\mathbf{a}_i)$。令 $\mathcal{S} = e_1 e_2 \ldots e_n$，其中每个 e_i 都是一个事件。那么：

$$[\![\pi_{X(\mathbf{a}_1,\ldots,\mathbf{a}_k)}(\varphi)]\!](\mathcal{S}) = \big\{ C \mid \exists C' \in [\![\varphi]\!](\mathcal{S}).\ \text{time}(C) = \text{time}(C') \wedge \forall Y \neq X.\, C(Y) = C(Y') \\ \wedge C(X) = \{\!\{\pi_{\mathbf{a}_1,\ldots,\mathbf{a}_k}(e) \mid e \in C'(X)\}\!\} \big\}$$

直观而言，给定一个复杂事件 $C' \in [\![\varphi]\!](\mathcal{S})$，其中 $C'(X) = \{\!\{e_1, \ldots, e_l\}\!\}$，上面的投影公式会创建一个新的复杂事件 C，它与 C' 具有相同的区间，并且变量 $Y \neq X$ 中的事件也相同，但它将 X 中的事件重新定义为 $C(X) = \{\!\{\pi_{\mathbf{a}_1,\ldots,\mathbf{a}_k}(e_1) \ldots, \pi_{\mathbf{a}_1,\ldots,\mathbf{a}_k}(e_l)\}\!\}$。

示例 3.3。 我们再次考虑示例 3.1 和 3.2 中的情形。现在，我们希望在满足相同约束的条件下，获取英特尔的销售价格。为此，我们可以使用元组投影来编写如下查询：

$\varphi_4 = \pi_{\text{intel}(\texttt{price})}$(标记为卖出 msft ; 标记为卖出 intel ; 标记为卖出 amzn) FILTER
([=] ∧ [> 100] ∧ [=] msft 名称 “MSFT”
∧ =
msft 价格 intel 名称 “INTC” amzn[名称 “AMZN”] ∧ amzn[价格 < 2000]).

4 复杂事件识别中的聚合函数建模

在介绍聚合逻辑之前，我们先提出一个基于幺半群、用于对复杂事件识别中的聚合函数进行建模的框架。我们的目标是提出一种尽可能通用的逻辑，涵盖实践中使用的大多数聚合查询，例如求和、最大值、计数或极差。下面我们回顾幺半群和聚合函数的定义，最后陈述我们关于复杂事件识别中聚合的主要假设。

幺半群。幺半群是一个代数结构 $(M, \oplus, \mathsf{O})$，其中 $(M, \oplus)$ 构成一个半群，而 $\mathsf{O} \in M$是 $\oplus$上的单位元。与半群类似，我们还假设 $\oplus$ 是交换的。例如，自然数与加法构成的结构 $(\mathbb{N}, +, 0)$ 是一个交换幺半群；不含零的自然数与乘法构成的结构 $(\mathbb{N} \setminus \{0\}, \times, 1)$ 也是一个交换幺半群。其他例子包括带有 min 的 $(\mathbb{N} \cup \{\infty\}, \min, \infty)$ 和带有 max 的 $(\mathbb{N}, \max, 0)$。给定一个交换幺半群 $(M, \oplus, \mathsf{O})$ 、一个有限多重集 $A = \{\!\{a_1, \ldots, a_n\}\!\} \subseteq M$ 和一个函数 $f\colon M \to M$，我们定义算子：$\bigoplus_{a \in A} f(a) = f(a_1) \oplus \ldots \oplus f(a_n)$，即将 $\oplus$从二元算子推广到一组元素。在

特别地，若 $A = \emptyset$，则定义 $\bigoplus_{a \in A} f(a) = \mathsf{O}$。下文中，我们将使用 $(M, \oplus, \mathsf{O})$ 或 $(M, \otimes, \mathsf{1})$ 表示某个集合 M上的任意交换幺半群。

聚合函数。在本文中，我们采用可通过幺半群定义的最一般的聚合函数定义（见 [22]）。具体而言，聚合函数是一个将值多重集映射到值的函数，形式上表示为 $f : \mathcal{P}_{bags}(\mathsf{D}) \to \mathsf{D}$。若存在一个交换幺半群[1] $(M, \oplus, \mathsf{O})$，使得对于任意不相交的多重集 $X, Y \subseteq \mathsf{D}$，$f(X \uplus Y) = f(X) \oplus f(Y)$，则聚合函数 f 是自分解的。自分解函数的例子包括求和、最大值、最小值和计数。例如，当 $X \uplus Y = \emptyset$ 时，$\mathsf{sum}(X \uplus Y)$ 等于 $\mathsf{0}$；当 $X \uplus Y = \{\!\{x\}\!\}$ 时，等于 x；否则，等于 $\mathsf{sum}(X)+\mathsf{sum}(Y)$。类似地，当 $X \uplus Y = \emptyset$ 时，$\mathsf{count}(X \uplus Y)$ 等于 $\mathsf{0}$；当 $X \uplus Y = \{\!\{x\}\!\}$ 时，等于 $\mathsf{1}$；否则，等于 $\mathsf{count}(X) + \mathsf{count}(Y)$。最后，当 $X \uplus Y = \emptyset$ 时，$\mathsf{min}(X \uplus Y)$ 等于 ∞；当 $X \uplus Y = \{\!\{x\}\!\}$ 时，等于 x；否则，等于 $\mathsf{min}(\mathsf{min}(X), \mathsf{min}(Y))$。

遗憾的是，实际中并非所有聚合函数都是自分解的；然而，大多数聚合函数仍可在应用一个简单运算之前进行分解。形式上，如果存在函数 $g : M \to M'$ 以及一个自分解聚合函数 h，使得 $f = g \circ h$，则称聚合函数 f 可分解的 。此外，我们假设 g 可以在常数时间内计算（即在随机存取机模型中）。这一条件是必要的，因为我们希望在 h完成聚合后，将 g 作为最后一步来执行一个简单运算（即耗时为常数），而不是让它强大到足以自行完成聚合。

每个自分解函数也都是可分解的（即，g 是恒等函数）。其他可分解（但非自分解）的聚合函数示例包括平均值和极差。例如，可以将平均值定义为 $\mathsf{avg}(X) = g(h(X))$，其中 $h(\{\!\{x\}\!\}) = (x, \mathsf{1})$，且 $h(X \uplus Y) = h(X)+h(Y)$；这里 $+$ 是对偶进行的标准逐点求和，而 $g((s, c)) = s/c$。另一个例子是极差，它可以定义为 $\mathsf{range}(X) = g(h(X))$，其中 $h(x) = (x, x)$，$h(X \uplus Y) = (\text{最大值}(X \uplus Y), \text{最小值}(X \uplus Y))$，且 $g((s, c)) = s - c$。在这两种情况下，都可以验证 h 是自分解函数，且 g 可以在随机存取机模型中以常数时间计算。

请注意，尽管自分解函数可以通过幺半群进行分解，但它们并不能完全由幺半群确定（例如计数）。然而，正如下一个引理所示，我们可以先将值映射到相应的幺半群，再将讨论限制在幺半群上。

引理 4.1。 f是自分解的，当且仅当存在一个交换幺半群（M，$\oplus$，O）和一个函数 $f' : \mathbf{D} \to M$，使得对于每个多重集 X，$f(X) = \bigoplus_{a \in X} f'(a)$。

根据前一个引理，如果存在一对（M，f'），使得 M 是一个交换幺半群，且 f' 是恒等函数，我们就称 f是强自分解的。换句话说，f 可以直接由一个交换幺半群定义。求和、最小值和最大值都是强自分解函数。另一方面，计数需要先将每个值映射为 $\mathsf{1}$，再将它们相加。

为简化起见，下文假设所有聚合函数都是强自可分解的。换言之，我们可以通过交换幺半群直接定义聚合函数的语义。我们可以作此假设，因为函数 f' 和 g（即，可分解聚合函数的函数）在读取每个数据项时或聚合完成后都能在常数时间内计算，例如，函数 f' 将单个元素映射为 $\mathsf{1}$（如计数），以及最终函数 g 计算两个元素之差（如极差），或将一个元素除以另一个元素（如平均值）。这一假设大大简化了我们的设定，使我们能够专注于聚合中最相关的细节，同时不舍弃实践中相关的聚合函数。

5 聚合复杂事件逻辑

本节介绍我们提出的为 CEL 扩展聚合功能的方案。具体而言，我们将基于以往工作的经验，展示如何为 CEL 扩展聚合操作。我们通过示例说明，这一新算子足以对早期工作中使用的大多数查询进行建模。我们首先介绍用于聚合函数建模的代数结构，随后将用它来定义 CEL 中的聚合算子。

聚合的代数结构。回顾一下，$\mathbf{A}$ 和$\mathbf{D}$ 是我们固定的属性名称集合

[1]在 [22], 自分解的定义中并未提及幺半群。不过，很容易看出， [22] 中的定义蕴含着幺半群的存在。

与数据 分别对应 a 值。我们固定一个代数结构　　结构:

$$\mathcal{D} \;=\; (\mathbf{D}, \oplus_1, \ldots, \oplus_k, \mathbb{0}_1, \ldots, \mathbb{0}_k) \tag{†}$$

定义在**D** 上，使得对于每个 $i \in [k]$，每个 **(D** $\oplus_i$, O$_i$**)** 都构成一个交换幺半群。例如，**(**N $\cup\{\infty\}, +$,最小值, 最大值, O ∞, O**)** 构成这样的代数结构，其中我们假设对于每个 $n \in$ N，都有 $n + \infty = \infty$。不失一般性，我们假设 NULL $\in$ **D**，并且对于每个 $a \in$ **D**，都有 $a \oplus_i$ NULL $=$ NULL，且 $i \in [k]$ （若非如此，可用一个新的值 NULL 扩展 $\mathcal{D}$ ）。NULL 的作用是在某个属性未定义（即对于某个 **a** $\in$**A**， e **(a)** $=$NULL）时，定义事件 e 上的聚合算子。

单属性聚合算子。我们的目标是扩展 CEL 的语法，为其添加一个聚合算子，用于聚合单个事件中的值。为便于说明，我们将先介绍针对单个属性的运算，然后再说明如何将其扩展到多个属性。

具体而言，我们在 CEL 语法中加入单属性聚合算子，称为聚合 *CEL* （ACEL），如下所示:

$$\varphi \;:=\; \mathtt{Agg}_{Y[\mathbf{b} \leftarrow \otimes X(\mathbf{a})]}(\varphi)$$

其中， φ 是任意 CEL 公式， X 和 Y 是**X**中的变量，**a** 和**b** 是**A**中的属性名称，$\otimes$ 是来自 $\mathcal{D}$ 的二元算子, 其中（**D** $\otimes$,**1**）构成一个幺半群。直观而言，语法 $Y[\mathbf{b} \leftarrow \otimes X(\mathbf{a})]$表示聚合将创建一个新事件 e ，并将其存储在变量 Y中；该事件 e包含单个属性**b** ，用于存储 $\otimes$ 聚合，即对 X 中各事件的**a**属性进行聚合。我们递归地定义单属性聚合算子 Agg 的形式语义如下。设 $\mathcal{S} = e_1 e_2 \ldots e_n$ 是一个流。那么:

$$\begin{aligned}
&[\![\mathtt{Agg}_{Y[\mathbf{b} \leftarrow \otimes X(\mathbf{a})]}(\varphi)]\!](\mathcal{S}) \;= \\
&\qquad \big\{ C \;\mid\; \exists C' \in [\![\varphi]\!](\mathcal{S}).\ \mathrm{time}(C) = \mathrm{time}(C') \wedge \forall Z \neq Y.\ C(Z) = C'(Z) \\
&\qquad\quad \wedge\, C(Y) = C'(Y) \uplus \{\!\{ e \mid e = [\mathbf{b} \mapsto \bigotimes_{e' \in C'(X)} e'(\mathbf{a})] \wedge e(\mathrm{time}) = \mathrm{end}(C') \}\!\} \;\big\}
\end{aligned}$$

直观而言，给定一个复杂事件 $C' \in [\![\varphi]\!](\mathcal{S})$，其中 $C'(X) = \{\!\{e_1, \ldots, e_\ell\}\!\}$，上述聚合公式会创建一个新的复杂事件 C ，它与 C' 具有相同的区间和事件，只是 Y 多了一个事件 e（即 $C(Y) = C(Y') \uplus \{\!\{e\}\!\}$），并且 $e(\mathbf{b}) = e_1(\mathbf{a}) \otimes \cdots \otimes e_\ell(\mathbf{a})$。若 $C'(X) = \emptyset$，则返回 $\otimes$ 的幺元 **1**。此外，新事件 e 的 e（时间）=end(C')，即 C' 中的最后一个时间。注意，事件 e 始终有定义，因为我们假设，若某个 e_i 未定义 **a**，则 $e_i(\mathbf{a}) = \mathrm{NULL}$，因此 $e(\mathbf{a}) =$NULL。

下文中，我们使用一些特殊符号来表示诸如求和、最大值、最小值等有用函数，而不是使用 $\oplus$。例如，如果使用求和函数，我们写作 Agg$^{Y[\mathbf{b} \leftarrow \mathrm{sum}(X(\mathbf{a}))]}(\varphi)$。此外，请注意，尽管我们使用交换幺半群来定义语义，但这一语义可以轻松推广到诸如计数、平均值或极差等可分解的聚合函数。因此，不失一般性，我们也写作例如 Agg$^{Y[\mathbf{b} \leftarrow \mathrm{count}(X(\mathbf{a}))]}(\varphi)$，尽管严格来说，计数并不是强自分解聚合函数。

示例 5.1。 再次考虑示例 3.1 和 3.2 中的场景。现在，我们希望在相同约束下，获取一次微软销售与一次亚马逊销售之间的英特尔销售序列中的最高价格，并将其存储在变量 M 的属性 MAX 中。然后，我们可以使用聚合算子将此查询指定为:

φ_6 =聚合$_{M[\mathtt{MAX} \leftarrow \max(\mathrm{intel}(\mathtt{price}))]}$(标记为卖出 msft; (标记为卖出 intel)+; 标记为卖出 amzn)筛选 [msft[名称 = “MSFT”] $\wedge$ msft[价格 $>$ 100] $\wedge$ intel[名称 = “INTC”]$\wedge$ amzn[名称 =“AMZN”] $\wedge$ amzn[价格 $<$ 2000]])

读者可以从 Agg 的语义中看出，intel 序列的最大值将存储在变量 M 处的一个新事件中。

示例 5.2. 再举一个例子，假设现在我们想获取价格在 100 到 2000 之间、处于上升趋势中的序列 (买入 或 卖出) 的长度，并将其存储在一个属性中

QNT 到变量 Q 中。此外，我们还想检查该长度是否大于 5。于是，我们可以将查询表示为：

$$\varphi_7 \;=\; \Big[\text{聚合}_{M[\texttt{QNT}\leftarrow \mathrm{count}(\mathrm{m}(\texttt{price}))}\big(\big[(\text{买入 或 卖出})\ \text{AS}\ \ l;\ (\text{买入 或 卖出})\ +\text{AS}\ \ m;\ (\text{买入 或 卖出})\ \text{AS}\ \ h\big]\text{FILTER}\ \ [\mathrm{l}[\text{价格}\ < 100]\ \wedge \mathrm{m}[\text{价格}\ \geq 100] \wedge\ \mathrm{m}[\text{价格} \leq 2000]\ \wedge \mathrm{h}[\text{价格}\ > 2000]]\big)\Big]\ \text{FILTER M}[\text{QNT}\ > 5]$$

请注意，尽管在前一个示例中，聚合是针对一个简单的 CEL 公式（即在最顶层）执行的，但在 ACEL 中，所有算子（包括聚合算子）都可以自由组合。具体而言，我们可以对已计算出的聚合应用筛选器（例如 FILTER M.QNT $>$ 5）。

多属性聚合算子。现在我们介绍将聚合算子推广到多个属性的形式。尽管这个推广版本更冗长，但在实践中，为了同时对不同属性中的不同集合进行聚合，这是必需的。我们将（多属性）聚合算子扩展到 CEL 的语法中：

$$\varphi \;:=\; \mathtt{Agg}_{Y[\mathbf{b}_1\leftarrow\otimes_1 X_1(\mathbf{a}_1),\ldots,\mathbf{b}_\ell\leftarrow\otimes_\ell X_\ell(\mathbf{a}_\ell)]}(\varphi)$$

其中，φ 是任意的 CEL 公式，X_1、……、X_ℓ 和 Y 是 X 中的变量，a1、……、a_ℓ 和 b1、……、b_ℓ 是 A 中的属性名称，$\otimes_1$、……、$\otimes_\ell$ 是来自 $\mathcal{D}$ 的二元算子。直观地说，语法 $Y[\mathrm{b}_1 \leftarrow \otimes_1 X_1(\mathrm{a}_1) \ldots, \mathrm{b}_\ell \leftarrow \otimes_\ell X_\ell(\mathrm{a}_\ell)]$ 表示聚合将创建一个新事件 e，该事件具有属性 b1、……、b_ℓ，并存储在变量 Y 中；其中，每个属性 b_i 都将存储 X_i 中事件的 a_i 属性的 $\otimes_i$-聚合。

给定一个流 $\mathcal{S}$，Agg 的推广形式的形式语义如下：

$$\begin{aligned}&[\![\mathtt{Agg}_{Y[\mathbf{b}_1\leftarrow\otimes_1 X_1(\mathbf{a}_1),\ldots,\mathbf{b}_\ell\leftarrow\otimes_\ell X_\ell(\mathbf{a}_\ell)]}(\varphi)]\!](\mathcal{S}) \;=\\ &\qquad \big\{C \mid \exists C' \in [\![\varphi]\!](\mathcal{S}).\ \mathrm{time}(C) = \mathrm{time}(C') \wedge \forall Z \neq Y.\, C(Z) = C'(Z) \wedge C(Y) = C'(Y) \uplus\\ &\qquad\quad \{\!\{e \mid e = [\mathbf{b}_1 \mapsto \bigotimes_{1}\limits_{e'\in C'(X_1)} e'(\mathbf{a}_1), \ldots, \mathbf{b}_\ell \mapsto \bigotimes_{\ell}\limits_{e'\in C'(X_\ell)} e'(\mathbf{a}_\ell)] \wedge e(\mathrm{time}) = \mathrm{end}(C')\}\!\}\big\}\end{aligned}$$

直观地说，Agg 的一般形式允许通过分别对属性$\mathbf{a}_1,\ldots,\mathbf{a}_\ell$执行聚合，来定义多个属性$\mathbf{b}_1,\ldots,\mathbf{b}_\ell$。其思路与单属性聚合算子类似，但可以一次处理多个属性$\mathbf{b}_1,\ldots,\mathbf{b}_\ell$。

在附录 D 中，我们展示如何使用 ACEL 描述先前学术提案和实际系统中的若干示例。具体而言，我们展示了文献中的一些示例，在这些示例中需要使用多属性聚合算子。与简单属性聚合算子类似，在 ACEL 中，可以自由组合所有算子，包括这个新的聚合算子。本节最后，我们将讨论在 CEL 中定义聚合算子时所作的若干相关设计决策。

为什么 CEL 中的聚合采用这种语义？。在 CEL 中定义聚合语义有多种方式；然而，我们为 CEL 和 ACEL 提出的方案包含一些需要论证的关键设计决策。具体而言，我们提出的语义：(1) 输出复杂事件的集合（即没有重复项）；(2) 每个复杂事件都包含事件多重集；(3) 每个事件都有一个时间戳，用以标明其到达或创建的时间。事实上，我们也可以考虑其他方案，例如输出复杂事件袋的语义、复杂事件内部采用集合的语义，或不带时间戳的事件语义，也可以组合采用这些方案。接下来，我们将讨论为何提出基于 (1)、(2) 和 (3) 的语义，以及采用其他方案会带来哪些后果。

对于（1），如果我们选择基于复杂事件袋的语义，而不考虑其他选项，那么根据查询的指定方式，语义输出的结果可能会重复。例如，假设采用基于多重集的语义，并且用户编写了以下查询：

$$\varphi_1 \;=\; \pi_X(A\ \text{AS}\ X;\ B+;\ A\ \text{AS}\ X)$$

在流 $\mathcal{S}_1 = A_1B_2B_3A_4$ 上，其中 A 和 B 是事件的类型（即，属性中的数据无关紧要）。如果我们使用基于多重集的语义对 $\mathcal{S}_1$ 执行 φ_1，则会得到相同的结果

（1，4）会出现多次（潜在地呈指数级多次），具体取决于每个结果捕获了多少个 B 。相反，基于集合的语义确保每个复杂事件只出现一次，无论查询是如何指定的。

对于（2），如果我们选择将复杂事件中的每个变量映射到事件集合，而不是事件多重集，就可能得到一些未考虑某些结果的答案，因为这些结果会被视为重复元素。例如，如果我们考虑以下查询：

$$\varphi_2 \;=\; \mathtt{Agg}_{Y(\mathbf{b}\leftarrow\mathrm{sum}(X(\mathbf{a})))}\big[(\mathtt{Agg}_{X(\mathbf{a}\leftarrow\mathrm{sum}(A(\mathbf{value})))}[B\;:\;A\oplus])\oplus\big]$$

以及流 $\mathcal{S}_2 = B_1A_2[\mathbf{a}:3]A_3[\mathbf{a}:5]B_4A_5[\mathbf{a}:2]A_6[\mathbf{a}:4]A_7[\mathbf{a}:2]$, 首先，我们会得到两个匹配（一个来自 $B_1A_2A_3$ ，另一个来自 $B_4A_5A_6A_7$），然后分别对它们进行聚合，但每次聚合的结果都相同（即 8）。它们来自不同的值，却不会被保存为两个不同的值，因此，最后外层聚合将作用于一个元素，而不是两个。

最后，对于（3），如果我们认为每个到达或创建的事件都没有时间戳（即来源标记），那么在聚合过程中仍可能丢失一些信息（即使我们在复杂事件内部使用多重集来存储事件）。例如，考虑以下查询

$$\varphi_3 \;=\; (\mathtt{Agg}_{Y(\mathbf{b}\leftarrow\mathrm{sum}(X(\mathbf{a})))}(X\oplus)\,;\,\psi_1)\;\;\mathrm{OR}\;\;(\psi_2\,;\,\mathtt{Agg}_{Y(\mathbf{b}\leftarrow\mathrm{sum}(W(\mathbf{a})))}(W\oplus))$$

对于某些子公式 ψ_1 和 ψ_2。对于流上的公式 φ_3 ，析取式（即 OR ）左右两部分的聚合结果可能相同，因此应用或运算符时，无法区分聚合来自哪一部分。相反，假设每个事件都有一个时间戳（包括通过聚合创建的事件），对于 φ_3，至少会有两个输出，由此可以区分执行聚合的位置。

需要注意的是，CEL 和聚合运算符还可以采用另一种语义，以上论述并未否定这种语义。不过，这可能会导致查询语言出现令用户难以预料的行为。在本研究中，我们选择专注于基于 (1)、(2) 和 (3) 的语义，研究其性质，并将其他逻辑变体的研究留待未来工作。

ACEL 的表达能力。引入聚合这类新算子时，我们希望它只建模其预期功能；然而，将它与其他算子结合，可能会产生能够表达的意外性质。尤其是，将聚合算子与过滤器结合非常强大，因为这样可以检查事件之间的等价性。例如，考虑以下包含聚合和过滤的查询：

$$\pi_{X,Y}\big(\mathtt{Agg}_{Z(\mathbf{b}_1\leftarrow\mathrm{sum}(X(\mathbf{a})),\mathbf{b}_2\leftarrow\mathrm{sum}(Y(\mathbf{a})))}(R\;\;\mathrm{AS}\;\;X\,;T\;\;\mathrm{AS}\;\;Y)\;\;\mathrm{FILTER}\;[Z(\mathbf{b}_1=\mathbf{b}_2)]\big)$$

直观地说，前一个查询检查类型为 T 的事件（将其命名为 Y）是否发生在类型为 R 的事件（将其命名为 X）之后，并分别对两个事件中属性 $\mathbf{a}$ 的值求和，将这些值保存到一个事件中变量 Z 的属性 $\mathbf{b}_1$ 和 $\mathbf{b}_2$ 中。然后，查询通过检查属性 $\mathbf{b}_1$ 和 $\mathbf{b}_2$ 的值是否相等来对其进行过滤，因为它们对应于同一个事件。最后，它投影变量X 和 Y。可以看出，该查询仅使用针对单个事件的聚合和过滤器算子，来关联不同类型的事件。

这种将聚合与过滤结合起来的意外行为，是一种有趣的副作用，可能有助于更好地理解复杂事件识别中的聚合。请注意，如果没有查询语言具体且形式化的语义，就不可能观察到聚合器、过滤器和其他算子之间的这种交互。

6 聚合的自动机模型（复杂事件识别）

本文提出一种用于聚合的自动机模型，该模型通过寄存器扩展复杂事件自动机，与代价寄存器自动机的模型 [5] 类似。我们先回顾复杂事件自动机（CEA）模型，随后给出引入我们新的聚合自动机模型所需的定义。

复杂事件自动机。复杂事件自动机（*CEA*） [20, 9] 是一个元组 $\mathcal{A} = (Q,\ \Delta,\ q_0,\ F)$，其中 Q 是有限状态集，$\Delta \subseteq Q \times \mathbf{P} \times \mathcal{P}(\mathbf{X}) \times Q$ 是有限转移关系，$q_0 \in Q$ 是初始状态，而 $F \subseteq Q$ 是终态集。从 $\mathcal{A}$ 在流 $\mathcal{S} = e_1 \dots e_n$ 的位置 i 到 j 的一次运行是一系列转移:

$$\rho\ :=\ q_i \xrightarrow{P_i/L_i} q_{i+1} \xrightarrow{P_{i+1}/L_{i+1}} \dots \xrightarrow{P_j/L_j} q_{j+1}$$

使得 $q_i = q_0$ 是 $\mathcal{A}$ 的初始状态，并且对于每个 $k \in [i..j]$，都有 $(q_k, P_k, L_k, q_{k+1}) \in \Delta$ 和 $e_k \models P_k$。如果 $q_{j+1} \in F$，则运行 ρ 是接受的。$\mathcal{A}$ 在流 $\mathcal{S}$ 上从 i 到 j 的一次接受运行 ρ 自然定义了复杂事件 $C_\rho := (i, j, \mu_\rho)$，使得对于每个 $X \in \mathbf{X}$，$\mu_\rho(X) = \{t_k \mid i \le k \le j \wedge X \in L_k\}$。如果位置 i 和 j 可由上下文确定，我们就称 ρ是 $\mathcal{A}$ 在 $\mathcal{S}$上的一次运行。最后，我们将 $\mathcal{A}$ 在流 $\mathcal{S}$ 上的语义定义为：$[\![\mathcal{A}]\!](\mathcal{S}) := \{C_\rho \mid \rho$ 是 $\mathcal{A}$ 在 $S\}$.上的一次接受运行

复杂事件自动机对于捕捉 CEL 的表达能力、将 CEL 查询编译为复杂事件自动机以及高效地对其进行求值至关重要。不幸的是，很容易发现，复杂事件自动机并不适用于我们的 ACEL 语义，因为它无法记住 我们已见过的属性值以进行聚合。换言之，它没有机制像 ACEL 的新语义那样聚合值并生成新事件。我们将在接下来的定义中展示如何克服这些缺陷。

表达式。回顾一下，$\mathbf{A}$ 是一个固定的属性集合，$\mathbf{D}$ 是一个固定的数据值集合。此外，回顾第5节中我们固定了一个形式为 (†) 的代数结构，其定义在 $\mathbf{D}$ 上，使得对于每个 $i \in [k]$，每个 $(\mathbf{D}, \oplus_i, \mathsf{O}_i)$ 都构成一个交换幺半群。我们将一个 $(\mathcal{D},\mathbf{A})$-表达式 e （或简称为表达式）定义为由文法生成的、在 $\mathcal{D}$ 和 $\mathbf{A}$ 上的语法公式:

$$\alpha\ :=\ d\ \mid\ \mathbf{a}\ \mid\ \alpha \oplus_i \alpha \quad i \in [k]$$

其中 $d \in \mathbf{D}$ 且 $\mathbf{a} \in \mathbf{A}$。我们用 Expr($\mathcal{D}$,**A**) 表示所有 $(\mathcal{D}, \mathbf{A})$-表达式 构成的集合。对于任意表达式$\alpha \in$ Expr($\mathcal{D}$,**A**)，我们用 Att(α) 表示 α 中所有属性构成的集合。给定一个事件 $e : \mathbf{A} \mapsto \mathbf{D}$，以及一个满足 Att($\alpha$) $\subseteq$ Att(e) 的表达式 α，我们将 α 在 e 上的语义定义为 $[\![\alpha]\!](e)$，即将每个 $\mathbf{a} \in \mathbf{A}$ 替换为 e(**a**) 后对 α 求值所得的 $\mathbf{D}$ 中的值。

例6.1. 设 $\mathcal{D}$ = (D, min,max $+, \infty$, 0, 0)，表达式 α = a + b 和 β = min(a, b) +max(b, c)，以及一个事件 e 其中 e(a) = 4, e(b) = 2 与 e(c) = 5。那么，每个表达式在该事件 α与 β 上的结果分别为 $[\![\alpha]\!](e) = 6$ 和 $[\![\beta]\!](e) = 7$。

赋值。一个 ($\mathcal{D}$, A)-赋值（当 $\mathcal{D}$ 和 A 可根据上下文明确时，也简称赋值）是一个将 A 中的属性赋值给 Expr($\mathcal{D}$, A) 中表达式的程序。形式上，赋值定义为一个映射 σ : A $\mapsto$ Expr($\mathcal{D}$, A)。与表达式类似，我们将 $\mathrm{Att}_{\mathrm{in}}(\sigma) = \bigcup_{\mathrm{a} \in \mathrm{dom}(\sigma)} \mathrm{Att}(\sigma(\mathrm{a}))$ 定义为 σ 中表达式所用的所有属性，并将 $\mathrm{Att}_{\mathrm{out}}(\sigma)$ = dom(σ) 定义为所有被赋值的属性。给定一个事件 e : A $\to$ D 和一个 ($\mathcal{D}$, A)-赋值 σ，使得 $\mathrm{Att}_{\mathrm{in}}(\sigma) \subseteq$ Att(e)，赋值 σ 在 e 上的语义是一个事件 $e' := [\![\sigma]\!](e) : \mathrm{A} \mapsto \mathrm{D}$，使得 Att($e'$) = $\mathrm{Att}_{\mathrm{out}}(\sigma)$，并且对于每个 a $\in \mathrm{Att}_{\mathrm{out}}(\sigma)$，都有 e'(a) = $[\![\sigma(\mathrm{a})]\!](e)$。换言之，$[\![\sigma]\!](e)$ 是将赋值 σ 应用于事件 e 中的值后得到的结果。我们用 Asg($\mathcal{D}$, A) 表示所有 ($\mathcal{D}$, A)-赋值的集合。

例6.2。再考虑例6.1中的情形，以及赋值 σ，其定义如下：σ: **a** $\leftarrow$max(**a** + **b**,**c**)。这里，我们将 σ 视为一个程序，其中 $\leftarrow$ 的左侧用右侧更新，即 σ(**a**) = max(**a** + **b**,**c**)。于是，$[[\![\sigma]\!](e)](\mathbf{a}) = 6$。

最后，我们回顾重命名的概念（第3节），并定义对事件的更新，这对我们的自动机模型很有用。请记住，重命名 r 定义为 $r : \mathbf{A} \mapsto \mathbf{A}$，它将每个属性映射到一个新属性。可以注意到，重命名也是赋值的一种特殊情况$r : \mathbf{A} \mapsto$ Expr($\mathcal{D}$,**A**)，满足 r(**a**) $\in \mathbf{A}$。此外，回顾一下，我们用 Ren 表示 $\mathbf{A}$ 上所有元组重命名构成的集合。给定事件 e 和 e'，我们将 e' 通过 e进行的更新 记作 $e \gg e'$，并将其定义为一个新事件，使得 Att($e \gg e'$) = Att(e) $\cup$Att(e')，且当 $\mathbf{a} \in$ Att(e) 时，$[e \gg e']$(**a**) = $e(a)$；否则，$[e \gg e']$(**a**) = e'(**a**)。

聚合复杂事件自动机。我们现在可以定义带聚合的复杂事件自动机模型。聚合复杂事件自动机（*ACEA*）是一个元组 $\mathcal{A} = (Q, \Delta, q_0, F)$，其中 Q 是有限状态集，$q_0 \in Q$ 是初始状态，$F \subseteq Q$ 是终态，并且:

$$\Delta \subseteq Q \times \mathrm{Asg}(\mathcal{D}, \mathbf{A}) \times \mathbf{P} \times \{\lambda : \mathbf{X} \mapsto \mathcal{P}_{bags}(\mathrm{Ren})\} \times Q$$

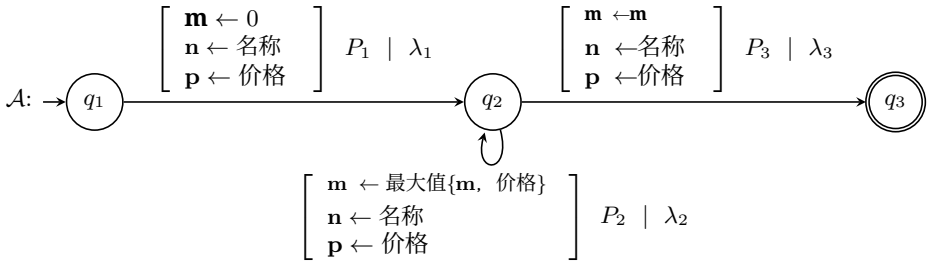

图 2：一个表示示例 5.1 中给定查询的 ACEA $\mathcal{A}$，其中 $P_1 := \mathbf{n} =$ "MSFT" $\wedge \mathbf{p} > 100$，$P_2 := \mathbf{n} =$ "INTC" 与 $P_3 := \mathbf{n} =$ "AMZN" $\wedge \mathbf{p} < 2000$。此外，λ_1(msft) $= \lambda_2$(intel) $= \lambda_3$(amzn) $= \{\!\{[\text{名称} \mapsto \mathbf{n},\ \text{价格} \mapsto \mathbf{p}]\}\!\}$，且 λ_3(M) $= \{\!\{[\text{最大值} \mapsto \mathbf{m}]\}\!\}$。

是一个有限转移关系，其中 $\{\lambda : \mathbf{X} \mapsto \mathcal{P}_{bags}(\text{Ren})\}$ 是所有映射 λ 的集合，这些映射将变量 X 映射到有限重命名多重集 $\{\!\{r_1, \ldots, r_k\}\!\}$。转移 $(p, \sigma, P, \lambda, q) \in \Delta$ 指定了 $\mathcal{A}$在读取一个事件后，可以通过使用 σ 更新一些内部寄存器，并使用 P检查一个条件（针对寄存器），从状态 p 转移到状态 q。与复杂事件自动机类似，λ 负责创建复杂事件的输出，其中重命名 $\lambda(X) = \{\!\{r_1, \ldots, r_k\}\!\}$ 会根据内部寄存器中存储的值，在变量 X 中创建 k 个新元组。我们假设，形式为 $(p, \sigma, P, \lambda, q) \in \Delta$ 的转移中，λ里的重命名与 σ一致，即对于每个 $X \in \text{dom}(\lambda)$，$\text{Att}_{\text{in}}(\lambda(X)) \subseteq \text{dom}(\sigma)$。

一对 (q, ν) 是 $\mathcal{A}$ 的一个配置，其中 $q \in Q$ 与 $\nu : \mathbf{A} \mapsto \mathbf{D}$ 是一个事件，表示属性的当前值。为简化起见，在 ACEA 中，我们将属性用作存储临时值的"寄存器"。因此，配置 (q, ν) 表示自动机处于状态 q，并且寄存器 dom(ν)（即属性的一个子集）存储当前计算出的值。

设 $\mathcal{S} = e_1 \ldots e_n$为一个流。从位置 i到 j、在流 S上运行 $\mathcal{A}$，是由配置和转移组成的序列:

$$\rho := (q_i, \nu_i) \xrightarrow{\sigma_i, P_i/\lambda_i} (q_{i+1}, \nu_{i+1}) \xrightarrow{\sigma_{i+1}, P_{i+1}/\lambda_{i+1}} \cdots \xrightarrow{\sigma_j, P_j/\lambda_j} (q_{j+1}, \nu_{j+1}) \qquad (\ddagger)$$

满足以下条件：q_i 是初始状态 q_0，ν_i 是空事件（即 dom(ν_i) $= \emptyset$），并且对于每个 $k \in [i..j]$，$(q_k, \sigma_k, P_k, \lambda_k, q_{k+1}) \in \Delta$，$(q_{k+1}, \nu_{k+1})$ 是 $\mathcal{A}$ 的一个配置，其中 $\nu_{k+1} := [\![\sigma_k]\!](e_k \gg \nu_k)$，且 $\nu_{k+1} \models P_k$。此外，还必须满足 $\text{Att}_{\text{in}}(\sigma_k) \subseteq \text{dom}(e_k \gg \nu_k)$。直观地说，新的值 ν_{k+1} 首先通过新事件 e_k（即 $e_k \gg \nu_k$）更新 ν_k，然后通过赋值 σ_k操作 $e_k \gg \nu_k$ 而产生。计算出新的值 ν_{k+1} 后，我们会检查它们是否满足该转移的谓词 P_k。

与复杂事件自动机类似，如果 $q_{j+1} \in F$，运行 ρ is 接受。从 i 到 j，$\mathcal{A}$ 在 $\mathcal{S}$ 上的一个接受运行 ρ，例如（‡），定义了复杂事件 $C_\rho := (i, j, \mu_\rho)$，使得:

$$\mu_\rho(X) = \{\!\{e \mid k \in [i..j] \wedge r \in \lambda_k(X) \wedge e = [\![r]\!](\nu_{k+1}) \wedge e(\text{time}) = k\}\!\}$$

对于每个 $X \in \mathbf{X}$。最后，我们将 $\mathcal{A}$ 在流 $\mathcal{S}$ 上的语义定义为:

$$[\![\mathcal{A}]\!](\mathcal{S}) := \{C_\rho \mid \rho \text{ is an accepting run of } \mathcal{A} \text{ over } S\}.$$

示例 6.3。考虑示例 5.1 中的 ACEL 查询。我们可以通过图 2 中的 ACEA 得到相同的结果，其中 $P_1 :=$ n $=$ "MSFT" $\wedge$p > 100, $P_2 :=$ n $=$ "INTC" and $P_3 :=$ n $=$"AMZN" $\wedge$ p < 2000。此外，λ_1(msft) $= \lambda_2$(intel) $= \lambda_3$(amzn) $= \{\!\{[\text{name} \mapsto \text{n}, \text{price} \mapsto \text{p}]\}\!\}$, and$\lambda_3$(M) $= \{\!\{[\text{MAX} \mapsto \text{m}]\}\!\}$。直观地说，在第一个转移中，$\mathcal{A}$ 用 0 初始化寄存器 m（即一个属性），并通过将名称存储在 n 中、将价格存储在 p 中，检查价格和名称属性是否满足谓词 P_1。然后，在 q_2的循环中，$\mathcal{A}$ 用新的价格更新 m 中的最大值，并再次检查名称是否满足 P_2。最后，在最后一个转移中，它保留 m 中的最大值，并验证名称和价格属性是否满足 P_3。映射 λ_1、λ_2 和 λ_3 分别负责输出变量 msft、intel 和 amzn 中的事件。此外，λ_3 负责生成变量 M 中的最终事件，该事件包含英特尔价格的最大值聚合。

首先需要回答的一个自然问题是：新模型 ACEA 的表达能力是否涵盖由复杂事件自动机定义的查询。与 ACEL 和 CEL 之间的问题类似，复杂事件自动机输出带有位置的复杂事件，而我们的新模型则输出带有事件的复杂事件，并具备其他新特性。下面我们将证明，ACEA 可以通过将流中的位置映射到事件，来定义每一个复杂事件自动机。

定理 6.1. 对于模式 Σ上的流，ACEA 与 CEA 的表达能力相同；也就是说，对于每个 CEA $\mathcal{A}$，都存在一个 ACEA $\mathcal{A}'$，使得对于模式 Σ上的每个 $\mathcal{S}$，$[\![\mathcal{A}]\!](\mathcal{S}) = [\![\mathcal{A}']\!](\mathcal{S})$。

与 ACEL 的等价性。本文的第一个主要目标是提供一种查询语言，并为其赋予形式化的指称语义，以便在复杂事件识别中执行聚合。第二个主要目标是提供一种计算模型，用于编译该语言中的查询。在下面的结果中，我们证明 ACEA 是能够实现这一目标的计算模型。具体而言，我们证明 ACEL 中的每个公式 φ 都可以编译为一个 ACEA，从而证明该模型具备执行复杂事件提取和聚合所需的全部特性。

定理 6.2. 设 Σ 为一个模式。对于每个 ACEL 公式 φ，都存在一个 ACEA $\mathcal{A}_\varphi$，使得对于模式 Σ上的每个流 $\mathcal{S}$，$[\![\varphi]\!](\mathcal{S}) = [\![\mathcal{A}_\varphi]\!](\mathcal{S})$。

我们在附录 C 中给出证明。证明通过对公式进行归纳，展示如何将每个算子编译为 ACEA。标准 CEL 运算符的构造方式与 [20]中的类似（AND 运算符除外），但在这里，我们还必须确保正确维护寄存器，以生成输出。

需要指出的是，ACEA 是一种混合自动机模型，需要执行计算（即聚合）、检查过滤器（即谓词）并生成输出（即事件）。因此，在设计该模型时，我们力求在满足所有这些目标的同时，尽可能保持模型简单。这种简洁性有助于理解其表达能力，并设计高效的求值算法。

尽管 ACEA 十分简单，但其表达能力仍强于 ACEL，也就是说，存在一些查询可以用 ACEA 定义，却无法用 ACEL 定义。例如，考虑自然数幺半群 (N $+$, 0)（即求和）。给定流 $R[\mathbf{a}:1]^{n\text{-次}}\ldots\ R[\mathbf{a}:1],$，可以定义一个只有一个寄存器的 ACEA，它总是将当前值加倍，并在事件 $[\mathbf{b}:2^n]$ 中输出其内容。直观地说，采用 (N $+$, 0) 的 ACEL 无法指定此查询，因为它只能生成相对于流中所有值之和线性增长的值。即使我们将寄存器的使用限制为无复制方式（参见 [5] 中的无复制代价寄存器自动机），仍可以设计出无法用 ACEL 指定的 ACEA。例如，给定前述流，可以编写一个 ACEA，使其生成包含以下事件序列的复杂事件：$[\mathbf{b}:1][\mathbf{b}:2]\ldots[\mathbf{b}:n]$（即在一个寄存器中累加输入值，并在每次转移时输出其内容）。由于在 ACEL 中，事件的每个值只能用于生成有限数量的新事件，因此无法用 ACEL 指定这种行为。因此，ACEA 的表达能力强于 ACEL，而如何通过对 ACEA 加以限制来刻画 ACEL，仍是一个有待解决的有趣问题。我们将此问题留待未来工作。

7 未来工作

本文为复杂事件识别中的聚合提供了逻辑基础，但仍留下了若干有待未来研究解决的开放问题。一个相关的开放问题是进一步理解 ACEL 与 ACEA 之间的等价性，即哪些 ACEA 可以用 ACEL 表达。另一个有趣的问题是了解聚合与过滤器及其他算子结合后的表达能力（见第5节）。最后，为了让 ACEL 能够在实践中发挥作用，一项至关重要的研究方向是探讨如何高效地求值 ACEL 查询，寻找这样的枚举算法：给定一个 ACEA 和一个流，算法能够以常数更新时间和常数延迟枚举运行。

参考文献

[1] Esper 企业版网站。https://www.espertech.com/，2025年。 [访问日期：23-06-2025]。

[2] Asaf Adi 和 Opher Etzion。Amit——情境管理器。《VLDB Journal》，13：177–203，2004。[3] Alfred V Aho 和 John E Hopcroft。《计算机算法的设计与分析》。Pearson Education India，1974。[4] Rajeev Alur、Loris DAntoni、Jyotirmoy Deshmukh、Mukund Raghothaman 和 Yifei Yuan。正则函数与代价寄存器自动机。载于《2013年第28届ACM/IEEE计算机科学逻辑研讨会论文集》，第13–22页，2013。doi:10.1109/LICS.2013.65。[5] Rajeev Alur、Loris D'Antoni、Jyotirmoy V. Deshmukh、Mukund Raghothaman 和 Yifei Yuan。正则函数与代价寄存器自动机。载于 LICS，第13–22页。IEEE计算机学会，2013。[6] Alexander Artikis、Alessandro Margara、Mart´ín Ugarte、Stijn Vansummeren 和 Matthias Weidlich。复杂事件识别语言：教程。载于 DEBS，第7–10页。ACM，2017。[7] Miko-laj Boja´nczyk。带起源信息的转换器。载于《自动机、语言与程序设计：第41届国际学术研讨会（ICALP 2014）》，丹麦哥本哈根，2014年7月8–11日，论文集第二部分41，第26–37页。Springer，2014。[8] Kyle Bossonney、Nicol´as Buzeta、Vicente Calisto、Juan-Eduardo L´opez、Cristian Riveros 和 Stijn Vansummeren。CORE+：C++中的复杂事件识别引擎。载于 Volker Markl、Joseph M. Hellerstein 和 Azza Abouzied（编），SIGMOD 演示论文集，第47–50页。ACM，2025年。[9] Marco Bucchi、Alejandro Grez、Andr´es Quintana、Cristian Riveros 和 Stijn Vansummeren。CORE：复杂事件识别引擎。《VLDB》，15(9)：1951–1964，2022。[10] Gianpaolo Cugola 和 Alessandro Margara。Raced：用于复杂事件检测的自适应中间件。载于《第8届自适应与反射式中间件国际研讨会论文集》，第1–6页，2009。[11] Gianpaolo Cugola 和 Alessandro Margara。TESLA：一种形式化定义的事件规范语言。载于 DEBS，第50–61页。ACM，2010。[12] Gianpaolo Cugola 和 Alessandro Margara。处理信息流：从数据流到复杂事件处理。《ACM计算调查》（CSUR），44(3)：1–62，2012。[13] Alan J Demers、Johannes Gehrke、Biswanath Panda、Mirek Riedewald、Varun Sharma、Walker M White 等。Cayuga：通用事件监控系统。载于 CIDR，第7卷，第412–422页，2007。[14] Yanlei Diao、Neil Immerman 和 Daniel Gyllstrom。Sase+：一种用于事件流上克林闭包的敏捷语言。马萨诸塞大学技术报告，2007。[15] Antony Galton 和 Juan Carlos Augusto。事件定义的两种方法。载于《数据库与专家系统应用国际会议论文集》，第547–556页。Springer，2002。[16] Juli´an Garc´ıa 和 Cristian Riveros。时间约束下的复杂事件识别：面向高效查询求值的形式化框架。Proc. ACM Manag. Data，3(2)：94:1–94:17，2025年。[17] Nikos Giatrakos、Elias Alevizos、Alexander Artikis、Antonios Deligiannakis 和 Minos Garofalakis。大数据时代的复杂事件识别：综述。《The VLDB Journal》，29：313–352，2020。[18] Michel Grabisch、Jean-Luc Marichal、Radko Mesiar 和 Endre Pap。《聚合函数》，第127卷。剑桥大学出版社，2009。[19] Alejandro Grez、Cristian Riveros 和 Mart´ín Ugarte。复杂事件处理的形式化框架。载于 ICDT，LIPIcs 第127卷，第5:1–5:18页。Schloss Dagstuhl - Leibniz-Zentrum f¨ur Informatik，2019。[20] Alejandro Grez、Cristian Riveros、Mart´ın Ugarte 和 Stijn Vansummeren。复杂事件识别的形式化框架。《ACM TODS》，46(4)：16:1–16:49，2021。

[21] Mikell P Groover. 自动化、生产系统与计算机集成制造. 印度培生教育出版社，2016。[22] Paulo Jesus、Carlos Baquero 和 Paulo S´ergio Almeida. 分布式数据聚合算法综述。*IEEE Communications Surveys & Tutorials*，17(1):381–404，2014。[23] D Luckham. Rapide：一种通过事件偏序对分布式系统进行仿真的语言和工具集。1996。[24] Lei Ma、Chuan Lei、Olga Poppe 和 Elke A Rundensteiner. Gloria：用于事件趋势聚合的基于图的共享优化器。载于*2022*年国际数据管理会议论文集，第1122–1135页，2022。[25] Biswanath Mukherjee、L Todd Heberlein 和 Karl N Levitt. 网络入侵检测。*IEEE Network*，8(3):26–41，1994。[26] Peter R Pietzuch、Brian Shand 和 Jean Bacon. 分布式系统中的事件组合框架。载于*ACM/IFIP/USENIX*分布式系统平台与开放分布式处理国际会议，第62–82页。Springer，2003。[27] Manolis Pitsikalis、Alexander Artikis、Richard Dreo、Cyril Ray、Elena Camossi 和 Anne-Laure Jousselme. 海事监控中的复合事件识别。载于第*13*届*ACM*分布式与事件驱动系统国际会议论文集，第163–174页，2019。[28] Olga Poppe、Chuan Lei、Lei Ma、Allison Rozet 和 Elke A Rundensteiner. 共享还是不共享：突发事件流上的在线事件趋势聚合。载于*2021*年国际数据管理会议论文集，第1452–1464页，2021。[29] Olga Poppe、Chuan Lei、Elke A Rundensteiner 和 David Maier. 丰富的事件匹配语义下的事件趋势聚合。载于*2019*年国际数据管理会议论文集，第555–572页，2019。[30] Olga Poppe、Chuan Lei、Elke A Rundensteiner 和 David Maier. Greta：基于图的实时事件趋势聚合。*arXiv*预印本 *arXiv:2010.02988*，2020。[31] Olga Poppe、Allison Rozet、Chuan Lei、Elke A Rundensteiner 和 David Maier. Sharon：共享在线事件序列聚合。载于 *2018* 年 *IEEE* 第 *34* 届国际数据工程会议（*ICDE*），第737–748页。IEEE，2018。[32] BS Sahay 和 Jayanthi Ranjan. 供应链分析中的实时商业智能。信息管理与计算机安全，16(1):28–48，2008。[33] Nicholas Poul Schultz-Møller、Matteo Migliavacca 和 Peter Pietzuch. 通过查询重写实现分布式复杂事件处理。载于第三届*ACM*分布式事件驱动系统国际会议论文集，第1–12页，2009。[34] Luc Segoufin. 以常数延迟枚举查询答案。载于第*16*届国际数据库理论会议论文集，第10–20页，2013。[35] Walker White、Mirek Riedewald、Johannes Gehrke 和 Alan Demers. 事件处理中“下一个”是什么？载于第*26*届*ACM SIGMOD-SIGACT-SIGART*数据库系统原理研讨会论文集，第263–272页，2007。[36]Eugene Wu、Yanlei Diao 和 Shariq Rizvi. 流上的高性能复杂事件处理。载于*2006*年*ACM SIGMOD*国际数据管理会议论文集，第407–418页，2006。[37] Haopeng Zhang、Yanlei Diao 和 Neil Immerman. 复杂事件处理中高开销查询的复杂度与优化。载于*SIGMOD*，2014。[38] Detlef Zimmer 和 Rainer Unland. 主动数据库管理系统中的复杂事件语义。载于*ICDE*，第392–399页。IEEE，1999。

第3节的证明

A.1 定理 3.1 的证明

证明。 设 $\mathcal{S}$为一个流， φ为一个查询。令 $C = (i, j, \mu) \in [\![\varphi]\!](S)$，其中 C由赋值语义得到，而每个 $\mu(X_i)$都是与变量 X_i 中的事件相对应的位置集合（旧语义）。另一方面，令 $C_{new} = (i, j, \mu_{new}) \in [\![\varphi]\!](S)$，其中 μ 是从变量到事件多重集的映射（新语义）。我们可以构造一个从旧语义到新语义的函数 f ，使得 $f\colon \mu(X) \to \mu_{new}(X)$，也就是说，它将 $\mu(X_i)$中的每个位置映射到 $\mu_{new}(X_i)$中的一个事件。可以看出，f 是单射且满射，因为 $\mu(X_i)$和 $\mu_{new}(X_i)$来自 φ的相应赋值，并且两种语义得到相同的事件，只是一种标记事件的位置，另一种标记该位置上的事件。

$$
\begin{aligned}
&[\![R]\!](S) = \{(i, i, R \mapsto \{j\}) \mid S[j] \in \text{元组}(R)\},\ [\![\varphi \text{ 作为 } A]\!](S) = \{(i, j, \mu[A \mapsto \text{上确界}(\mu)]) \mid \mu \in [\![\varphi]\!](S)\},\ [\![\varphi \text{ FILTER } P(A_1, \ldots, A_n)]\!](S) = \{(i, j, \mu \in [\![\varphi]\!](S)) \mid (S[\mu(A_1)], \ldots, S[\mu(A_n)]) \in P\},\ [\![\varphi_1 \text{ 或 } \varphi_2]\!](S) = [\![\varphi_1]\!](S) \cup [\![\varphi_2]\!](S)[\![\varphi_1 \text{ 与 } \varphi_2]\!](S) = [\![\varphi_1]\!](S) \cap [\![\varphi_2]\!](S)[\![\varphi_1 ; \varphi_2]\!](S) = \{(i, j, \mu_1 \cup \mu_2) \mid \exists k,\ i \le k < j\colon \mu_1 \in [\![\varphi_1]\!](S, i, k),\ \mu_2 \in [\![\varphi_2]\!](S, k+1, j)\},\ [\![\varphi_1 : \varphi_2]\!](S) = \{(i, j, \mu_1 \cup \mu_2) \mid \exists k,\ i \le k < j\colon \mu_1 \in [\![\varphi_1]\!](S, i, k),\ \mu_2 \in [\![\varphi_2]\!](S, k+1, j),\ \text{最大值}(\sup(\mu_1)) = k,\ \text{最小值}(\sup(\mu_2)) = k+1\},\ [\![\varphi+]\!](S) = [\![\varphi]\!](S) \cup [\![\varphi ; \varphi+]\!](S)[\![\varphi\oplus]\!](S) = [\![\varphi]\!](S) \cup [\![\varphi : \varphi\oplus]\!](S)[\![\pi_L(\varphi)]\!](S) = \{(i, j, \mu|_L) \mid \mu \in [\![\varphi]\!](S)\}
\end{aligned}
$$

图 3：CEL 公式在流 $S = e_1 e_2 \ldots e_n$ 上定义的旧语义，其中每个 e_i 都是一个事件，且位置介于 i 和 j 之间。

□

B 第 4 节的证明

B.1 引理 4.1 的证明

证明。 首先，我们假设 f' 和 $\oplus$ 可以在常数时间内计算。然后，如果 f 是自分解的，我们知道存在一个交换幺半群 $(M, \oplus, 0)$，并且对于任意不相交的多重集 $X = \{\!\{a_1^X, \ldots, a_k^X\}\!\}$ 和 $Y = \{\!\{a_1^Y, \ldots, a_l^Y\}\!\}$，都有 $f(X \uplus Y) = f(X) \oplus f(Y)$ 成立。根据定义，我们可以递归地分解每个多重集，直到得到作用于单个元素的函数。从示例中可以看出， $f(\{\!\{x\}\!\})$ 可以是 x 或 1，因此我们可以概括为：存在一个函数 f'，使得 $f(\{\!\{x\}\!\}) = f'(x)$。所以，

$$
\begin{aligned}
f(X \uplus Y) &= f(X) \oplus f(Y) \\
&= f'(a_1^X) \oplus \cdots \oplus f'(a_k^X) \oplus f'(a_1^Y) \oplus \cdots \oplus f'(a_l^Y) \\
&= \bigoplus_{a_i^X \in X} f'(a_i^X) \oplus \bigoplus_{a_i^Y \in Y} f'(a_i^Y) \\
&= \bigoplus_{a \in X \uplus Y} f'(a)
\end{aligned}
$$

最后，条件得证。

设 $(M, \oplus, \mathbf{O})$ 是一个幺半群，f 且 $f' \colon \mathbf{D} \to M$ 是函数，并且对于每个多重集 X，$f(X) = \bigoplus_{a \in X} f'(a)$。另一方面，设 $X = \{\!\{a_1^X, \ldots, a_k^X\}\!\}$ 和 $Y = \{\!\{a_1^Y, \ldots, a_l^Y\}\!\}$ 是两个不相交的多重集，使得 $f(X) = \bigoplus_{a_i^X \in X} f'(a_i^X)$ 且 $f(Y) = \bigoplus_{a_i^Y \in Y} f'(a_i^Y)$。我们可以将 $\bigoplus_{a_i^X \in X} f'(a_i^X)$ 展开为 $f'(a_1^X) \oplus \cdots \oplus f'(a_k^X)$。于是，

$$\begin{aligned} f(X \uplus Y) &= \bigoplus_{a \in X \uplus Y} f'(a) \\ &= f'(a_1^X) \oplus \cdots \oplus f'(a_k^X) \oplus f'(a_1^Y) \oplus \cdots \oplus f'(a_l^Y) \\ &= \bigoplus_{a_i^X \in X} f'(a_i^X) \oplus \bigoplus_{a_i^Y \in Y} f'(a_i^Y) \\ &= f(X) \oplus f(Y) \end{aligned}$$

最后，f 是自分解的。 □

C 第 6 节的证明

定理 6.1 的证明

证明。设 $\Sigma : \mathbf{T} \to \mathcal{P}(\mathsf{A}_\Sigma)$ 是一个模式。我们假设，对于每个属性 $\mathsf{a} \in \mathsf{A}_\Sigma$，都存在一个副本 $\mathsf{a}' \in \mathsf{A} \setminus \mathsf{A}_\Sigma$。我们将使用这个副本 a' 暂时存储来自属性的值，这些值位于¯机器的寄存器中，以便随后生成事件的完全相同副本。对于集合 $A \subseteq \mathsf{A}_\Sigma$，令 $A = \{\mathsf{a}' \in \mathsf{A}_\Sigma \mid \mathsf{a} \in A\}$。¯我们将 $\sigma_A \colon A \cup \{\text{类型}\} \to A \cup \{\text{类型}\}$ 定义为如下赋值：将 type 映射到 type，并将每个 $\mathsf{a} \in A$ 映射到其副本 a'，即 $\sigma_A(\mathsf{type}) = \mathsf{type}$，而 $\sigma_A(\mathsf{a}) = \mathsf{a}'$，否则。注意，$\sigma_A$ 是一个双射，因此 σ_A^{-1} 定义良好。

设 $\mathcal{A} = (Q, \Delta, q_0, F)$ 是一个复杂事件自动机。我们将 $\mathcal{A}$ 在流 Σ 上的相同行为定义为 ACEA $\mathcal{A}' = (Q, \Delta', q_0, F)$，其中 Q、q_0 和 F 保持不变，而 Δ' 定义如下：

$$\begin{aligned} \Delta' = \{(p, \sigma_{\Sigma(A)}, P_A, \lambda_A, q) \mid\ & (p, P, L, q) \in \Delta \ \wedge\ A \in \mathbf{T} \\ & \wedge\ \forall X \in L.\, \lambda_A(X) = \{[X \mapsto \{\sigma_{\Sigma(A)}^{-1}\}]\} \\ & \wedge\ P_A = P \wedge (\mathrm{type} = A)\} \end{aligned}$$

该自动机对每个 $A \in \mathbf{T}$ 都有一条转移，用相应的幺元赋值更新其寄存器，然后检查谓词是否属于该类型，最后使用其幺元来创建元组。

我们证明，对于流 $\mathcal{S}$，在 Σ 上有 $[\![\mathcal{A}]\!](\mathcal{S}) = [\![\mathcal{A}']\!](\mathcal{S})$。设 $C = (i, j, \mu_\rho) \in [\![\mathcal{A}]\!](\mathcal{S})$ 是 $\mathcal{S} = e_1 \ldots e_n$ 上的一个复杂事件，且 $\mathcal{A}$ 在 $\mathcal{S}$ 上的一个接受运行具有以下形式：

$$\rho := q_i \xrightarrow{P_i/L_i} q_{i+1} \xrightarrow{P_{i+1}/L_{i+1}} \cdots \xrightarrow{P_j/L_j} q_{j+1}$$

使得对于每个 $X \in \mathbf{X}$，$q_{j+1} \in F$ 且 $\mu_\rho(X) = \{t_k \mid i \le k \le j \wedge X \in L_k\}$。那么，根据构造，我们可以找到 $\mathcal{A}'$ 在 $\mathcal{S}$ 上的一个接受运行，其形式如下：

$$\rho' := (q_i, \nu_i) \xrightarrow{\sigma_i, P_i'/\lambda_i} (q_{i+1}, \nu_{i+1}) \xrightarrow{\cdots} (q_j, \nu_j) \xrightarrow{\sigma_j, P_j'/\lambda_j} (q_{j+1}, \nu_{j+1})$$

其中，对于每个 $\ell \in \{i, \ldots, j\}$，$\sigma_\ell = \sigma_{\Sigma(e_\ell(\text{类型}))}$、$P_\ell' = P_\ell \wedge (\text{类型} = e_\ell(\text{类型}))$、$\lambda_\ell(X) = \{[X \mapsto \{\sigma_\Sigma^{-1}{}_{(e_\ell(\text{类型}))}\}]\}$ 以及每个 $X \in L_\ell$，ν_ℓ 均按相应方式定义。于是，根据 $\mathcal{A}'$ 转移的定义可知，它们与 $\mathcal{A}$ 具有相同的状态和谓词，但也会检查事件的类型；而且它们不再用 L_k 中的变量集合标记复杂事件中的事件，而是使用重命名来标记相同的事件。最后，可以看出 $\mu_\rho = \mu_{\rho'}$ 和 $C \in [\![\mathcal{A}']\!](\mathcal{S})$。

容易验证，反方向可用相同的论证得出。 □

定理 6.2 的证明

证明。我们通过对公式 φ 的语法进行归纳，按如下方式构造 ACEA $\mathcal{A}_\varphi = (Q, \Delta, q_0, F)$，从而证明该结果。注意，我们假设此构造针对特定模式 $\Sigma : \mathbf{T} \to$

$\mathcal{P}(\mathbf{A}_\Sigma)$，并且我们采用定理 6.1 中给出的相同定义 σ_A。我们将空赋值 $\sigma_\varnothing$ 定义为一种与输入无关、会清除所有寄存器的赋值。我们还假设每个自动机都有不同的寄存器集合，也就是说，如果自动机 $\mathcal{A}_{\psi_1}$ 的寄存器集合为 A_1，而 $\mathcal{A}_{\psi_2}$ 的寄存器集合为 A_2，那么 $A_1 \cap A_2 = \varnothing$。

证明通过对 ACEL 公式 φ 进行结构归纳来完成。我们从基例开始。$[\varphi = R]$。如果 $\varphi = R$，那么 $\mathcal{A}_\varphi$ 定义为

$$\mathcal{A}_\varphi = (\{p_1, p_2\}, \{(p_1, \sigma_{\Sigma(R)}, P_R, \underbrace{[R \mapsto \{\sigma^{-1}_{\Sigma(R)}\}]}_{\lambda}, p_2)\}, p_1, \{p_2\}),$$

其中，P_R 是包含所有类型为 R 的元组的谓词，定义如前；而 λ 是这样一个函数：对于变量 R，它将寄存器元组的属性重命名为其原始名称。更具体地说，$\lambda(R) = \{\sigma^{-1}_{\Sigma R()}\}$。直观来说，自动机将事件读入其寄存器，检查类型是否为 R，并输出相同的事件。我们通过将属性存储到 $\Sigma(R)$ 中的属性副本来实现这一点。我们要求满足此条件，以确保如果在构造的后续步骤中使用这些属性，那么新事件到达时不会覆盖它们。

我们将证明 $[\![\mathcal{A}_R]\!](\mathcal{S}) = [\![R]\!](\mathcal{S})$。设 $\mathcal{S} = e_1 \dots e_n$ 是模式 Σ 上的一个流。如果 $C \in [\![\mathcal{A}_R]\!](\mathcal{S})$，那么在 $\mathcal{S}$ 上的一个接受运行形式如下：

$$\rho := (q_i, \nu_i) \xrightarrow{\sigma_{\Sigma(R)}, P_R/\lambda} (q_{i+1}, \nu_{i+1})$$

对于某个 R，其中 $q_i = p_1$ 与 $q_{i+1} = p_2$。然后 $C = (i, i, \mu)$，其中 $\mu = [R \mapsto \{\!\{e_i\}\!\}]$。我们可以注意到，它也满足之前定义的语义，因此 $C \in [\![R]\!](\mathcal{S})$。反过来，如果$C \in [\![R]\!](\mathcal{S})$，那么 $C = (i, i, [R \mapsto \{\!\{e_i\}\!\}])$。鉴于 e_i(类型) $= R$ 且 $\mathcal{S}$ 满足模式 Σ，我们知道 Att(e_i) $= \Sigma(e_i$(类型))。那么，自动机在 $\mathcal{S}$ 上从位置 i 到 $i+1$ 的运行是 $\rho := (p_1, \nu_1) \xrightarrow{\sigma_{\Sigma(R)}, P_R/\lambda} (p_2, \nu_2)$。于是，显然有 $C \in [\![\mathcal{A}_R]\!](\mathcal{S})$。

我们继续讨论归纳情形。接下来，我们假设对于公式 ψ，存在一个 ACEA $\mathcal{A}_\psi$，其输出相同。

$[\varphi = \psi \text{ AS } X]$。如果 $\varphi = \psi \text{ AS } X$，那么 $\mathcal{A}_\varphi = (Q_\psi, \Delta_\varphi, q_{0\psi}, F_\psi)$，其中 Δ_φ 是将变量 X 添加到 Δ_ψ 的所有“标记”转移后得到的结果。形式化地，

$$\begin{aligned}\Delta_\varphi \;=\; & \{(p, \sigma, P, \lambda, q) \in \Delta_\psi \mid \forall Y \in \mathbf{X}.\ \lambda(Y) = \emptyset\} \\ \cup\; & \{(p, \sigma, P, \lambda', q) \mid (p, \sigma, P, \lambda, q) \in \Delta_\psi \ \wedge \\ & \qquad \lambda'(X) = \biguplus_{Z \in \mathbf{X}} \lambda(Z) \wedge \forall Y \neq X.\ \lambda'(Y) = \lambda(Y)\}.\end{aligned}$$

我们将证明 $[\![\mathcal{A}_\varphi]\!](\mathcal{S}) = [\![\psi \text{ AS } X]\!](\mathcal{S})$。如果 $C \in [\![\mathcal{A}_\varphi]\!](\mathcal{S})$，那么流 $\mathcal{S} = e_1 \dots e_n$ 上的一个接受运行为

$$\rho := (q_i, \nu_i) \xrightarrow{\sigma_i, P_i/\lambda_i} (q_{i+1}, \nu_{i+1}) \xrightarrow{\cdots} (q_j, \nu_j) \xrightarrow{\sigma_j, P_j/\lambda_j} (q_{j+1}, \nu_{j+1})$$

那么 $C = (i, j, \mu)$，其中 $\mu(X) = \biguplus_{Z \in \mathbf{X}} \mu(Z)$。根据定义，$C' \in [\![\mathcal{A}_\psi]\!](\mathcal{S})$ 在同一流 $\mathcal{S}$ 上也有相同的接受运行，但它不考虑变量 X；这是因为 $C' = (i, j, \mu')$，并且对于每个 $Y \neq X$，$\mu'(Y) = \mu(Y)$。我们可以注意到，它也满足前面定义的公式的语义，因此 $C \in [\![\psi \text{ AS } X]\!](\mathcal{S})$。

如果 $C \in [\![\psi \text{ AS } X]\!](\mathcal{S})$ 且 $\mathcal{S} = e_1 \dots e_n$，则根据定义：

$$C = (i, j, \mu \cup [X \mapsto \biguplus_{Z \in \mathbf{X}} \mu(Z)])$$

并且存在一个定义在同一流上的 $C' \in [\![\psi]\!]$，使得

$$C' = (i, j, \mu).$$

根据归纳法，自动机 $\mathcal{A}_\psi$ 在 $\mathcal{S}$ 上的一次运行将是

$$\rho := (q_i, \nu_i) \xrightarrow{\sigma_i, P_i/\lambda_i} (q_{i+1}, \nu_{i+1}) \xrightarrow{\cdots} (q_j, \nu_j) \xrightarrow{\sigma_j, P_j/\lambda_j} (q_{j+1}, \nu_{j+1})$$

使得 $C_\rho = (i, j, \mu) = C'$。然后，根据构造：

$$\rho' := (q_i, \nu_i) \xrightarrow{\sigma_i, P_i/\lambda'_i} (q_{i+1}, \nu_{i+1}) \xrightarrow{\cdots} (q_j, \nu_j) \xrightarrow{\sigma_j, P_j/\lambda'_j} (q_{j+1}, \nu_{j+1})$$

是 $\mathcal{A}_\varphi$ 与 $C_{\rho'} = C$ 的一次运行。由此我们得出 $C \in [\![\mathcal{A}_\varphi]\!](\mathcal{S})$。

$[\varphi = \psi\,\textsc{filter}\ \ X\,[P]]$。如果对于某个变量 X 和谓词 P，$\varphi = \psi\,\textsc{filter}\ \ X\,[P]$，那么 $\mathcal{A}_\varphi = (Q_\psi,\ \Delta_\varphi,\ q_{0\psi},\ F_\psi)$，其中 Δ_φ 定义如下：

$$\begin{array}{rcl}\Delta_\varphi & = & \{(p,\sigma,P',\lambda,q) \in \Delta_\psi \mid \lambda(X) = \emptyset\} \\ & \cup & \{(p,\sigma,P' \wedge \bigcap_{r\in\lambda(X)} r(P),\lambda,q) \mid (p,\sigma,P',\lambda,q) \in \Delta_\psi \wedge \lambda(X) \neq \emptyset\}.\end{array}$$

假设 $[\![\mathcal{A}_\psi]\!](\mathcal{S}) = [\![\psi]\!](\mathcal{S})$。我们将证明：

$$[\![\mathcal{A}_\varphi]\!](\mathcal{S}) = [\![\psi\ \ \textsc{filter}\ \ X\,[P]]\!](\mathcal{S}).$$

令 $\mathcal{S} = e_1 \ldots e_n$ 为一个流。如果 $C \in [\![\mathcal{A}_\varphi]\!](\mathcal{S})$，那么在 $\mathcal{S}$ 上的一个接受运行具有如下形式 ：

$$\rho := (q_i,\nu_i) \xrightarrow{\sigma_i,P_i/\lambda_i} (q_{i+1},\nu_{i+1}) \longrightarrow \cdots \longrightarrow (q_j,\nu_j) \xrightarrow{\sigma_j,P_j/\lambda_j} (q_{j+1},\nu_{j+1})$$

其中 $C_\rho = (i,\ j,\ \mu) = C$。根据构造，该自动机执行的操作与 $\mathcal{A}_\psi$ 相同，但转移中标记变量 X 的重命名必须满足谓词 P，因此 $\mu(X)$ 中的每个事件都满足P。最后，它满足公式的定义，并且 $C \in [\![\psi\,\textsc{filter}\ \ X\,[P]]\!]$。

反过来，若 $C \in [\![\varphi]\!](\mathcal{S})$，则根据定义，$C \in [\![\psi]\!](\mathcal{S})$ 且 $C(X) \models P$。由于 $C \in [\![\psi]\!](\mathcal{S})$，因此 $C \in [\![\mathcal{A}_\psi]\!](\mathcal{S})$。另一方面，$\mathcal{A}_\varphi$ 的运行与 $\mathcal{A}_\psi$ 的运行相同，但根据定义，标记为变量 X 的事件还必须满足谓词 P。最后，它满足自动机的定义，且 $C \in [\![\mathcal{A}_\varphi]\!](\mathcal{S})$。

$[\varphi = \psi_1$ 或 $\psi_2]$。若 $\varphi = \psi_1$ 或 ψ_2，则 $\mathcal{A}_\varphi$ 是 $\mathcal{A}_{\psi_1}$ 与 $\mathcal{A}_{\psi_2}$的并集，形式上 y,

$$\mathcal{A}_\varphi = (Q_{\psi_1} \cup Q_{\psi_2} \cup \{q_{0\varphi}\}, \Delta_\varphi, q_{0\varphi}, F_{\psi_1} \cup F_{\psi_2})$$

其中 $q_{0\varphi}$ 是一个新状态，且 $\Delta_\varphi = \Delta_{\psi_1} \cup \Delta_{\psi_2} \cup \{(q_{0\varphi},\ \sigma,\ P,\ \lambda,\ q) \mid (q_{0\psi_1},\ \sigma,\ P,\ \lambda,\ q) \in \Delta_{\psi_1} \vee (q_{0\psi_2},\ \sigma,\ P,\ \lambda,\ q) \in \Delta_{\psi_2}\}$。这里，不失一般性，我们假设 $\mathcal{A}_{\psi_1}$ 与 $\mathcal{A}_{\psi_2}$ 的状态集互不相交。
我们假设 $[\![\mathcal{A}_{\psi_1}]\!](\mathcal{S}) = [\![\psi_1]\!](\mathcal{S})$ 与 $[\![\mathcal{A}_{\psi_2}]\!](\mathcal{S}) = [\![\psi_2]\!](\mathcal{S})$，因此我们将证明 $[\![\mathcal{A}_\varphi]\!](\mathcal{S}) = [\![\psi_1$ 或 $\psi_2]\!](\mathcal{S})$。设 $C = (i,\ j,\ \mu) \in [\![\mathcal{A}_\varphi]\!](\mathcal{S})$，则在流 $\mathcal{S} = e_1 \ldots e_n$上的一个接受运行为

$$\rho := (q_i,\nu_i) \xrightarrow{\sigma_i,P_i/\lambda_i} (q_{i+1},\nu_{i+1}) \longrightarrow \cdots \longrightarrow (q_j,\nu_j) \xrightarrow{\sigma_j,P_j/\lambda_j} (q_{j+1},\nu_{j+1})$$

根据转移的定义，$\mathcal{A}_\varphi$ 可以选择执行 $\mathcal{A}_{\psi_1}$ 或 $\mathcal{A}_{\psi_2}$，因此 $C \in [\![\mathcal{A}_{\psi_1}]\!](\mathcal{S})$ 或 $C \in [\![\mathcal{A}_{\psi_2}]\!](\mathcal{S})$。最后，它满足公式的定义，并且 $C \in [\![\psi_1$ 或 $\psi_2]\!](\mathcal{S})$。

令 $C = (i,\ j,\ \mu) \in [\![\psi_1$ 或 $\psi_2]\!](\mathcal{S})$ 在流 $\mathcal{S}$ 上运行，这意味着 $C \in [\![\psi_1]\!](\mathcal{S})$ 或 $C \in [\![\psi_2]\!](\mathcal{S})$。我们知道，如果 $C \in [\![\psi_1]\!](\mathcal{S})$，那么 $C \in [\![\mathcal{A}_{\psi_1}]\!](\mathcal{S})$（对于 ψ_2 也类似）。接着，ACEA 的一次运行$\mathcal{A}_\varphi$ 会选择运行 $\mathcal{A}_{\psi_1}$ 或 $\mathcal{A}_{\psi_2}$，因此结果 C_φ 可以来自 $\mathcal{A}_{\psi_1}$ 或 $\mathcal{A}_{\psi_2}$，即 $C_\varphi \in [\![\mathcal{A}_{\psi_1}]\!](\mathcal{S})$ 或$C_\varphi \in [\![\mathcal{A}_{\psi_2}]\!](\mathcal{S})$。最后，它满足自动机的定义，并且 $C \in [\![\mathcal{A}_\varphi]\!](\mathcal{S})$。

$[\varphi = \psi_1$ 与 $\psi_2]$。对于接下来的构造，我们需要一些定义。对于两个赋值 σ_1 和 σ_2，其中 $\mathrm{Att}_{\mathrm{out}}(\sigma_1) \cap \mathrm{Att}_{\mathrm{out}}(\sigma_2) = \emptyset$，我们将并集 $\sigma_1 \uplus \sigma_2$ 定义为：

$$\sigma_1 \uplus \sigma_2(\mathbf{a}) = \begin{cases}\sigma_1(\mathbf{a}) & \text{if } \mathbf{a} \in \mathrm{Att}_{\mathrm{out}}(\sigma_1) \\ \sigma_2(\mathbf{a}) & \text{if } \mathbf{a} \in \mathrm{Att}_{\mathrm{out}}(\sigma_2)\end{cases}$$

对于每个 $\mathbf{a} \in \mathrm{Att}_{\mathrm{out}}(\sigma_1) \cup \mathrm{Att}_{\mathrm{out}}(\sigma_2)$。对于重命名包 R，我们将其模式定义为其中每个重命名所含全部属性构成的集合的多重集 r：

$$\mathtt{schema}(R) = \{\!\{\mathrm{dom}(r) \mid r \in R\}\!\}.$$

重命名包 R_1 和 R_2 之间的同构是一个双射 $f: R_1 \to R_2$，使得对于 R_1中的所有重命名 r，$\mathrm{dom}(r) = \mathrm{dom}(f(r))$；隐含地，这意味着以下关系成立：$\mathrm{schema}(R_1) = \mathrm{schema}(R_2)$。令 $\mathrm{ISO}(R_1, R_2)$ 表示 R_1 和 R_2之间所有同构的集合。

我们称 λ_1 与 λ_2 $\lambda_1 \equiv \lambda_2$ 等价，当且仅当它们之间存在同构，$\mathrm{dom}(\lambda_1) = \mathrm{dom}(\lambda_2)$，并且对于 $\mathbf{X}$ 中的所有 X，$\mathrm{schema}(\lambda_1(X)) = \mathrm{schema}(\lambda_2(X))$。

回到构造，如果 $\varphi = \psi_1$ 与 ψ_2，那么 $\mathcal{A}_\varphi$ 是 $\mathcal{A}_{\psi_1}$ 与 $\mathcal{A}_{\psi_2}$ 的交集，形式化地表示为 $\mathcal{A}_\varphi = (Q_{\psi_1} \times Q_{\psi_2}, \Delta_\varphi, (q_{0\psi_1}, q_{0\psi_2}), F_{\psi_1} \times F_{\psi_2})$，并且

$$\begin{aligned}\Delta_\varphi = \{((p_1,p_2), \sigma_1 \uplus \sigma_2, P_1 \wedge P_2 \wedge P_{\lambda_1,\lambda_2}, \lambda_1, (q_1,q_2)) \mid \\ (p_1,\sigma_1,P_1,\lambda_1,q_1) \in \Delta_{\psi_1} \ \wedge \ (p_2,\sigma_2,P_2,\lambda_2,q_2) \in \Delta_{\psi_2} \ \wedge \ \lambda_1 \equiv \lambda_2 \\ \wedge P_{\lambda_1,\lambda_2} = \bigwedge_{X \in \mathbf{X}} \bigvee_{f \in \mathrm{ISO}(\lambda_1(X),\lambda_2(X))} \bigwedge_{r \in \lambda_1(X)} \bigwedge_{a \in \mathrm{dom}(r)} P_{r(a)=[f(r)](a)}\}\end{aligned}$$

我们假设不失一般性， $\mathcal{A}_{\psi_1}$ 和 $\mathcal{A}_{\psi_2}$ 的赋值集合互不相交。设 $\mathcal{S} = e_1 \ldots e_n$ 为一个流，使得 $[\![\mathcal{A}_{\psi_1}]\!](\mathcal{S}) = [\![\psi_1]\!](\mathcal{S})$且 $[\![\mathcal{A}_{\psi_2}]\!](\mathcal{S}) = [\![\psi_2]\!](\mathcal{S})$。因此，我们将证明 $[\![\mathcal{A}_\varphi]\!](\mathcal{S}) = [\![\psi_1 \text{ 与 } \psi_2]\!](\mathcal{S})$。

设 $C = (i,j,\mu) \in [\![\mathcal{A}_\varphi]\!](\mathcal{S})$，那么在流 $\mathcal{S}$ 上的一个接受运行具有如下形式

$$\rho := (q_i,\nu_i) \xrightarrow{\sigma_i,P_i/\lambda_i} (q_{i+1},\nu_{i+1}) \longrightarrow \cdots \longrightarrow (q_j,\nu_j) \xrightarrow{\sigma_j,P_j/\lambda_j} (q_{j+1},\nu_{j+1})$$

根据转移的定义， $\mathcal{A}_\varphi$ 同时执行 $\mathcal{A}_{\psi_1}$ 和 $\mathcal{A}_{\psi_2}$，维护两个寄存器并检查两个谓词，因此 $C \in [\![\mathcal{A}_{\psi_1}]\!](\mathcal{S})$ 且 $C \in [\![\mathcal{A}_{\psi_2}]\!](\mathcal{S})$。它还会在转移中检查，对于每个变量 X，都存在一个双射，使得 $\lambda_1(X)$的重命名中有一个与 $\lambda_2(X)$等价，也就是说，它们写入相同属性及相同值。由于这些重命名在转移中等价，因此它只考虑来自 λ_1 的重命名来进行写入。最后，它满足公式的定义，并且 $C \in [\![\psi_1 \text{ 与 } \psi_2]\!](\mathcal{S})$。

令 $C = (i,\ j,\ \mu) \in [\![\psi_1 \text{与 } \psi_2]\!](\mathcal{S})$，这意味着 $C \in [\![\psi_1]\!](\mathcal{S})$ 和 $C \in [\![\psi_2]\!](\mathcal{S})$。我们知道，如果 $C \in [\![\psi_1]\!](\mathcal{S})$，那么 $C \in [\![\mathcal{A}_{\psi_1}]\!](\mathcal{S})$（对于 ψ_2 也是类似的）。然后，ACEA 的一次运行 $\mathcal{A}_\varphi$ 同时 $\mathcal{A}_{\psi_1}$ 运行 $\mathcal{A}_{\psi_2}$，因此结果 C_φ 来自 $\mathcal{A}_{\psi_1}$ 和 $\mathcal{A}_{\psi_2}$，即 $C_\varphi \in [\![\mathcal{A}_{\psi_1}]\!](\mathcal{S})$ 和 $C_\varphi \in [\![\mathcal{A}_{\psi_2}]\!](\mathcal{S})$。最后，它满足自动机的定义，并且 $C \in [\![\mathcal{A}_\varphi]\!](\mathcal{S})$。

$[\varphi = \psi_1;\ \psi_2]$。若 $\varphi = \psi_1;\ \psi_2$，则 $\mathcal{A}_\varphi$ 是考虑 $\mathcal{A}_{\psi_1}$之后的 $\mathcal{A}_{\psi_2}$ 的 ACEA，形式上，$\mathcal{A}_\varphi = (Q_{\psi_1} \cup Q_{\psi_2}, \Delta_\varphi, q_{0\psi_1}, F_{\psi_2})$，其中

$$\begin{aligned}\Delta_\varphi = &\Delta_{\psi_1} \cup \Delta_{\psi_2} \cup \{(q_{0\psi_2}, \sigma_\varnothing, \mathtt{TRUE}, \emptyset, q_{0\psi_2})\} \cup \\ &\{(p,\sigma,P,\lambda,q_{0\psi_2}) \mid \exists q' \in F_{\psi_1}.(p,\sigma,P,\lambda,q') \in \Delta_{\psi_1}\}.\end{aligned}$$

这里，不失一般性，我们假设 $\mathcal{A}_{\psi_1}$ 和 $\mathcal{A}_{\psi_2}$ 的状态集互不相交。
在下文中。部分中，我们假设 $\nu_\varnothing$ 是空事件，并且 $[\![\mathcal{A}_{\psi_1}]\!](\mathcal{S}) = [\![\psi_1]\!](\mathcal{S})$和 $[\![\mathcal{A}_{\psi_2}]\!](\mathcal{S}) = [\![\psi_2]\!](\mathcal{S})$对于 $\mathcal{S} = e_1 \ldots e_n$上。因此，我们证明 $[\![\mathcal{A}_\varphi]\!](\mathcal{S}) = [\![\psi_1;\ \psi_2]\!](\mathcal{S})$。
设 $C = (i,\ j,\ \mu) \in [\![\mathcal{A}_\varphi]\!](\mathcal{S})$，则在 $\mathcal{S}$ 上的接受运行形式如下：

$$\rho := (q_i,\nu_i) \xrightarrow{\sigma_i,P_i/\lambda_i} (q_{i+1},\nu_{i+1}) \longrightarrow \cdots \longrightarrow (q_j,\nu_j) \xrightarrow{\sigma_j,P_j/\lambda_j} (q_{j+1},\nu_{j+1})$$

根据转移的定义， $\mathcal{A}_\varphi$ 首先执行 $\mathcal{A}_{\psi_1}$，执行完毕后，它会在状态$q_{0\psi_2}$ 中等待，直到执行$\mathcal{A}_{\psi_2}$，因此我们可以将该运行分解为：

$$\rho := \rho_1 \xrightarrow{\sigma_{k+1},P_{k+1}/\lambda_{k+1}} (q_{0\psi_2},\nu_\varnothing) \longrightarrow \cdots \longrightarrow (q_{0\psi_2},\nu_\varnothing) \xrightarrow{\sigma_{l-1},P_{l-1}/\lambda_{l-1}} \rho_2$$

其中， ρ_1 和 ρ_2 的形式如下：

$$\rho_1 := (q_i,\nu_i) \xrightarrow{\sigma_i,P_i/\lambda_i} (q_{i+1},\nu_{i+1}) \longrightarrow \cdots \longrightarrow (q_k,\nu_k) \xrightarrow{\sigma_k,P_k/\lambda_k} (q_{k+1},\nu_{k+1})$$

$$\rho_2 := (q_l,\nu_l) \xrightarrow{\sigma_l,P_l/\lambda_l} (q_{l+1},\nu_{l+1}) \longrightarrow \cdots \longrightarrow (q_j,\nu_j) \xrightarrow{\sigma_j,P_j/\lambda_j} (q_{j+1},\nu_{j+1})$$

其中， ρ_1 是 $\mathcal{A}_{\psi_1}$ 的一个接受运行， ρ_2 是 $\mathcal{A}_{\psi_2}$的一个接受运行，因此有 $C_1 \in [\![\mathcal{A}_{\psi_1}]\!](\mathcal{S})$ 和$C_2 \in [\![\mathcal{A}_{\psi_2}]\!](\mathcal{S})$。然后，$C_1 \in [\![\psi_1]\!](\mathcal{S})$、 $C_2 \in [\![\psi_2]\!](\mathcal{S})$ 与 C 是两个结果的并集。最后，它满足公式的定义以及 $C \in [\![\psi_1;\ \psi_2]\!](\mathcal{S})$。

令 $C = (i,j,\mu) \in [\![\psi_1;\ \psi_2]\!](\mathcal{S})$。这意味着 C 是 $C_1 \in [\![\psi_1]\!](\mathcal{S})$ 和 $C_2 \in [\![\psi_2]\!](\mathcal{S})$ 的并集，并且 $\mathrm{end}(C_1) < \mathrm{start}(C_2)$。我们知道，如果 $C_1 \in [\![\psi_1]\!](\mathcal{S})$，那么 $C_1 \in [\![\mathcal{A}_{\psi_1}]\!](\mathcal{S})$（对 ψ_2 也同样成立）。然后，我们可以运行 $\mathcal{A}_{\psi_1}$，在 $q_{0\psi_2}$ 中等待，再执行 $\mathcal{A}_{\psi_2}$，从而得到两个复杂事件的并集。最后，它满足 ACEA 的定义以及 $C \in [\![\mathcal{A}_\varphi]\!](\mathcal{S})$。

$[\varphi = \psi_1 : \ \psi_2]$。如果 $\varphi = \psi_1 : \ \psi_2$，那么 $\mathcal{A}_\varphi$ 是紧接在 $\mathcal{A}_{\psi_1}$ 之后考虑 $\mathcal{A}_{\psi_2}$ 的 ACEA，形式化地说，$\mathcal{A}_\varphi = (Q_{\psi_1} \cup Q_{\psi_2}, \Delta_\varphi, q_{0\psi_1}, F_{\psi_2})$，其中

$$\Delta_\varphi = \Delta_{\psi_1} \cup \Delta_{\psi_2} \cup \{(p, \sigma, P, \lambda, q_{0\psi_2}) \mid (p, \sigma, P, \lambda, q_f) \in \Delta_{\psi_1} \wedge q_f \in F_{\psi_1}\}$$

在此，不失一般性，我们假设 $\mathcal{A}_{\psi_1}$ 和 $\mathcal{A}_{\psi_2}$ 的状态集互不相交。

与前面的构造类似，我们假设 $\nu_\varnothing$ 是空事件，$[\![\mathcal{A}_{\psi_1}]\!](\mathcal{S}) = [\![\psi_1]\!](\mathcal{S})$ 且$[\![\mathcal{A}_{\psi_2}]\!](\mathcal{S}) = [\![\psi_2]\!](\mathcal{S})$，因此我们将证明 $[\![\mathcal{A}_\varphi]\!](\mathcal{S}) = [\![\psi_1 : \ \psi_2]\!](\mathcal{S})$。

令 $C = (i, \ j, \ \mu) \in [\![\mathcal{A}_\varphi]\!](\mathcal{S})$，则在 $\mathcal{S}$ 上的一个接受运行为：

$$\rho := (q_i, \nu_i) \xrightarrow{\sigma_i, P_i/\lambda_i} (q_{i+1}, \nu_{i+1}) \longrightarrow \cdots \longrightarrow (q_j, \nu_j) \xrightarrow{\sigma_j, P_j/\lambda_j} (q_{j+1}, \nu_{j+1})$$

根据转移的定义，$\mathcal{A}_\varphi$ 先执行 $\mathcal{A}_{\psi_1}$，完成后立即执行 $\mathcal{A}_{\psi_2}$，因此我们可以将该运行分解为：

$$\rho := \rho_1 \xrightarrow{\sigma_k, P_k/\lambda_k} \rho_2$$

$$\rho_1 := (q_i, \nu_i) \xrightarrow{\sigma_i, P_i/\lambda_i} (q_{i+1}, \nu_{i+1}) \longrightarrow \cdots \longrightarrow (q_{k-1}, \nu_{k-1}) \xrightarrow{\sigma_{k-1}, P_{k-1}/\lambda_{k-1}} (q_k, \nu_k)$$

$$\rho_2 := (q_{k+1}, \nu_{k+1}) \xrightarrow{\sigma_{k+1}, P_{k+1}/\lambda_{k+1}} (q_{k+2}, \nu_{k+2}) \longrightarrow \cdots \longrightarrow (q_j, \nu_j) \xrightarrow{\sigma_j, P_j/\lambda_j} (q_{j+1}, \nu_{j+1})$$

其中，ρ_1 与转移 k 是 $\mathcal{A}_{\psi_1}$ 的一个接受运行（若认为原文中的转移 k 会到达一个终态），而 ρ_2 是 $\mathcal{A}_{\psi_2}$ 的一个接受运行，因此我们有 $C_1 \in [\![\mathcal{A}_{\psi_1}]\!](\mathcal{S})$和$C_2 \in [\![\mathcal{A}_{\psi_2}]\!](\mathcal{S})$。于是，$C_1 \in [\![\psi_1]\!](\mathcal{S})$、$C_2 \in [\![\psi_2]\!](\mathcal{S})$，且 C 是两个结果的并集。最后，它满足公式和 $C \in [\![\psi_1$ 的定义：$\psi_2]\!](\mathcal{S})$。

设 $C = (i, \ j, \ \mu) \in [\![\psi_1 : \ \psi_2]\!](\mathcal{S})$，这意味着 C是 $C_1 \in [\![\psi_1]\!](\mathcal{S})$和 $C_2 \in [\![\psi_2]\!](\mathcal{S})$的并集，并且$\mathsf{end}(C_1) + 1 = \mathsf{start}(C_2)$。我们知道，如果 $C_1 \in [\![\psi_1]\!](\mathcal{S})$，那么 $C_1 \in [\![\mathcal{A}_{\psi_1}]\!](\mathcal{S})$（对于$\psi_2$也同理）。接下来，我们可以运行$\mathcal{A}_{\psi_1}$，然后紧接着执行$\mathcal{A}_{\psi_2}$，将$\mathcal{A}_{\psi_1}$ 的最后一个转移连接到 $\mathcal{A}_{\psi_2}$的第一个状态，从而得到两个复杂事件的并集。最后，它满足自动机和 $C \in [\![\mathcal{A}_\varphi]\!](\mathcal{S})$的定义。

$[\varphi = \psi+]$。如果 $\varphi = \psi+$，那么 $\mathcal{A}_\varphi = (Q_\psi \cup \{q_{new}\}, \ \Delta_\varphi, \ q_{0\psi}, \ F_\psi)$，其中 q_{new} 是一个新状态，而 Δ_φ 的形式化定义如下：

$$\begin{array}{rcl} \Delta_\varphi & = & \Delta_\psi \\ & \cup & \{(p, \sigma, P, \lambda, q_{new}) \mid \exists q' \in F_\psi . (p, \sigma, P, \lambda, q') \in \Delta_\psi\} \\ & \cup & \{(q_{new}, \sigma_\varnothing, \mathtt{TRUE}, \emptyset, q_{new})\} \\ & \cup & \{(q_{new}, \sigma, P, \lambda, p) \mid (q_{0\psi}, \sigma, P, \lambda, p) \in \Delta_\psi\}. \end{array}$$

我们假设 $\nu_\varnothing$ 是空事件。我们将证明 $[\![\mathcal{A}_\varphi]\!](\mathcal{S}) = [\![\psi+]\!](\mathcal{S})$。设 $C = (i, \ j, \ \mu) \in [\![\mathcal{A}_\varphi]\!](\mathcal{S})$，那么在 $\mathcal{S}$ 上的一个接受运行形式如下：

$$\rho := (q_i, \nu_i) \xrightarrow{\sigma_i, P_i/\lambda_i} (q_{i+1}, \nu_{i+1}) \longrightarrow \cdots \longrightarrow (q_j, \nu_j) \xrightarrow{\sigma_j, P_j/\lambda_j} (q_{j+1}, \nu_{j+1})$$

那么，根据转移的定义，$\mathcal{A}_\varphi$ 执行 $\mathcal{A}_\psi$；执行完后，它会停留在状态 q_{new}，直到再次执行 $\mathcal{A}_\psi$，因此我们可以将该运行分解如下：

$$\rho := \rho'_1 \longrightarrow \cdots (q_{new}, \nu_\varnothing) \longrightarrow \rho'_2 \longrightarrow \cdots (q_{new}, \nu_\varnothing) \longrightarrow \cdots \rho'_l$$

其中，每个 ρ'_k 都是 $\mathcal{A}_\psi$的一个接受运行，满足 $C'_k \in [\![\mathcal{A}_\psi]\!](\mathcal{S})$，因此 $C'_k \in [\![\psi]\!](\mathcal{S})$。然后，$C$ 是所有已执行的 $\mathcal{A}_\psi$ 运行所得结果的并集。最后，它满足该公式的定义，且 $C \in [\![\psi+]\!](\mathcal{S})$。

设 $C = (i, \ j, \ \mu) \in [\![\psi+]\!](\mathcal{S})$ 在流 $\mathcal{S}$ 上，这意味着 C是多次执行查询 ψ 所得到的复杂事件的并集，每次执行依次进行。我们知道，如果每个 $C'_k \in [\![\psi]\!](\mathcal{S})$，那么$C_k \in [\![\mathcal{A}_\psi]\!](\mathcal{S})$。然后，我们可以运行 $\mathcal{A}_\psi$，在 q_{new} 中等待，再次执行 $\mathcal{A}_\psi$，如此多次，以获得每个 C'_k中的所有复杂事件的并集。最后，它满足自动机的定义，并且$C \in [\![\mathcal{A}_\varphi]\!](\mathcal{S})$。

$[\varphi = \psi\oplus]$。如果 $\varphi = \psi\oplus$，那么我们定义 $\mathcal{A}_\varphi = (Q_\psi \cup \{q_{new}\}, \ \Delta_\varphi, \ q_{0\psi}, \ F_\psi)$，其中 q_{new} 是一个新状态，转移关系 Δ 定义如下：

$$\begin{array}{rcl} \Delta_\varphi & = & \Delta_\psi \\ & \cup & \{(p, \sigma, P, \lambda, q_{new}) \mid \exists q' \in F_\psi . (p, \sigma, P, \lambda, q') \in \Delta_\psi\} \\ & \cup & \{(q_{new}, \sigma, P, \lambda, p) \mid (q_{0\psi}, \sigma, P, \lambda, p) \in \Delta_\psi\}. \end{array}$$

接下来，我们将证明 $[\![\mathcal{A}_\varphi]\!](\mathcal{S})$ 在流 $\mathcal{S}=e_1\dots e_n$ 上的 $=[\![\psi\oplus]\!](\mathcal{S})$。设 $C=(i,\ j,\ \mu)\in[\![\mathcal{A}_\varphi]\!](\mathcal{S})$，那么 $\mathcal{S}$ 上的接受运行形式如下：

$$\rho:=(q_i,\nu_i)\xrightarrow{\sigma_i,P_i/\lambda_i}(q_{i+1},\nu_{i+1})\xrightarrow{\cdots}(q_j,\nu_j)\xrightarrow{\sigma_j,P_j/\lambda_j}(q_{j+1},\nu_{j+1})$$

根据转移的定义，$\mathcal{A}_\varphi$ 执行 $\mathcal{A}_\psi$，完成后再次执行 $\mathcal{A}_\psi$，因此我们可以将运行分解为：

$$\rho:=\rho'_1\xrightarrow{\cdots}\rho'_2\xrightarrow{\cdots}\dots\xrightarrow{\cdots}\rho'_l$$

其中每个 ρ'_k 都是 $\mathcal{A}_\psi$的接受运行，且有 $C'_k\in[\![\mathcal{A}_\psi]\!](\mathcal{S})$，因此 $C'_k\in[\![\psi]\!](\mathcal{S})$。然后，$C$ 是 $\mathcal{A}_\psi$ 在前一个运行刚刚完成后执行的每个运行结果的并集。最后，它满足公式的定义，并且 $C\in[\![\psi\oplus]\!](\mathcal{S})$。

设 $C=(i,\ j,\ \mu)\in[\![\psi\oplus]\!](\mathcal{S})$，这意味着 C是多次执行查询ψ 所得到的复杂事件的并集，每次执行都紧接在上一次之后。我们知道，如果每个 $C'_k\in[\![\psi]\!](\mathcal{S})$，那么 $C_k\in[\![\mathcal{A}_\psi]\!](\mathcal{S})$。然后，我们可以多次执行 $\mathcal{A}_\psi$，随后再次执行 $\mathcal{A}_\psi$，从而得到每个C'_k中的所有复杂事件的并集。最后，它满足自动机的定义，并且 $C\in[\![\mathcal{A}_\varphi]\!](\mathcal{S})$。

$[\varphi=\pi_L(\psi)]$。若对于某个 $L\subseteq\mathbf{X}$，$\varphi=\pi_L(\psi)$，则 $\mathcal{A}_\varphi=(Q_\psi,\ \Delta_\varphi,\ q_{0\psi},\ F_\psi)$，其中 Δ_φ 是仅考虑与 Δ_ψ 中每个转移的 L 内某个变量相关联的重命名后得到的结果。形式化地，

$$\begin{aligned}\Delta_\varphi\ =\ \{(p,\sigma,P,\lambda',q)\ \mid\ &(p,\sigma,P,\lambda,q)\in\Delta_\psi\ \wedge\\ &\forall X\in L.\ \lambda'(X)=\lambda(X)\ \wedge\ \forall X\notin L.\ \lambda'(X)=\emptyset\}\end{aligned}$$

固定一个流 $\mathcal{S}=e_1\dots e_n$，并假设 $[\![\mathcal{A}_\psi]\!](\mathcal{S})\ =[\![\psi]\!](\mathcal{S})$。接下来，我们将证明$[\![\mathcal{A}_\varphi]\!](\mathcal{S})\ =[\![\pi_L(\psi)]\!](\mathcal{S})$。

设 $C\in[\![\mathcal{A}_\varphi]\!](\mathcal{S})$，则在 $\mathcal{S}$ 上的一个接受运行形式如下：

$$\rho:=(q_i,\nu_i)\xrightarrow{\sigma_i,P_i/\lambda_i}(q_{i+1},\nu_{i+1})\xrightarrow{\cdots}(q_j,\nu_j)\xrightarrow{\sigma_j,P_j/\lambda_j}(q_{j+1},\nu_{j+1})$$

于是 $C=(i,\ j,\ \mu)$，其中 $\mathsf{dom}(\mu)=\{X_1,\dots,X_k\}\subseteq L$。根据定义，该自动机与 $\mathcal{A}_\psi$ 执行相同操作，但只考虑那些标记了 $X_1,\dots,X_k\in L$ 中变量的转移所涉及的重命名。最后，它满足公式的定义，并且 $C\in[\![\pi_L(\psi)]\!](\mathcal{S})$。

如果 $C\in[\![\pi_L(\psi)]\!](\mathcal{S})$，那么根据定义，存在 $C'\in[\![\psi]\!](\mathcal{S})$使得 $C=\pi_L(C')$。这意味着 C只包含属于变量 $X_i\in L$的事件。由于 $C'\in[\![\psi]\!](\mathcal{S})$，因此 $C'\in[\![\mathcal{A}_\psi]\!](\mathcal{S})$。另一方面，$\mathcal{A}_\varphi$ 的一次运行与 $\mathcal{A}_\psi$ 的一次运行执行相同的操作，但只考虑标记在变量 $X_i\in L$ 中的事件。最后，它满足自动机的定义，且 $C\in[\![\mathcal{A}_\varphi]\!](\mathcal{S})$。

$[\varphi\ =\ \pi_{X(\mathbf{a}_1,\dots,\mathbf{a}_k)}(\psi)]$。如果$\varphi\ =\ \pi_{X(\mathbf{a}_1,\dots,\mathbf{a}_k)}(\psi)$，那么 $\mathcal{A}_\varphi\ =\ (Q_\psi,\ \Delta_\varphi,\ q_{0\psi},\ F_\psi)$，其中 Δ_φ 是考虑与每个转移的变量 X中属性$\mathbf{a}_1$、.....、$\mathbf{a}_k$相关联的重命名后得到的结果。形式化地说，即

$$\begin{aligned}\Delta_\varphi\ =\big\{(p,\sigma,P,\lambda',q)\ |(p,\sigma,P,\lambda,q)\in\Delta_\psi\wedge\forall Y\neq X.\lambda'(Y)=\lambda(Y)\\ \wedge\lambda'(X)=\{\!\{\ r'\ \mid\ r\in\lambda(X)\\ \wedge\ \mathrm{dom}(r')=\mathrm{dom}(r)\cap\{\mathbf{a}_1,\dots,\mathbf{a}_k\}\\ \wedge\forall\mathbf{a}_i\in\mathrm{dom}(r').\ r'(\mathbf{a}_i)=r(\mathbf{a}_i)\}\!\}\big\}.\end{aligned}$$

对于流 $\mathcal{S}\ =\ e_1\dots e_n$，假设 $[\![\mathcal{A}_\psi]\!](\mathcal{S})\ =\ [\![\psi]\!](\mathcal{S})$。接下来，我们将证明 $[\![\mathcal{A}_\varphi]\!](\mathcal{S})\ =[\![\pi_{X(\mathbf{a}_1,\dots,\mathbf{a}_k)}(\psi)]\!](\mathcal{S})$。

设 $C\in[\![\mathcal{A}_\varphi]\!](\mathcal{S})$，则在 $\mathcal{S}$ 上的一个接受运行形式如下：

$$\rho:=(q_i,\nu_i)\xrightarrow{\sigma_i,P_i/\lambda_i}(q_{i+1},\nu_{i+1})\xrightarrow{\cdots}(q_j,\nu_j)\xrightarrow{\sigma_j,P_j/\lambda_j}(q_{j+1},\nu_{j+1})$$

其中 $C_\rho=C=(i,\ j,\ \mu)$。根据定义，该自动机执行的操作与 $\mathcal{A}_\psi$ 相同，但它只考虑转移中那些标记变量 X 且具有属性 $\{\mathbf{a}_1,\ .....,\ \mathbf{a}_k\}$ 的重命名，并保留其他变量的重命名。最后，它满足公式的定义以及 $C\in[\![\pi_{X(\mathbf{a}_1,\dots,\mathbf{a}_k)}(\psi)]\!](\mathcal{S})$。

如果 $C\in[\![\pi_{X(\mathbf{a}_1,\dots,\mathbf{a}_k)}(\psi)]\!](\mathcal{S})$ 且 $\mathcal{S}=e_1\dots e_n$，那么根据定义，存在 $C'\in[\![\psi]\!](\mathcal{S})$，使得 C只考虑变量 X 中事件的属性 $\{\mathbf{a}_1,\dots,\mathbf{a}_k\}$，并保持其他变量相对于 C' 时的状态不变。由于 $C'\in[\![\psi]\!](\mathcal{S})$，因此 $C'\in[\![\mathcal{A}_\psi]\!](\mathcal{S})$。另一方面，$\mathcal{A}_\varphi$ 的一次运行所做的

与 $\mathcal{A}_\psi$ 的一次运行相同，但只考虑标记在变量 X 中的事件的属性 $\{\mathbf{a}_1, \ldots, \mathbf{a}_k\}$，并保持其他变量的状态不变。最后，它满足自动机的定义以及 $C \in [\![\mathcal{A}_\varphi]\!](\mathcal{S})$。

$[\varphi = \text{聚合}_{Y[\mathbf{b} \leftarrow \otimes X(\mathbf{a})]}(\psi)]$。设 $(\mathbb{D}, \otimes, \mathbb{O})$ 为与该聚合相关联的幺半群。不失一般性，我们假设 $\mathcal{A}_\psi$ 的初始状态 $q_{0\psi}$ 只有出转移，而 F_ψ 中的终态只有入转移，即对于每个转移 $(p, \sigma, P, \lambda, q) \in \Delta_\psi$，都有 $q \neq q_{0\psi}$ 和 $p \notin F_\psi$。否则，我们可以添加新的状态，并利用 ACEA 的非确定性来满足这些要求。此外，为了简化起见，我们假设从初始状态出发，一步之内无法到达终态；也就是说，对于每个转移 $(q_{0\psi}, \sigma, P, \lambda, q) \in \Delta_\psi$，都有 $q \notin F_\psi$。如果出现这种情况，可以很容易地扩展下面的构造以涵盖此情形。

因此，假设 $\varphi = \text{Agg}_{Y[\mathbf{b} \leftarrow \otimes X(\mathbf{a})]}(\psi)$。我们定义 $\mathcal{A}_\varphi = (Q_\psi, \Delta_\varphi, q_{0\psi}, F_\psi)$，其中新的 Δ_φ 是通过向 Δ_ψ 添加一个全新的寄存器 $\mathbf{c}$ 扩展后得到的结果。该寄存器在 Δ_ψ 中的任何位置都未被使用，并将用于聚合。具体而言，我们定义：

$$\Delta_\varphi \;=\; \Delta_\varphi^{\text{init}} \uplus \Delta_\varphi^{\text{agg}} \uplus \Delta_\varphi^{\text{out}}$$

其中，每组转移定义如下。首先，集合 $\Delta_\varphi^{\text{init}}$ 负责初始化寄存器 c，以执行聚合。

$$\begin{aligned}
\Delta_\varphi^{\text{init}} = \{(q_{0\psi}, \sigma', P, \lambda, q) \mid\ & (q_{0\psi}, \sigma, P, \lambda, q) \in \Delta_\psi \wedge \lambda(X) = \emptyset \wedge \sigma'(\mathbf{c}) = \mathbb{O} \\
& \wedge \forall \mathbf{d} \neq \mathbf{c}.\ \sigma'(\mathbf{d}) = \sigma(\mathbf{d})\} \\
\cup \{(q_{0\psi}, \sigma', P, \lambda, q) \mid & (q_{0\psi}, \sigma, P, \lambda, q) \in \Delta_\psi \wedge \lambda(X) \neq \emptyset \\
& \wedge \sigma'(\mathbf{c}) = \begin{cases} \texttt{NULL} & \text{if } \exists r \in \lambda(X).\ r(\mathbf{a}) = \texttt{NULL} \\ \mathbf{c} & \text{otherwise} \end{cases} \\
& \wedge \forall \mathbf{d} \neq \mathbf{c}.\ \sigma'(\mathbf{d}) = \sigma(\mathbf{d})\}
\end{aligned}$$

其次，转移集合 $\Delta_\varphi^{\text{agg}}$ 负责在运行过程中、到达终态之前执行聚合。

$$\begin{aligned}
\Delta_\varphi^{\text{agg}} = \{(p, \sigma', P, \lambda, q) \mid\ & (p, \sigma, P, \lambda, q) \in \Delta_\psi \wedge \lambda(X) = \emptyset \wedge q \notin F \wedge \sigma'(\mathbf{c}) = \mathbf{c} \\
& \wedge \forall \mathbf{d} \neq \mathbf{c}.\ \sigma'(\mathbf{d}) = \sigma(\mathbf{d})\} \\
\cup \{(p, \sigma', P, \lambda, q) \mid & (p, \sigma, P, \lambda, q) \in \Delta_\psi \wedge \lambda(X) \neq \emptyset \wedge q \notin F \\
& \wedge \sigma'(\mathbf{c}) = \begin{cases} \texttt{NULL} & \text{if } \exists r \in \lambda(X).\ r(\mathbf{a}) = \texttt{NULL} \\ \mathbf{c} \otimes \bigotimes_{\forall r \in \lambda(X)} \sigma(r(\mathbf{a})) & \text{otherwise} \end{cases} \\
& \wedge \forall \mathbf{d} \neq \mathbf{c}.\ \sigma'(\mathbf{d}) = \sigma(\mathbf{d})\}
\end{aligned}$$

最后，集合 $\Delta_\varphi^{\text{out}}$ 包含所有到达终态的转移，以生成所需的聚合输出。

$$\begin{aligned}
\Delta_\varphi^{\text{out}} = \{(p, \sigma', P, \lambda', q) \mid\ & (p, \sigma, P, \lambda, q) \in \Delta_\psi \wedge \lambda(X) = \emptyset \wedge q \in F \wedge \sigma'(\mathbf{c}) = \mathbf{c} \\
& \wedge \forall \mathbf{d} \neq \mathbf{c}.\ \sigma'(\mathbf{d}) = \sigma(\mathbf{d}) \\
& \wedge \lambda'(Y) = \lambda(Y) \cup \{[\mathbf{b} \mapsto \mathbf{c}]\} \\
& \wedge \forall Z \neq Y.\ \lambda'(Z) = \lambda(Z)\} \\
\cup \{(p, \sigma', P, \lambda', q) \mid & (p, \sigma, P, \lambda, q) \in \Delta_\psi \wedge \lambda(X) \neq \emptyset \wedge q \in F \\
& \wedge \sigma'(\mathbf{c}) = \begin{cases} \texttt{NULL} & \text{if } \exists r \in \lambda(X).\ r(\mathbf{a}) = \texttt{NULL} \\ \mathbf{c} \otimes \bigotimes_{\forall r \in \lambda(X)} \sigma(r(\mathbf{a})) & \text{otherwise} \end{cases} \\
& \wedge \forall \mathbf{d} \neq \mathbf{c}.\ \sigma'(\mathbf{d}) = \sigma(\mathbf{d}) \\
& \wedge \lambda'(Y) = \lambda(Y) \cup \{[\mathbf{b} \mapsto \mathbf{c}]\} \\
& \wedge \forall Z \neq Y.\ \lambda'(Z) = \lambda(Z)\}
\end{aligned}$$

我们假设 $[\![\mathcal{A}_\psi]\!](\mathcal{S}) = [\![\psi]\!](\mathcal{S})$。接下来，我们将证明

$$[\![\mathcal{A}_\varphi]\!](\mathcal{S}) = [\![\texttt{Agg}_{Y[\mathbf{b} \leftarrow \otimes X(\mathbf{a})]}(\psi)]\!](\mathcal{S}).$$

设 $C \in [\![\mathcal{A}_\varphi]\!](\mathcal{S})$，那么流 $\mathcal{S} = e_1 \dots e_n$ 上的一个接受运行是

$$\rho := (q_i, \nu_i) \xrightarrow{\sigma_i, P_i/\lambda_i} (q_{i+1}, \nu_{i+1}) \xrightarrow{\cdots} (q_j, \nu_j) \xrightarrow{\sigma_j, P_j/\lambda_j} (q_{j+1}, \nu_{j+1})$$

与 $C = C_\rho = (i,\ j,\ \mu)$。根据定义，自动机 $\mathcal{A}_\varphi$ 与 $\mathcal{A}_\psi$执行相同的操作，但所有转移中都新增了一个寄存器 $\mathbf{c}$ 。在第一次转移中，如果 $\lambda_0(X) \neq \emptyset$ ，则将 $\mathbf{c}$ 设置为 O；否则设置为 $\mathbf{c}$。随后，对于每个满足 $\lambda_i(X) \neq \emptyset$ 的转移，它会更改 σ_i ，以使用事件中的值和重命名更新 $\mathbf{c}$ ；对于其他转移，则保持 $\mathbf{c}$不变。此外，它还会在指向终态的转移中添加一个新的重命名，将属性 $\mathbf{c}$ 的值放入属性 $\mathbf{b}$中。总之，它满足公式和$C \in [\![\mathrm{Agg}^{Y[\mathbf{b} \leftarrow \otimes X(\mathbf{a})]}(\psi)]\!]$的定义。

反过来，如果 $C \in [\![\mathrm{Agg}_{Y[\mathbf{b} \leftarrow \otimes X(\mathbf{a})]}(\psi)]\!](\mathcal{S})$ 且 $\mathcal{S} = e_1 \dots e_n$，那么根据公式的定义， $C = (i, j, \mu)$，其中 $\mathrm{dom}(\mu) = \{X_1, \dots, X_k, Y\}$，对于某些变量 X_1、.....、$X_k \in \mathbf{X}$，且存在同一流上的 $C' \in [\![\psi]\!](\mathcal{S})$，使得 $C' = (i, j, \mu')$， $\mu'(X_i) = \mu(X_i)$，并且 $\mu(Y) = \mu'(Y) \cup \{e\}$其中 e 是包含聚合结果的新事件。

我们还知道，如果 $C' \in [\![\psi]\!](\mathcal{S})$，那么 $C' \in [\![\mathcal{A}_\psi]\!](\mathcal{S})$。于是，我们可以找到自动机$\mathcal{A}_\psi$ 在 $\mathcal{S}$ 上的一次运行 ρ' ，使得 $C_{\rho'} = C'$。根据上述构造，我们可以将 ρ' 扩展为 $\mathcal{A}_\varphi$ 的一次运行 ρ ，其中初始转移取自 $\Delta_\varphi^{\mathrm{init}}$，中间转移取自 $\Delta_\varphi^{\mathrm{agg}}$，最后的转移取自 $\Delta_\varphi^{\mathrm{out}}$。可以验证， ρ 还会额外产生事件 e 和 $C = C_\rho$。

$[\varphi = \mathrm{Agg}^{Y[\mathbf{b}_1 \leftarrow \otimes_1 X_1(\mathbf{a}_1) \dots, \mathbf{b}_\ell \leftarrow \otimes_\ell X_\ell(\mathbf{a}_\ell)]}(\psi)]$。如果 $\varphi = \mathrm{Agg}_{Y[\mathbf{b}_1 \leftarrow \otimes_1 X_1(\mathbf{a}_1) \dots, \mathbf{b}_\ell \leftarrow \otimes_\ell X_\ell(\mathbf{a}_\ell)]}(\psi)$，那么我们可以类似于单属性聚合来构造 $\mathcal{A}_\varphi$，但要添加 l 个新寄存器。证明也类似。

□

D 实践示例

下面，我们介绍若干取自文献的查询，并展示如何用 ACEL 对它们进行建模。鉴于前文介绍的算子，请回顾一下：我们将 ACEL 定义为任何使用 CEL 标准算子（第2节）、聚合算子 Agg（第5节）或它们组合的公式 φ。本节首先介绍一些新的算子和谓词，它们作为 ACEL 的语法糖，可用于定义实际查询。随后，我们将介绍三个来自不同 CER 方案的含聚合查询，并使用 ACEL 对其进行描述。

D.1 用于描述现实查询的 ACEL 实用算子

为描述实际中的 CER 查询，以下算子会很有用。设 R 为任意事件类型。我们定义 ACEL 公式 NEXT(R)，其对应的 CEL 公式如下：

$$\texttt{NEXT}(R) \ \equiv \ \pi_R\big((X+ : R) \ \text{FILTER} \ X[\text{type} \neq R]\big) \ \text{OR} \ R$$

其中，我们假设 R 也可以用作变量名（即 $R \in \mathbf{X}$）。换言之，NEXT(R) 会在一个区间内查找第一个 R类型的事件，并丢弃其间的所有其他事件。例如，可以使用此算子简洁地定义 CEL 查询 S： NEXT(R)，以查找所有后面紧接着一个 R 事件的 S事件。我们定义此算子，以便在后续示例中使用，这些示例取自其他系统。

设$\mathbf{a}$ 为一个属性。我们还定义了一些辅助谓词，以便与 FILTER 算子配合使用，关联两个或多个事件。回顾一下，我们将谓词 P定义为事件的一个可能无限子集，并将 P 从事件推广到事件多重集 $E \subseteq \mathbf{E}$ ，使得 $E \models P$ 当且仅当对于每个 $e \in E$， $e \models P$。在实际指定查询时，我们需要一些无法直接通过将普通谓词推广得到的特殊多重集谓词。形式上，多重集谓词 $multiset\ predicate P$ 是事件多重集的一个子集，即 $P \subseteq \mathcal{P}_{bags}$(**E**)。例如，将（事件）谓词推广到事件多重集后，得到的就是一个多重集谓词。下面定义的这些多重集谓词将使我们能够分别（1）确保模式在属性 $\mathbf{a}$ 上具有相同的值，（2）确保某个属性的值递增，或（3）确保某个属性的值递减。具体而言，我们定义以下三个多重集谓词。对于一个事件多重集 E 和两个事件 $e_1, e_2 \in E$，令 Succ$_E$(e_1, e_2) 表示一个逻辑公式，用于检查 e_2 是否是 E中 e_1 的后继，形式化地说， e_1(time) $< e_2$ (time)，且不存在这样的 $e_3 \in E$，使得 e_1(time) $< e_3$(time) 且 e_3(time) $< e_2$(time)。

1. 我们将辅助谓词 [**a**]定义如下:

$$[\mathbf{a}] \;:=\; \{E \in \mathcal{P}_{bags}(\mathbf{E}) \mid \forall e_1, e_2 \in E.\; e_1(\mathbf{a}) = e_2(\mathbf{a})\}$$

2. 我们定义辅助谓词 [**increasing(a)**] 如下:

$$[\text{increasing}(\mathbf{a})] \;:=\; \{E \in \mathcal{P}_{bags}(\mathbf{E}) \mid \forall e_1, e_2 \in E.\; \mathsf{Succ}_E(e_1, e_2) \to e_1(\mathbf{a}) < e_2(\mathbf{a})\}$$

3. 最后，我们定义辅助谓词 [**decreasing(a)**] 如下:

$$[\text{decreasing}(\mathbf{a})] \;:=\; \{E \in \mathcal{P}_{bags}(\mathbf{E}) \mid \forall e_1, e_2 \in E.\; \mathsf{Succ}_E(e_1, e_2) \to e_1(\mathbf{a}) > e_2(\mathbf{a})\}$$

遵循 ACEL 的语法和语义，我们使用上述谓词以及 $\mathbf{X}$ 中的变量。例如，我们写作 φ FILTER $X[\mathbf{a}]$，以定义变量 X 中的所有事件都必须满足 $[\mathbf{a}]$。与标准谓词类似，我们在 FILTER 中使用合取和析取，分别作为组合过滤器或使用 OR 的语法糖。

下面我们给出几个取自以往文献的示例，并说明如何使用 ACEL 对其进行描述。

示例 D.1. 我们以改编自 SASE 论文第[14, 页2] , 的“查询 1”查询为例，其内容如下:

> “查询 1 会检索某些坏消息发生后 4 小时内谷歌股票的总交易量。PATTERN 子句声明模式的结构。它使用 SEQ 构造指定由两个部分组成的序列模式：第一部分指向一个类型为新闻的事件，第二部分指向一系列类型为股票的事件。后者使用 Kleene 加号，以“+”表示一个或多个特定类型的事件。每个部分都声明一个变量，用于指代相应的事件。使用 Kleene 加号的部分会使用“[ ]”符号将其变量声明为数组。WHERE 子句（如果存在）包含基于值的谓词，用于定义与模式相关的事件。在查询 1 中，第一个谓词要求新闻事件的类型属性为 bad。第二个谓词要求每个相关股票事件的股票代码均为 GOOG；“每个”语义由 b[i] 表示（其中 i ≤ 1）。我们将此类谓词称为个体迭代器谓词。WITHIN 子句指定一个覆盖整个模式的窗口，将所考虑的事件限制在 4 小时内。PATTERN、WHERE 和 WITHIN 子句共同完整定义一个模式。对事件流进行求值后，会产生一个模式匹配流。每个模式匹配都由用于匹配该模式的一组唯一事件序列组成，并存储在 a 和 b[ ] 变量中。RETURN 子句将每个模式匹配转换为结果事件。在其描述中，b[ ] 表示对数组中的事件进行迭代；从迭代器返回的每个事件中提取 volume 属性，并对所有提取的值应用聚合函数 sum()。”

此示例与原示例的区别在于，此处未考虑 within 运算符。SASE 查询语言中的查询如下:

Q13：模式SEQ(NEWS a, STOCK+ b[]) WHERE a.类型 = ‘bad’ 与 b[i].symbol = ‘GOOG’ RETURN 求和(b[].volume)

ACEL 中与前一个查询 Q1 等价的公式可以是:

$$\begin{aligned}\varphi_{13} \;=\; & \mathsf{Agg}_{Y[e \leftarrow \text{sum}(b(\text{volume}))]}\big(\\ & [\mathsf{NEWS}\ \text{AS}\ a\ :\ (\mathtt{NEXT}(\mathsf{STOCK}\ \text{AS}\ b))\oplus] \\ & \quad \text{FILTER}\ (a[\text{type} = \text{‘bad’}] \wedge b[\text{symbol} = \text{‘GOOG’}])\big)\end{aligned}$$

请注意，NEXT 算子是必要的，因为按照 SASE 语义，序列并非连续的，而我们需要准确地选取事件 NEWS 之后的那个事件。为此，SASE 使用了他们所称的 NEXT 选择策略。与其修改逻辑的语义，我们更倾向于引入一个能直接表达这一点的新算子。

示例 D.2. 我们以从 SASE 的论文 [37,p.3],中摘录的“查询 1”的一种改编版本为例，论文中写道

> “查询 1 计算 Hadoop 中映射器运行时间的统计数据。‘Pattern’ 子句指定了一个包含三个组件的序列模式：第一个是表示 Hadoop 作业开始的单个事件，接着是一个 Kleene+，用于汇集一系列表示作业中映射器的事件，最后是一个标记作业结束的事件。每个组件都声明一个变量，用来引用相应的事件，例如 a、b[ ] 和 c，其中数组变量 b[ ] 是为 Kleene+ 声明的。‘Where’ 子句使用这些变量指定基于值的谓词。此处的谓词要求所有事件都引用同一个作业 ID；这种对所有事件进行相等比较的写法可以简写为 ‘[作业 ID]’。‘Within’ 子句为该模式指定了一个 1 天窗口。最后，‘Return’ 子句构造每个输出事件，使其包含每个作业中映射器的平均时长和最长时长。‘”

此示例与原始示例的区别在于，它不考虑 within 运算符。

Q14：PATTERNseq(JobStart a, Mapper+ b[ ]，JobEnd c)
WHERE a.job_id = b[i].job_id and a.job_id=c.job_id RETURN b[ ].
period 的平均值，b[ ].period 的最大值
A 在 CEL 中，与前一个查询等价且包含聚合的公式如下：

$$\varphi_{14} = \mathsf{Agg}_{Y[e\leftarrow\max(b(\text{period})),f\leftarrow\text{avg}(b(\text{period}))]}\Big([(\text{JobStart AS } a;\ (\text{Mapper AS } b)+;\ \text{JobEnd AS } c) \text{ AS } X] \text{ FILTER } X[\text{job_id}]\Big)$$

示例 D.3。 我们以改编自 ESPER 文档[1],中的一个查询作为示例，其内容如下：

> “此示例语句通过以下方式展示了这一概念：针对每位客户，选取一组事件（先发生 ServiceOrder 事件，随后在 1 分钟内发生同一客户 ID 的 ProductOrder 事件）的总价；这些事件发生在最近 2 小时内，且价格总和大于 100；同时使用 where 子句按名称进行筛选。”

此示例与原始示例的区别在于，我们不考虑分组、窗口和滑动算子，且筛选条件略有不同。ESPER 查询语言中的查询如下：

Q15：选择 a.custId，求和(a.价格 + b.价格) 从 模式 [每个 a= ServiceOrder ->b=ProductOrder(custId = a.custId)]其中 a.名称 位于 (b.名称) 筛选 求和(a.价格 + b.价格) > 100

我们可以如下使用 ACEL 定义 Q2：

$$\varphi_{15} = \Big[\mathsf{Agg}_{Y(e\leftarrow\text{sum}(X(price)))}([(\text{ServiceOrder AS } a; \text{ProductOrder AS } b) + \text{AS } X]\text{FILTER } (X[\text{名称}]) \wedge X[\text{custId}])\Big] \text{FILTER } Y[\text{e} > 100]$$

示例 D.4. 我们以 GRETA 论文 [30,p.1],中提取的一个查询的改编版本为例，文中写道

> “查询 Q_1 计算每个行业在每个 10 秒滑动一次的 10 分钟时间窗口内的下降趋势数量。这些股票趋势由 Kleene 加号算子 S+表示。根据谓词 [company, sector]的要求，趋势中的所有事件都具有相同的公司和行业标识符。谓词 S.price > NEXT(S).price 表示趋势中价格从一个事件到下一个事件持续下降。该查询通过跳过价格上涨记录来忽略局部价格波动。”

此示例与原示例的区别在于，它不考虑滑动、窗口内和分组运算符。

Q16：返回 行业，计数(*) 模式 Stock S+WHERE ［公司, 行业］ 与 S.
价格 > NEXT(S).价格

前一个查询在 CEL 中对应的、带有聚合的公式可以是：

$$\varphi_{16} \quad = \quad \mathsf{Agg}_{Y[e \leftarrow \mathrm{count}(S(\mathrm{id}))]}\big([((\mathsf{Stock}\ \ \mathrm{AS}\ \ S\,;\,\mathsf{Stock}+)\ \ \mathrm{AS}\ \ X)+] \\ \mathrm{FILTER}\ \ X[\mathrm{sector}] \wedge X[\mathrm{company}] \wedge X[\mathrm{decreasing}(\mathrm{sector})]\big)$$

示例 D.5. 我们以改编自 GRETA 论文 [30,p.1],中提取的查询“Q_2”为例，论文中写道

> “查询 Q_2 计算集群上每个经历负载递增趋势的映射器在一个时间窗口内每个作业的 CPU 周期总数。该时间窗口长度为 1 分钟，每 30 秒滑动一次。由模式 Q_2 匹配的趋势是一个事件序列，由一个作业开始事件 S、任意数量的映射器性能测量 M+以及一个作业结束事件 E 组成。趋势中的所有事件都必须携带相同的作业和映射器标识符，由谓词 [job, mapper]表示。谓词 M.load < NEXT(M).load 要求负载测量在负载分布趋势中从一个事件到下一个事件递增。查询可以忽略任意事件，以检测所有感兴趣的负载趋势，从而实现准确的集群监控。”

此示例与原示例的区别在于，它不考虑滑动、within 和分组运算符。

Q17：RETURN mapper, SUM(M.cpu) PATTERNSEQ(Start S,
Measurement M+, End E) WHERE ［job,mapper］ 与
M.load < NEXT(M).load

下面给出一个在 CEL 中等价于前述查询、且包含聚合的公式：

$$\varphi_{17} \quad = \quad \mathsf{Agg}_{Y[e \leftarrow \mathrm{sum}(M(\mathrm{cpu}))]}\big([(\mathsf{Start}\ \mathrm{AS}\ \ S\,;\,\mathsf{Measurement}\ \mathrm{AS}\ \ M+\,; \\ \mathsf{End}\ \mathrm{AS}\ \ E)\ \mathrm{AS}\ \ X]\mathrm{FILTER}\ \ X[\text{作业}] \wedge X[\text{映射器}] \wedge M[\text{递增(load)}]\big)$$

示例 D.6. 我们以改编自 SHARON 论文[31,第1], 页中查询“q_1”的内容为例，其表述如下

> “图 1 中的查询 q_1 - q_7 计算路线上的行程计数，并以此衡量路线受欢迎程度。它们消费车辆位置报告流。每条报告都包含以秒为单位的时间戳、汽车标识符及其位置。这里，事件类型对应于车辆位置。例如，位于 Main Street 的车辆会发送一条类型为 MainSt 的位置报告。每次行程对应于同一辆车的一系列位置报告（由谓词 [vehicle] 所要求），这些报告发生在一个持续 10 分钟、每分钟滑动一次的时间窗口内（.....）。例如，模式 p_1 = (OakSt, MainSt) 出现在查询 q_1 - q_4 中。在多个相似查询之间共享公共模式的聚合，对于提升系统响应能力至关重要。”

此示例与原示例的区别在于，它不考虑 within 和分组运算符。

Q18：RETURN COUNT (*) PATTERNOakSt,
MainSt, StateSt WHERE ［车辆］

A 与之前的 CEL 聚合查询等价的公式可以如下所示：

$$\varphi_{18} \quad = \quad \mathsf{Agg}_{Z[f \leftarrow \mathrm{count}(Y(\mathrm{vehicle}))]}\big(\\ [((\mathsf{OakSt}\,;\,\mathsf{MainSt}\,;\,\mathsf{StateSt}\ \ \mathrm{AS}\ \ Y)\ \ \mathrm{AS}\ \ X)+] \\ \mathrm{FILTER}\ \ X[\mathrm{vehicle}]\big)$$

示例 D.7. 我们以改编自 HAMLET 论文 [28,p.1], 中提取的查询“q_3”作为示例，该论文指出

> “一次行程中的所有事件都必须具有相同的司机和乘客标识符，这是谓词 [driver, rider] (...) 的要求。查询 q_3 跟踪那些在司机被困于缓慢行驶的交通流中时取消已接受请求的乘客。所有三个查询都包含代价高昂的 Kleene 子模式 T+ ，它可以匹配任意长的事件趋势。”‘

这个示例与原始示例的区别在于，它不考虑 within、slide 和分组运算符。

Q19：返回 T.district、计数(*)、求和(T.duration) 模式
序列(Request R、行程 T+、取消 C) 其中 ［司机、乘客］

以下是在 CEL 中带有聚合功能、与前一个查询等价的公式：

$$\varphi_{19} = \text{Agg}_{Y[e\leftarrow\text{求和}(T(\text{时长}))\ f\leftarrow\text{计数}(C(\text{司机}))][((\text{请求 AS } R; \text{行程 AS } T+; \text{取消 AS } C) \text{ AS } X)+]\text{FILTER } X[\text{司机}] \wedge X[\text{乘客}]$$

示例 D.8. 我们以从 COGRA 论文 [29, p.2], 中提取的查询改编版为例，其中

> “查询 q_2 计算在每 30 秒滑动一次的 10 分钟时间窗口内，Uber 司机在部分乘客联系司机后取消行程时能够完成的 Uber 拼车行程数量。每趟行程以一个接受事件开始，接着是任意数量的呼叫和取消事件，最后以一个完成事件结束。每个事件都带有与司机关联的会话标识符。根据谓词 [driver] 的要求，构成同一趟行程的所有事件必须带有相同的会话标识符。跳至下一个匹配语义允许查询 q_2 跳过途中、下车点等无关事件。”‘

此示例与原始示例的区别在于，此示例不考虑 within、slide 和分组运算符。

Q20：返回司机，计数(*) 模式 SEQ(Accept, (SEQ(Call, 取消))+,
Finish) 语义 skip-till-next-match WHERE ［driver］

A 在支持聚合的 CEL 中，与前一个查询等价的公式可以是：

$$\varphi_{20} = \text{Agg}_{Z[f\leftarrow\text{count}(F(\text{driver}))]}\big([((\text{Accept}; (\text{Call}; \text{Cancel})+; \text{Finish AS } F) \text{ AS } X)+] \text{ FILTER } X[\text{driver}]\big)$$

示例 D.9. 我们以改编自 COGRA 论文[29,p.1], 所述的查询“q_1”为例：

> “查询 q_1 用于检测被动身体活动（例如阅读、看电视）期间的最小和最大心跳。查询 q_1 处理重症监护患者的心率测量值流。每个事件都包含以秒为单位的时间戳、患者标识符、活动标识符和心率。对于每位患者， q_1 会在每 30 秒滑动一次的 10 分钟时间窗口内，连续检测不断升高的心率测量值。对于每位患者，匹配的事件之间不得跳过任何测量值，这正是连续语义所表达的。”

此示例与原始示例的区别在于，此版本不考虑 within、group-by 和滑动运算符。查询如下：

Q21：RETURN患者, 最小值(M.rate), 最大值(
M.rate) 模式 Measurement M+语义 连续 WHERE［
患者］与 M.rate ＜ NEXT(M).rate 与 M.activity ＝
被动

与前一个查询等价的 ACEL 公式可以如下所示：

$$\varphi_{21} \quad = \text{聚合}_{Y[e\leftarrow\text{最小值}(M(\text{速率}))\ f\leftarrow\text{最大值}(M(\text{速率}))]}\big[(\text{测量 AS} \quad M)\oplus\text{FILTER} \quad M[\text{患者}] \wedge M[\text{递增(速率)}] \wedge M.\text{活动} \ =\text{‘被动’}\big]$$

可以检查公式 φ_3 是否指定了与 Q3 相同的查询；不同之处在于， φ_3 具有形式化的指称语义。

值得注意的是，我们也考虑了其他使用聚合的方案示例（例如 CAYUGA [13]）；然而，这些方案的查询语言是过程式的，并不适用于本文所采用的声明式聚合概念。