远端的 Lambda：飞行 Lambda 与轮上的 Lambda 的故事

Giorgio Audrito Daniele Bortoluzzi FerruccioDamiani Giordano Scarso Gianluca Torta都灵大学计算机科学系（DI），意大利都灵 name.surname@unito.it

Andrea Basso
MITO Technology，意大利米兰
speedrun2006@gmail.com

MonicaCochi Lorenzo Gusman
都灵机场
卡塞莱托里内塞（TO），意大利
name.surname@sagat.trn.it

LorenzoComba Paolo Gay
农业、林业与食品科学系（DiSAFA）
都灵大学，意大利都灵
name.surname@unito.it

Paola Dal Zovo
Concept Engineering
Reply，意大利都灵
p.dalzovo@reply.it

Giada Galati Francesco Gallo
Eurix，都灵，意大利
surname@eurixgroup.com

Aljaž Grdadolnik
计算机与信息科学学院
斯洛文尼亚卢布尔雅那，卢布尔雅那大学
aljaz.grdadolnik1@gmail.com

Massimo Pescarollo PaolaPisano“Cognetti de Martiis”经济与统计系，都灵大学，都灵，意大利
name.surname@unito.it

聚合编程（AP）是一种编程范式，它依靠基于邻近性的异步交互，对分布式设备集合的集体行为进行编程；这些设备可能位于网络远端边缘。交换演算（XC）是近期提出的一种 AP 基础模型，本质上是一种带类型的λ演算，并扩展了一个算子（交换算子），以便在邻居设备之间提供隐式通信机制。本文将简要介绍 XC 及其作为 C++ 库的实现，该库名为 FCPP。FCPP 库及其工具链主要由都灵大学计算机科学系开发。Stefano Berardi 曾在该系度过大部分学术生涯，致力于计算机科学逻辑基础方面的杰出研究，并将自己对研究的热情传递给学生和青年研究人员；他常常借助带类型的λ演算开展工作。FCCP 程序本质上是一个带类型的λ项，而 FCPP 已被用于编写代码，并将其部署在网络远边缘的设备上，包括漫游车和（即将投入使用的）无人驾驶航空器（UAV）；这也正是本文标题的由来。

谨以此文献给 Stefano Berardi，祝贺他迎来第 2^6 个生日。

1 引言

嵌入式系统、智能尘埃、泛在计算、信息物理系统和物联网都是一些流行词（大致按时间顺序排列），研究人员和从业者用它们来

T. Altenkirch、F. Barbanera、F. Damiani 和 U. de’Liguoro（编）：《逻辑与类型理论：献给 Stefano Berardi 的论文集，值其 64 岁生日》，EPTCS 441，2026 年，第 19–45 页，doi:10.4204/EPTCS.441.2

© G. Audrito 等。本文采用知识共享署名许可协议授权。

指代部署在物理环境中的感知、计算和执行实体（称为节点、设备等）。这些技术的一个明显趋势是，部署的实体数量不断增加，因此，若逐一为每个实体编程，就无法部署全局层面的软件功能。为此，人们正在研究一系列方法——可归类为“宏编程”和“空间自组织”（综述见 [18, 31, 38]）——旨在提供一种方式，将节点网络视为一个可整体编程的空间分布式平台。

在这些方法中，聚合编程（*AP*） [19, 49] 范式基于这样一种系统交互理解：信息通过设备集合传播，这些设备可能（但并非必然 [12]）位于网络的极端边缘，依赖基于邻近性的隐式交互，而不是单个设备与其对等节点及环境之间的显式交互。AP 的参考编程模型是场演算（*FC*） [3, 17, 29] 以及其最近提出的推广——交换演算（*XC*） [5–7]。它们是核心函数式语言，并通过基于邻近性的交互原语加以扩展，使程序员能够通过自底向上构建可复用的、具有内在弹性的聚合行为模块，来组织复杂的计算任务。特别是，XC 可以理解为一种带类型的λ演算，其通过一个算子（交换算子）提供了邻居设备之间的隐式通信机制。

AP 的主要实践支持是 C++ 库和工具链 *FCPP* [2, 14–16]（主要由都灵大学计算机科学系开发），以及 Scala 库和工具链 *ScaFi* [10, 25–27]（主要由博洛尼亚大学计算机科学与工程系开发）。FCPP 允许程序员将 XC 程序嵌入 C++ 应用程序，并将其部署到物理设备网络上，也可部署在模拟器中。它已用于多个目前正与 Reply（https://www.reply. com）、Synesthesia（https://synesthesia.it）、Eurix（https://www.eurixgroup.com/）和都灵机场（https://www.aeroportoditorino.it/en）合作开展的工业案例研究中。[1] –这些活动的一项初步成果最近已发表 [47]。

论文贡献本文旨在为研究 λ 演算和类型论的研究者（如 Stefano Berardi）平易地介绍 XC。我们还简要介绍 FCPP 库及其工具链，并概述一些工业案例研究：其中，FCCP 程序可理解为类型化 λ 项，已部署在漫游车上，也将部署到无人机上——这正是本文标题的由来。Stefano Berardi 的大部分学术生涯都在都灵大学计算机科学系度过；他在那里围绕计算机科学的逻辑基础开展杰出研究，并将对研究的热情传递给学生和青年研究人员（按时间倒序排列，包括本文第一作者和第三作者），而且经常借助带类型的 λ 演算。因此，我们认为，本文所报告的研究活动完全属于 Stefano Berardi 的研究兴趣范围。

[1] 例如，PNRR NODES（https://ecs-nodes.eu/en）工作组1“可持续交通与航空航天”的 RoboNG（https://www.ecs-nodes.eu/en/1-aerospace-and-sustainable-mobility/ progetti-imprese/robong）和 RoboAPP（https://www.ecs-nodes.eu/en/ 1-aerospace-and-sustainable-mobility/progetti-imprese/robonghttps://ecs-nodes.eu/en/1-aerospace-and-sustainable-mobility/progetti-accademici/roboapp）级联资助项目，以及 CN AgriTech（https://agritechcenter.it/）。

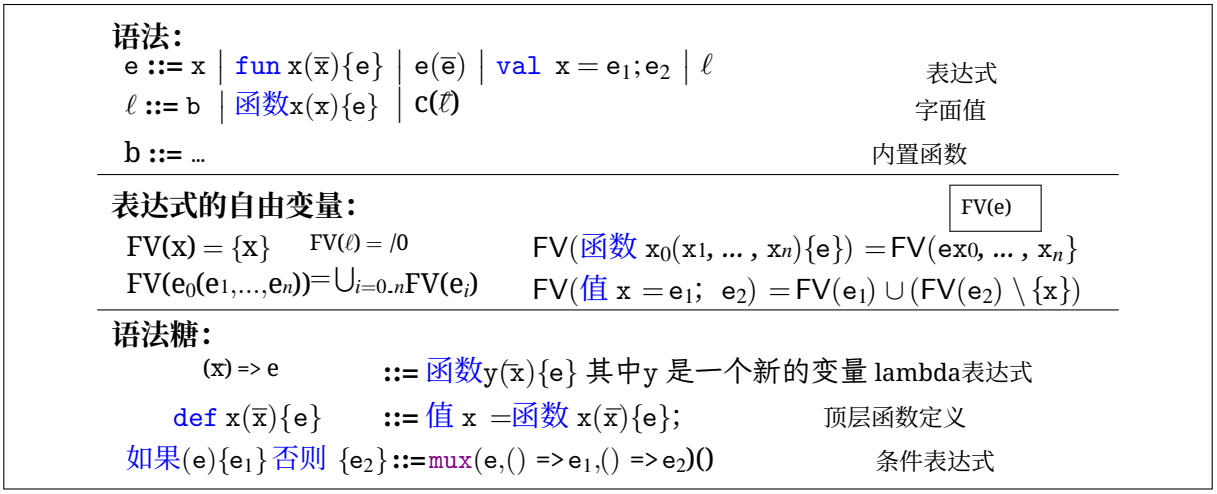

图 1：XC：语法（上）、自由变量（中）和语法糖（下）。

论文结构第 2 节形式化介绍 XC（语法、类型判断和操作语义），第 3 节简要概述 XC 的设计决策和主要性质。第 4 节简要介绍 FCPP 库及其工具链。第 5 节简要概述一些工业案例研究。第 6 节讨论相关工作。第 7 节以总结性评述和未来工作方向作结。

2 从 λ 演算到交换演算

2.1 交换演算中的 λ 演算

本节介绍 XC 的语法和类型判断。XC 可以理解为一种采用 ML 风格类型系统的按值调用 λ 演算。

2.1.1 语法

XC 的语法见图 1（上）。按照 [33], 上横线记法，横线上方表示一个（可能为空的）元素序列，例如，x表示 $x1$，……，$x_n (n \geq 0)$。XC 表达式 e 可以是以下任意一种：

- 一个变量 x;
- 一个（可能递归的）函数 funx(x){ē}，它可以包含自由变量;
- 一个函数调用 e(e);
- 一个 类 *let* 的表达式 val x = e;e;
- 一个 字面值 ℓ，即要么是内置函数 b、已定义函数 funx(x){e} 且不含 自由变量，要么是数据构造器c 应用于字面值（可能没有）。

内置函数——见图 2——包括多路复用函数 mux，一种急切条件运算符，它会求值其全部三个参数，并根据第一个参数的值返回第二个或第三个参数；以及用于数字、布尔值、对、列表等标准数据类型的内置函数和数据构造器。

自由变量的定义遵循标准方式（图 1，中间），如果 FV(e) = /0，则表达式 e 是闭合的。程序都是闭合表达式。

NAME	类型方案	描述
函数：		
mux	$(\texttt{bool}, A, A) \to A$	多路复用运算符
+, -, *, /	$(\texttt{num}, \texttt{num}) \to \texttt{num}$	算术运算符
and, or	$(\texttt{bool}, \texttt{bool}) \to \texttt{bool}$	布尔运算符
==, <=, >=	$(A, A) \to \texttt{bool}$	关系运算符
pair	$(A, B) \to \texttt{PAIR}[A, B]$	对的创建
fst	$(\texttt{PAIR}[A, B]) \to A$	对的第一个元素
snd	$(\texttt{PAIR}[\text{甲}、\text{乙}]) \to \text{乙}$	对的第二个元素
数据构造器：		
−1, 0, 0.25, 1, Infinity	num	数值构造器
True, False	bool	布尔构造子
Pair	$(A, B) \to \texttt{PAIR}[A, B]$	对构造子

图 2：XC：示例中使用的内置函数和数据构造器的名称、类型方案和描述——关系运算符中的泛型A 类型不允许包含函数类型。

除了某些标准语法糖（二元运算符采用中缀记法、零元数据构造器省略括号）外，图 1（底部）还介绍了一些非标准语法糖。具体来说：lambda表达式编码为一个使用新名称（即在整个程序的其他任何地方都未出现过的名称）的 fun表达式；顶层函数定义编码为一个绑定函数名称的类let表达式；条件表达式则通过对分支中的代码进行 lambda 抽象，使用mux 运算符选择其中一个分支，然后应用所选分支来编码。

2.1.2 类型判断

图 3 展示了 XC 的经典 Hindley-Milner类型系统 [28] 。类型T 可以是：

- 一个类型变量 α;
- 一个（可能递归的）数据类型 $K[T], \urcorner$ 由一个参数化类型名 K （例如 num、bool、PAIR、LIST 等）构成，其元数为$n \geq 0$，应用于n 个类型$T1$，.....T_n（可能为零）——作为语法糖，我们将K 写作$K[]$的简写（例如，num 作为 num[] 的简写）；
- 或一个函数类型（$T\urcorner \to T$。

我们用 TV(T) 表示 T 中类型变量的集合。多态函数和数据构造器由类型方案 TS 支持，其形式为 $\forall\alpha.T$，其中 α 在 T 中自由出现，表示依照类型模式实例化关系 $\prec$，通过用类型 T 替换 α 而得到的所有类型。一个定型环境$\mathscr{A}$是一组假设 a :TS ，其中假设主体 a 可以是变量、内置函数或数据构造器。我们假定传感器 s 具有类型 () $\to T$ ，其中 TV(T) = /0。在程序的类型判断中，一个初始定型环境 $\mathscr{A}_0$ 为每个可用的数据构造器和内置函数声明一个（唯一的）类型方案。随后，该 $\mathscr{A}_0$ 会用在对程序的子表达式进行类型判断时遇到的有界变量的（唯一）假设加以扩展。我们用 $\mathscr{A}$(a) 表示 $\mathscr{A}$ 中唯一的类型方案 a。

我们通过判断式 $\mathscr{A} \vdash \mathsf{e} : T$ 来规定表达式的类型判断，其读作“在假设 $\mathscr{A}$ 下，表达式 e 具有类型 T”。按照 [33], , 多个上划线会一并展开（例如， $\mathscr{A} \vdash \mathsf{e} : T$ 表示 $\mathscr{A} \vdash \mathsf{e}_1 : T1, \cdots, \mathscr{A} \vdash \mathsf{e}_n : T_n$）。表达式的类型判断规则是语法导向的。所有规则均为标准规则。

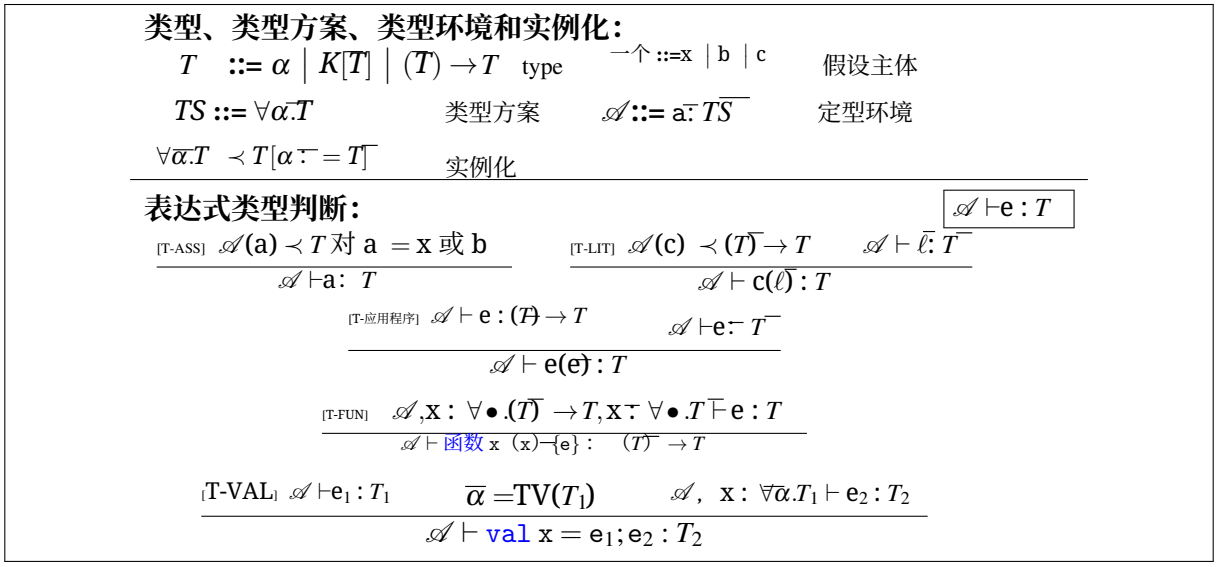

图 3：XC 表达式的类型判断。

NAME	类型方案	描述
示例中使用的传感器：		
uid	$() \to \texttt{num}$	唯一设备标识符
gps	$() \to \texttt{PAIR}[\texttt{num},\texttt{num}]$	*GPS*纬度和经度
time	$() \to \texttt{num}$	纪元 *Unix* 时间戳（自 *1970* 年 *1* 月 *1* 日起的秒数（协调世界时））
temperature	$() \to \texttt{num}$	设备传感器测得的温度（摄氏度）

图 4：XC：一些传感器的内置函数的名称、类型方案和描述。

2.2 设备、空间、时间与物理环境

到目前为止，给定 XC 程序的每一次执行总是表现出相同的行为：要么产生一个值（始终相同），要么始终不终止。现在，让我们把设备、空间、时间和物理环境引入其中，让事情变得更有趣。具体而言，我们考虑一组在空间中分布的设备，它们通常配备传感器 和执行器。这些设备反复计算同一个 XC 程序，该程序可能包含用于访问传感器值的内置函数，如图 4 所列。

设备的行为通过（执行）轮次或触发来建模：设备独立地“触发”——即收集传感器数据，然后“原子地执行”一个 XC 程序。执行一个终止的 XC 程序会产生输出（程序的值），可用于描述执行动作；随后设备会等待再次触发——有时我们会说设备“醒来”、触发，然后“重新入睡”。

集合中的所有设备都执行同一个程序。不过，每台设备每次触发时，会根据传感器的值，以不同的方式执行该程序。所有设备的此类触发可能按照大致相同的周期执行，但通常并不作此假设（每台设备都可以有自己的触发计划）。事实上，设备可能会耗尽电池电量，再也无法唤醒；也可能在电池充电后经过很长时间才重启；或者进入程序的非终止执行状态并“消失”（期间可能会发生重启，中止当前执行并允许再次触发）。因此，轮次完全是异步的。

例如，设想一组设备，每台设备都配有屏幕，用于显示最近一次已完成触发所产生的值。如果这些设备运行程序 `pair(pair(time(),gps()),温度())`，

NAME	类型方案	描述
通信：		
exchange	$(A,(\underline{A},\underline{A}) \to \text{PAIR}[T,A]) _ \to T$	交换消息
相邻值操作：		
nfold	$((A,B) \to A,\underline{B},\underline{A}) \to A$	相邻值折叠
self	$(\underline{A}) \to A$	提取自身消息
updateSelf	$(\underline{A},A) \to A _$	更新 *self* 消息
updateDef	$\underset{A}{(\underline{A},\ A)\ \to _}$	更新默认消息
示例中使用的传感器：		
senseDist	$\underline{\text{num}}$	到邻居的距离估计值

图5：XC：内置通信函数的名称、类型方案和描述。

那么，每台至少触发过一次的设备都会显示一个形如 Pair(Pair(ℓ_1, Pair(ℓ_2, ℓ_3)), ℓ_4) 的值，其类型为 PAIR[PAIR[num, PAIR[num, num]],num],其中： ℓ_1是最近一次已完成触发的开始时间； ℓ_2 和 ℓ_3 分别是该次触发开始时设备的 GPS 纬度和经度； ℓ_4 是设备在该次触发开始时感测到的温度。

2.3 基于邻近的交互

到目前为止，如果我们将每次触发视为一个事件，认为它在特定时间发生于某个空间位置设备上，那么终止的 XC 程序的执行会产生一组可数的独立事件，每个事件都由程序生成的值标记（如果某次触发未能终止，就不会产生事件）。现在，让我们加入基于邻近的交互，让内容更加丰富。具体来说，我们考虑设备可以通过交换邻居设备的消息进行异步通信，并考虑可以使用交换内置函数进行通信，以及使用其他内置函数访问从邻居处接收的信息的 XC 程序，例如图5中列出的那些内置函数。

2.3.1 消息与邻居

一次触发期间，设备可以安排发送消息。这些消息会在触发结束时发送给邻居。每台设备都会将接收的消息收集到缓冲区中，并且每个邻居只保留最新的一条消息。此外，超过某个可配置生存期的消息会过期（即从缓冲区中移除）。我们的形式化假设来自同一发送方的消息顺序保持不变，但除此之外，对消息几乎不作任何假设。具体而言：*(i)* 设备可能距离过远，无法向（曾经的）邻居发送消息；*(ii)* 消息可能丢失；*(iii)* 设备可能消失或发生故障；*(iv)* 设备可能重启，从而丢失其接收消息队列；*(v)* 设备可能判定来自某些设备的消息已过期。值得注意的是，XC 对这些情况进行了抽象：设备触发时，会找到来自其他某些设备（最近可用的触发）的若干消息；设备触发不会从缓冲区中移除消息，因此消息可以在多次触发间留存。某次触发中有消息可用的设备，就是该次触发的邻居。一次触发期间，设备可以安排向自己发送一条消息——如果该消息未过期，且设备在下次触发前未重启，那么它会在设备下次触发时处理。

图6展示了一个涉及四台设备的系统执行示例，设备标记为 δ_1 至 δ_4。在观察期间， δ_2 激活了两次，执行了两次计算事件，分别记为 ε_1 和 ε_2。灰色箭头所示的消息表示传输失败，永远不会送达。在第二次计算 ε_2中，设备 δ_2 处理了休眠阶段收到的三条消息，这些消息分别来自 δ_4、δ_3和 δ_1。此次计算之后， δ_2 向 δ_3 和 δ_1都发送了消息。

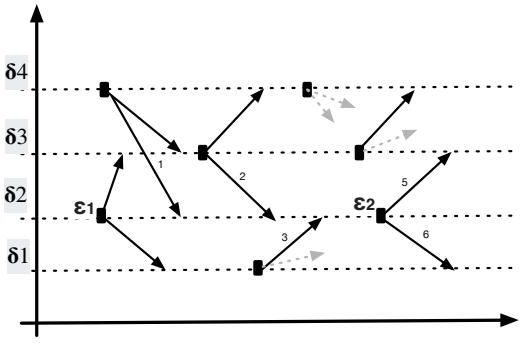

图6：XC：系统执行的表示（取自 [7]）。

2.3.2 邻近值

邻近值（*nvalue*）是一个映射 w，它将设备标识符 δ_i 与字面值 ℓ_i 关联起来，并且还包含一个默认 字面值 ℓ。它写作 $\ell[\delta_1 \mapsto \ell_1, \ldots, \delta_n \mapsto \ell_n]$（$n \geq 0$），读作"n值 w 在所有地方（即对所有邻居）都为 ℓ，但设备 $\delta_1, \ldots, \delta_n$ 对应的值为 $\ell_1, \ldots, \ell_n$"。n值不能出现在程序的源代码中：它们是在运行时产生的。每个 n值都用于描述从邻居接收和发送给邻居的（集合）字面值。设备 δ 所操作的 n值中带有对应条目的设备，通常只是所有设备的一个子集，例如，因为某些设备距离 δ 太远，无法提供值，或者上次提供的值已过期。当某台设备的字面值不可用时（例如，操作该 n值的设备刚出现或重启，因此此前没有触发事件安排向自身发送消息），就会使用默认值。

为说明 n值 的用法，同时参照图6：当 δ_2 为进行计算 ε_2 而唤醒时，它可以构造一个 nvalue $w = 0[\delta_4 \mapsto 1, \delta_3 \mapsto 2, \delta_1 \mapsto 3]$，该 nvalue 聚合其在休眠阶段从 δ_4、δ_3 和 δ_1 接收到的标量值，分别为 1、2 和 3。任何未明确列出的设备均贡献默认值 0。在此次计算之后，δ_2 可能会发出由 nvalue $w' = 0[\delta_3 \mapsto 5, \delta_1 \mapsto 6]$ 捕获的出站消息，这意味着 δ_3 接收到 5，δ_1 接收到 6，而所有其他接收方 —— 例如刚刚加入的设备 —— 则接收默认值 0。

记号 $w(\delta')$ 表示：若 δ' 存在于 w 中，则为显式赋给它的值 ℓ'，否则为默认值 ℓ，从而突出将 n值解释为带有回退的映射。例如，我们有 $w'(\delta_1) = 6$ 以及 $w'(\delta_2) = 0$。为清晰起见，在代码片段中，我们对承载邻居特定值的变量加下划线。同样，我们对基本类型 A 加下划线，以标明诸如 $w = \ell[\delta_1 \mapsto \ell_1, \ldots, \delta_n \mapsto \ell_n]$ 之类的邻值表达式的类型，其中每个 ℓ 与 ℓ_i 的类型均为 A。

n值推广了字面值 任何字面值 ℓ 都可以隐式提升为形式为 $\ell[]$ 的 nvalue，其中所有设备都被赋予相同的默认值。这使得字面值与 n值的形式化处理得以统一。此外，定义在字面值上的运算会被自动提升，以逐点方式作用于 n值，即该函数被分别应用于每个映射值。例如，给定 $w_1 = 0[\delta_1 \mapsto 1, \delta_2 \mapsto 2]$ 与 $w_2 = 2[\delta_2 \mapsto 1]$，我们有 $w_3 = w_1 + w_2 = 2[\delta_1 \mapsto 3, \delta_2 \mapsto 3]$。请注意，$w_3$ 中的 $\delta_1 \mapsto 3$ 是由于 w_1 中的 $\delta_1 \mapsto 1$，以及 δ_1 在 w_2 中的默认值为 2。此外，得益于字面值到 n值的自动提升，诸如 $w_1 + 1$ 之类的表达式也是有效的，并被解释为 $w_1 + 1 = 0[\delta_1 \mapsto 1, \delta_2 \mapsto 2] + 1 = 1[\delta_1 \mapsto 2, \delta_2 \mapsto 3]$，其中标量值 1 被提升为 nvalue $1[]$，然后逐点相加。

对 n值的运算 除了支持逐点运算之外，n值还可以通过折叠进行归约，这与列表折叠类似。此操作使用内置函数 `nfold`$(f\colon (A,B) \to A, \mathtt{w}\colon B\ \ell\colon A)\colon A$，

其中，函数 f 会迭代应用于场 w 中与相邻设备关联的值（不包括执行计算的设备对应的值），并从初始值 ℓ 开始。

例如，设备 δ_2 对其邻居执行折叠 $\{\delta_1, \delta_3\}$。如果像之前一样定义 $\mathtt{w}_1 = 0[\delta_1 \mapsto 1, \delta_2 \mapsto 2]$，那么表达式 $\mathtt{nfold}(+, \mathtt{w}_1, 10)$ 的计算结果为 $10 + \mathtt{w}_1(\delta_1) + \mathtt{w}_1(\delta_3) = 10 + 1 + 0$。由于 n 值的语义不依赖于设备标识符的特定顺序，因此通常假设折叠函数 f 同时具有结合性和交换性。

示例 2.1（距离估计）。设备可以依靠邻居传达的距离估计值 n，估算自身到网络中目标节点的距离。具体而言，计算各邻居的值 n 与设备对该邻居的本地距离估计值之和的最小值；该本地距离估计值由 senseDist 表示（通常由设备上的传感器提供）。这种组合使用 nfold 完成：从 无穷大 开始，对所有调整后的邻居估计值应用 min 函数：

```
1 定义 distanceEstimate(n) { // 类型: (num) → num
2 nfold(min, n -+ senseDist, 无穷大)}
3
```

这里，场 n 和 senseDist 会在邻居之间逐点求和。也就是说，对于每个邻居 δ，设备都会计算 n(δ) + senseDist(δ) 的和，作为单独的估值。最终结果是这些求和值中的最小值；如果没有可用值，则为 无穷大。

我们还可以使用更多内置的 n值 运算。其中包括 self(w : A) : A，它会从场 w 中提取与当前设备 δ 关联的值，也就是说，它返回 w(δ)。此外，updateSelf(w : A ℓ : A) : A 会通过更新键对应于 self 设备的 w 来生成一个新的 nvalue，并将先前的值（如果有）替换为 ℓ，同时保持所有其他条目不变。此运算对应于带默认值的映射中的替换，表示为语法 w[$\delta \mapsto \ell$]。更一般地说，像 $\ell[\delta_1 \mapsto \ell_1, \ldots]$ 这样的表达式表示映射：它将 ℓ_i 赋给 δ_i，并将 ℓ 用作所有其他设备的回退值，从而有效地描述了对统一默认映射的更新。

2.3.3 交换内置函数

XC 提供一个通信原语 exchange(e_i, (o, n) => return e_r send e_s)，它会去糖化为 exchange(e_i, (o, n) => pair(e_r, e_s))，后者通过本地起始值 e_i 和更新函数来建模集体计算。给定旧值 o（即当前设备在上一轮计算出的出站消息的 nvalue）和邻居值 n（即相邻设备在最近轮次中为当前设备计算出的传入消息的 nvalue），此函数会计算一个要返回的 nvalue e_r 和一个要发送给邻居的 nvalue e_s。此类表达式的求值过程如下：(i) 设备将表达式 e_i 求值为字面值 ℓ_i（初始值）；(ii) 将变量 o 替换为本地设备在上一轮为 e_s 计算出的 nvalue w_o（如果这是当前设备首次计算此 exchange 结构，则改用 ℓ_i）；(iii) 将变量 n 替换为此 exchange 收到的来自邻居的消息的 nvalue w_n，默认使用 ℓ_i；(iv) 整个 exchange 表达式返回对 e_r 求值所获得的（邻居或字面）值 v_r；(v) 对 e_s 求值，得到由要发送给邻居设备 δ' 的字面值组成的 nvalue w_s；这些设备会使用各自对应的 $w_s(\delta')$ 来构造其下一执行轮次的 w_n 值。

由于在大多数情况下表达式 e_r 与 e_s 相同，我们提供交换(e$_i$, (o, n) => retsend e)作为交换(e$_i$, (o, n) => (val x = e; pair(x, x))的简写。另一种有用的通信模式是获取某一表达式的邻居的值，这可通过已定义的函数 nbr(e_i, e_s) = 交换(e_i, (o, n) => return n send e_s)来完成。在 nbr(e_i, e_s) 中，表达式的值

e_s 会被发送给邻居，而从邻居接收到的值（连同来自 e_i 的默认值一起汇集于 n 中）将作为 nvalue 返回，从而提供关于 e_s 的邻居的值的视图。另一种常用模式是跨轮次的状态保持，可通过已定义的函数 old(e_i, e_s) = 交换(e_i, (o, n) => return o send e_s)获得。在此情况下，表达式 e_s 的值会与邻居共享，但仅与当前设备相关，当前设备将在其后续轮次中将其用作该交换表达式的整体结果。

XC 表达能力的一个关键方面在于，交换（以及由此实现的 *nbr* 和 *old*）能够为不同的邻居分配不同的值。这种能力支持设备之间进行定制化交互，对于许多协调模式至关重要，如以下示例所示。

例 2.2（乒乓计数器）。下面的函数计算一个邻居场“连接计数器”，将每个邻居映射到与其建立双向连接的次数。

```
1 定义 乒乓() { // 类型: () → num交换( 0,(o,n)
2  =>返回发送  n + 1)}
3
```

每次对 ping-pong 求值时，设备都会获取一个场 w，将每个邻居映射到其当前的连接计数——对于新连接的对等节点，默认值为 0。然后，设备使用 n + 1（其中 n 表示 w）递增每个邻居的计数器，包括刚刚连接的邻居（因此它们会被映射到 1）。更新后的场 w + 1既会与邻居共享，也会作为函数结果返回。

只要连接持续存在，每个设备就会不断接收邻居的计数器值，将其递增后再发送回去——这实际上是在统计乒乓式的消息交换次数。如果连接中断（且相关消息过期），计数器就会重置为 0。之后，如果连接重新建立，计数器可能会再次开始递增。关键的是，该函数会为每个邻居分别维护一个计数器，从而能够发送针对特定邻居的值，实现个性化的网络跟踪。

前面的示例提供了跟踪双向连接持续时间的方法。接下来的示例中，我们将介绍另一种连接计数器，专门用于衡量单向连接的持续时间。

例 2.3（单向连接计数器）。下面定义的 *uniconn-count* 函数会生成一个 nvalue，用于表示入站连接计数器，并为每个邻居记录成功接收到其消息的次数。

```
1 def uniconn-count() {交换( 0, (o,n)  => retsend  o +
2 updateDef(1,0) )}
3
```

请注意，*uniconn-count* 只使用参数 o，忽略 n。交换以初始值 0 开始。在每一轮次中，更新步骤都会递增场 o 中所有当前邻居对应的值，同时保持默认值不变。这一行为通过 *updateDef*函数实现：将其应用于输入 1 和 0 时，会产生默认值为 0、并为所有现在的邻居赋值 1 的邻近值。因此，该机制会统计某个设备连续保持为邻居的轮次数。每当先前连接的邻居从当前视图中消失时，计数器就会重置为 0。

由于相邻设备是指最近轮次中成功向当前设备传输过消息的设备，因此该函数可作为评估设备与其对等节点之间入站连接质量的指标。重要的是，与乒乓式计数器不同，它无法反映出站通信的质量：即使当前设备无法发送消息，只要它持续接收消息，计数器就会继续递增。

最后举一个例子，我们现在介绍自组织距离算法，展示交换如何与*nfold* 结合，实现分布式协调。

示例 2.4 （自组织距离）。计算每个设备到指定源设备集合的最短距离，可得到所谓的梯度 [9]。梯度是基础性的自组织模式，可用于近似远距离，并引导数据沿高效路径传播。`distanceTo`函数通过分布式 Bellman-Ford 算法的变体实现这一机制 [30]。

```
1 def distanceTo(src) { // 类型: (bool) → num交换( 无穷大, (n)  => 返回
2 mux(src, 0, distanceEstimate(n)))}                                    –
3
```

在网络中反复执行时，该函数会收敛到从源节点出发的稳定距离值，也就是说，这些节点上的 `src` 求值为 true。核心交换表达式通过以下方式细化本地估计值：*(i)* 无输入时默认取无穷大，*(ii)* 源设备的距离设为零，以及*(iii)* 对于所有其他节点，计算邻居距离与感知到的设备间距离之和中的最小值（参见示例 2.1）。如果所得估计值为d，则返回该值，并使用将每个对等节点映射到d的统一字段与邻居共享。

运算符`mux` 用于确保条件分支的两个分支都会被求值：即使源设备返回 0，它们仍必须通过`distanceEstimate`计算并共享其邻居场 n。该设计支持动态适应：网络拓扑的任何变化——无论由故障、移动还是设备加入/离开引起——都会直接改变 n 的定义域，从而在局部和全局层面触发重新计算。 –

2.4 通过对齐实现组合性

当一个程序包含多个交换 表达式时，XC 通过一种称为对齐的机制来保证正确的消息传递。这确保某一轮次中交换的值会在下一轮次由对应的交换表达式接收——具体而言，即位于相同 AST 位置和栈帧中的那个表达式。即使存在分支、函数调用或递归，这种对齐依然成立。

示例 2.5 （邻居平均值）。 平均值 函数计算某个值在一台设备的直接邻居上的加权平均值。

```
1 def average(weight, value) { // 类型: (num) → numval totW  =
2 nfold(+, nbr(0, weight),weight);val totVl  = nfold(+,
3 nbr(0, weight*value),weight*value);totV / totW}
4
5
```

第 2 行通过 `nbr(0, weight)` 交换邻居权重，并使用 `nfold`求和（以本地设备的 `weight`为初始值），计算总权重。第 3 行以类似方式处理加权值，将各邻居的`weight * value` 相加。结果以`totV / totW`的形式返回。该函数包含两次`nbr` 调用（调用 `exchange`），消息类型为 num。XC 能保证消息正确对齐，因此每次交换只会接收来自邻居的对应数据（权重或值），两者互不干扰。

图 7 展示了这一过程。左上方描绘了示例 2.5 中 XC 程序的栈帧和抽象语法树，重点展示了交换、`nfold`及相关函数。右上方展示了一次系统执行：点状箭头连接同一设备中连续的轮次，花括号箭头表示设备间的消息。每一轮都会重新求值相同的程序结构。左下方放大展示了两个

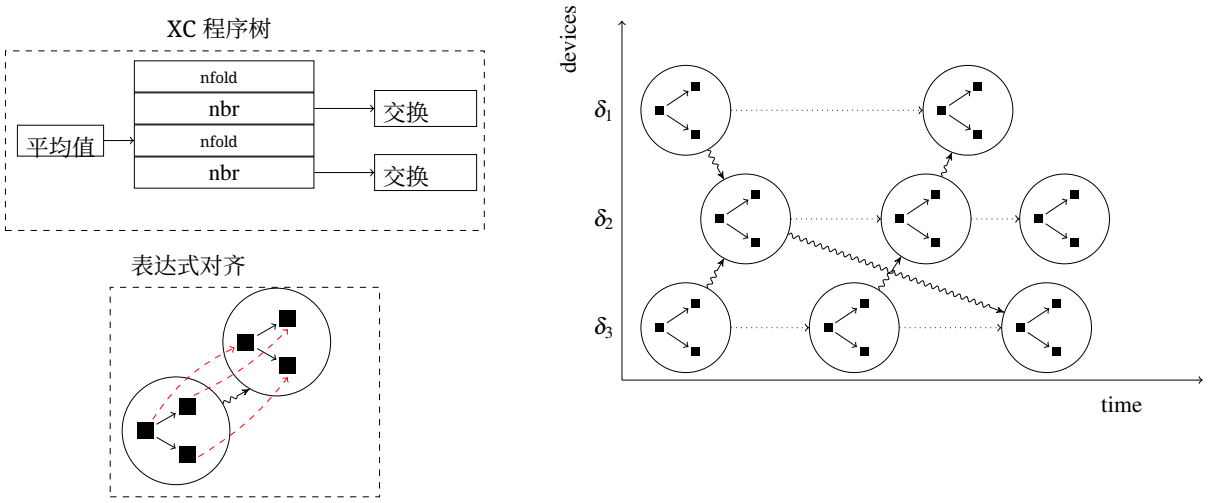

图 7：示例 2.5 的 XC 对齐机制。

在不同设备上求值平均值的轮次，其中相匹配的抽象语法树节点正确对齐。红色虚线箭头突出显示了对应的交换调用如何跨轮次进行通信。

对齐在 XC 中至关重要，因为它支持组合式分布式编程，确保设备之间的数据流遵循代码预期的逻辑结构。

示例 2.6（火灾探测）。 最近火源 函数通过组合平均值 和到……的距离，并使用温度和烟雾传感器输入，来估算到最近潜在火源的距离。

```
1 定义 最近火源(温度, 烟雾) {//类型: (num,num) → num值 信任度 =
2 nfold(+,1,1);值 高温 = 平均值(信任度, 温度) > 60;值 烟雾浓度
3  = 平均值(信任度, 烟雾) > 10;到……的距离(高温和 烟雾浓度) }
4
5
6
```

第 2 行将信任分数计算为与该设备的邻居数量成正比，即 `nfold(+, 1, 1)`。第 3 行和第 4 行利用该信任分数对温度与烟雾的平均值进行加权，并分别检查它们是否超过 60°C 和 10%。第 5 行随后计算到同时满足这两个阈值的节点的距离。在 平均值 与 到……的距离 中会发生多次 交换 调用：得益于对齐，每一次都会接收到来自邻居先前轮次中相同表达式的正确匹配的消息。事实上，若没有对齐，两次调用 平均值 函数所产生的消息将无法区分，因而我们可能会用烟雾水平来计算平均温度，或反之亦然。

XC 支持诸如 `if (cond) {e1} else {e2}`之类的条件语句，其语义会影响通信。由于交换对齐取决于 AST 位置和栈帧，只有采取相同分支的设备才会对齐并交换数据。处于不同分支的设备被视为未对齐，既不共享消息，也不参与诸如 交换 或 `nfold` 之类的运算。这实际上划分了网络：每个分支形成一个隔离的子网络。

例 2.7（域隔离计算）。 考虑一个由服务请求者和提供者组成的网络，其中设备会动态地分成两组：一组执行本地计算（本地），另一组通过网关 节点将计算卸载。我们希望在忽略本地设备的情况下计算到网关的距离。

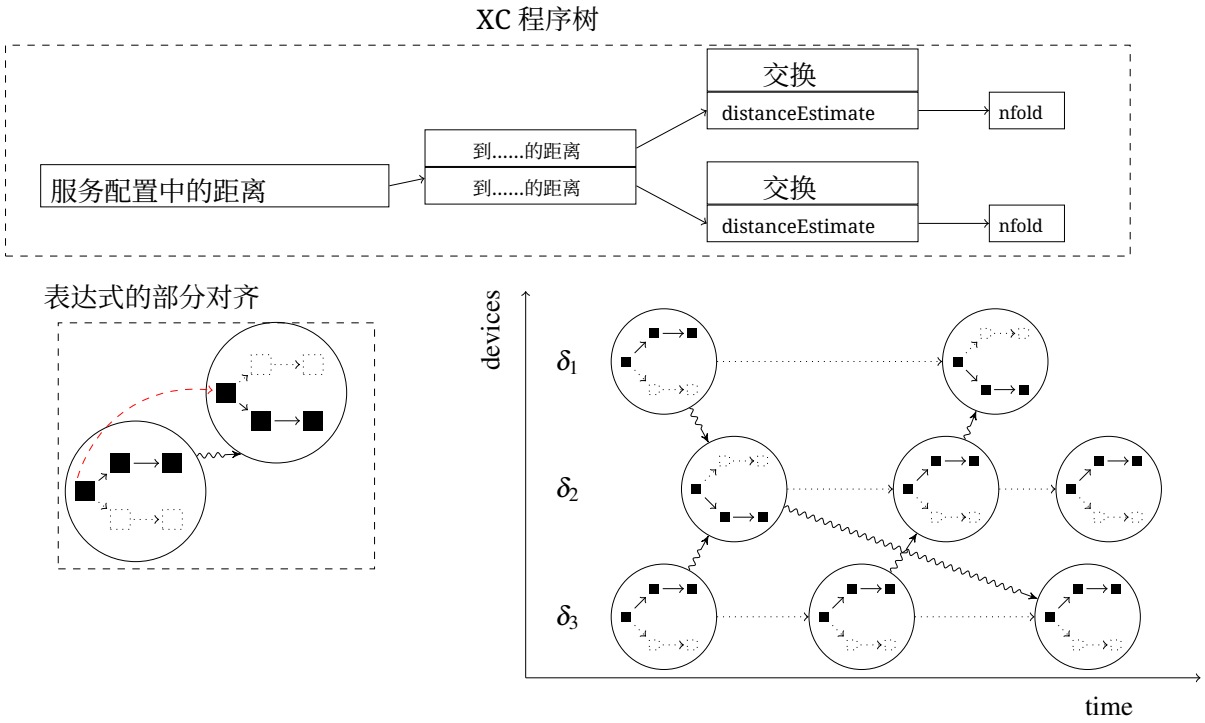

图 8：例 2.7 中带条件分支的 XC 对齐机制。

```
// 类型：(bool, bool) → num
定义 到网关的距离(local, gateway) {
  如果 (local) { 无穷大} 否则{distanceTo(gateway)}
}
```

当本地为 true 时，设备会返回无穷大，从而起到障碍物的作用。否则，它会计算 `distanceTo(gateway)`，其中包含一次交换。本地 组中的设备会跳过此计算，也不参与交换。现在，假设我们希望本地 设备计算到附近请求者的距离，而非本地 设备计算到网关的距离——每组计算时都排除另一组。

```
// 类型：(bool, bool, bool) → bool
定义 服务配置中的距离(local,requester,gateway){
  如果 (本地) { 到请求者的距离(requester) }
  否则 { 到网关的距离(gateway) }
}
```

尽管对到.....的距离 的所有调用都发生在同一个 AST 位置，但它们位于不同的分支中，因此拥有不同的栈帧。因此，XC 确保消息交换彼此分离，避免意外的跨组通信。

图 8 展示了不同设备如何根据本地条件，沿着上述函数定义的不同分支执行。图中显示，在每一轮中，每台设备仅执行一个分支，另一个分支则保持非活动状态（以虚线表示）。分支之前的某些子表达式在所有设备之间仍保持一致（以红色虚线箭头突出显示），但一旦分支发生分歧，交换和聚合就严格在各自的设备组内进行，不存在跨分支通信。

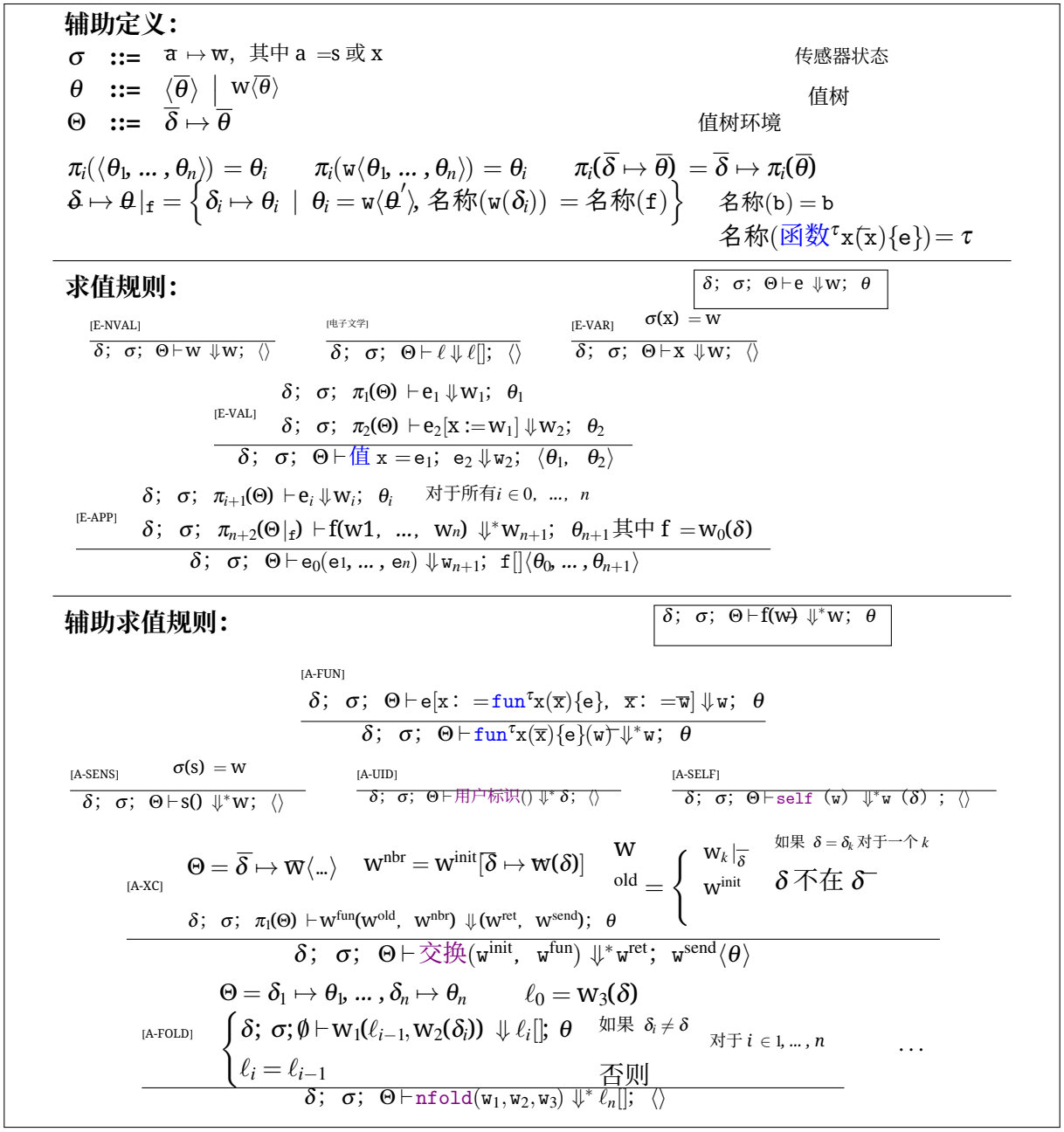

图 9：XC 的设备（大步）操作语义

2.5 语义

XC 表达式的含义由两个互补的语义模型刻画。首先，大步操作设备语义 定义了单台设备在单个执行轮次中的计算方式。其次，指称网络语义 描述了设备执行的本地计算如何共同决定整个网络随时间的演化。

2.5.1 设备级语义

图 9 定义了设备级语义，其形式化表示为判断式 $\delta; \sigma; \Theta \vdash \mathrm{e} \Downarrow \mathrm{w}; \theta$，其含义是“表达式 e 在设备 δ 上，针对传感器值 σ 和值树环境 Θ 求值，得到 nvalue w 和值树 θ”，其中：

- w 是对 e 求值后计算出的最终结果；
- θ 是一个值树，即对 e 求值过程中遇到的 n值的一种表示形式，它是一个有序树，其节点存储交换表达式和函数调用栈帧的结果（见图 9，上方）；
- Θ 是从相邻设备标识符到其最近生成的值树的映射，即 $\delta_1 \mapsto \theta_1$、……、$\delta_n \mapsto \theta_n$；
- σ 将传感器名称 s（以及开放表达式中的自由变量 x[2]）映射到各自的 n值。

此语义使用值树作为通信结构。每棵树都紧凑地表示一轮中生成的所有相关交换数据，同时其形状反映了执行的控制流——通过函数调用和子表达式的嵌套来编码对齐元数据。在求值开始之前，程序中的每个函数表达式函数$\mathtt{x(x)\{e\}}$ 都会被唯一地标记一个名称 τ，该名称用于定义函数相等关系，以确保消息对齐。因此，带注释的函数表示为函数$^{\tau}\mathtt{x(x)e}$；内置函数也同样如此，每个内置函数都有自己的名称ъ。

值树和环境的语法总结于图 9（上）。判断式 $\delta; \sigma; \Theta \vdash \mathrm{e} \Downarrow \mathrm{v}; \theta$的求值规则（图 9，中）大体遵循函数式语言中的标准构造，但扩展后也能通过辅助投影函数 π_i实现邻近值树的对齐。具体而言，对子表达式 e′ 求值时，会使用受限环境 Θ'；该环境通过应用 π_i 从 Θ中的每棵值树提取第i个子树而得到。当 π_i 应用于树 θ时，会返回其第i个子树；应用于完整环境时，则会逐棵树执行该操作。这种对齐机制保证 Θ 仅包含对应于同一子表达式（位于相同 AST 位置和函数调用栈中）的子树。对于函数应用 f(w)，对齐还会进一步细化：将环境缩减为子集 $\Theta|_{\mathrm{f}}$，其中包含先前对同一函数求值所产生的值树，并通过该函数唯一的名称 （如图 9 ¯（上）所定义）来识别。

操作规则如下。规则 [E-NVAL], [E-LIT],和 [E-VAR] 处理常量和自由变量的求值，产生平凡（空）值树和琐碎结果（表达式本身，或 [E-VAR]在传感器状态中的值）。规则 [E-VAL] 对第一个子表达式求值，得到一个 nvalue w_1,，然后将其代入第二个子表达式；求值分别相对于 Θ的两个子树进行。规则 [E-APP] 处理急切函数应用：它使用连续的子树 $\pi_{i+1}(\Theta)$对函数表达式和每个参数求值。随后，使用函数专属子树 $\pi_{n+2}(\Theta|_{\mathrm{f}})$对该应用求值，以确保与同一函数先前的调用保持对齐。此应用交由辅助规则处理，这些规则定义了如何应用不同类型的函数。

规则 [A-FUN] 会先替换函数的参数和名称，再计算其函数体。规则 [A- UID], [A-SELF], 和 [A-SENS]分别直接使用设备 ID、设备身份或来自 σ 的传感器值处理内置函数。规则 [A-FOLD] 定义了对 n值进行折叠的方式：按照一致的顺序枚举 Θ 中的设备，并使用二元函数和每台设备的值依次执行折叠操作，跳过当前设备的值，并将第三个参数 w_3 用作初始值。

[2]请注意为开放表达式定义语义对于支持理论证明中的归纳推理很重要 fs.

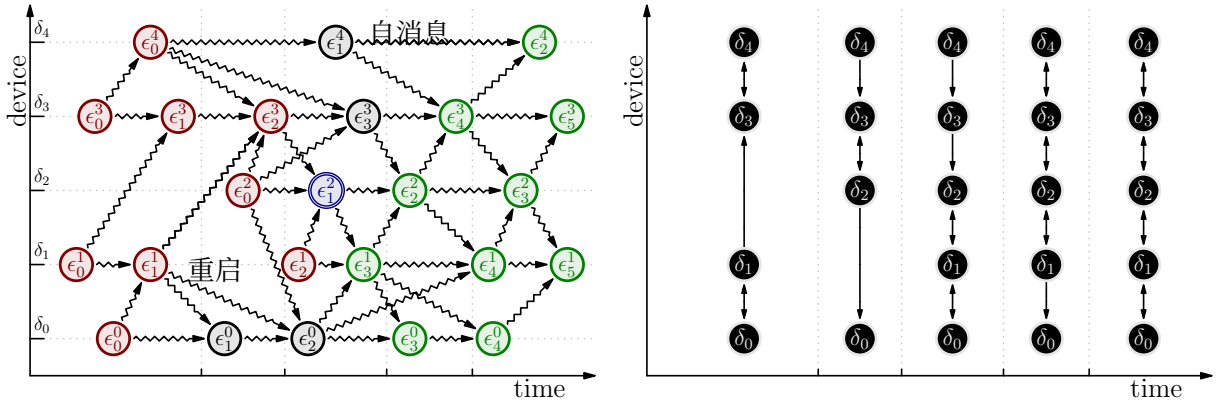

图 10：**左：**一个增强事件结构 $\mathbb{E} = \langle E \rightsquigarrow, d, s \rangle$ 其中 $\mathbb{E} = \{\varepsilon_0^0, ..., \varepsilon_4^0, \varepsilon_0^1, ..., \varepsilon_5^1, \varepsilon_0^2, ..., \varepsilon_4^2, \varepsilon_0^3, ..., \varepsilon_3^3, \varepsilon_0^4, ..., \varepsilon_5^4\}$ 由 24 个事件组成，且 $d(\varepsilon_j^i) = \delta_i$；它描绘了事件（圆形节点）、消息关系（花括号箭头）、设备 $\delta_0, ..., \delta_4$（y 轴），且每个圆形节点都标记为 $\mathbb{E}$ 中所描绘的事件。颜色表示相对于参考事件 ε_1^2（蓝色双圆圈）的因果关系，并将事件划分为因果过去（红色）、因果未来（绿色）和并发（无序，黑色）。**右：**设备上一种（可能的）演化网络拓扑，它可能产生 $\mathbb{E}$（假设某些选定的消息被丢弃）。

最值得注意的规则 [A-XC], 规定了交换表达式的行为。如第 2.3.3 节所述，该规则执行关键的通信步骤。记号 $\mathrm{w}^{\mathrm{init}}[\delta \mapsto \mathrm{w}(\delta)]$ 表示本地交换值，该值以 $\mathrm{w}^{\mathrm{init}}$ 初始化，并根据来自邻居 $\underline{\delta}$ 的传入消息进行更新。为计算旧值 $\mathrm{w}^{\mathrm{old}}$，规则区分两种情况：*(i)* 如果设备 δ 未出现在 Θ 中，则表示表达式在当前轮次中首次计算，因此将 $\mathrm{w}^{\mathrm{old}}$ 设为初始值；*(ii)* 否则，如果 δ 出现在邻居之中（即 $\delta = \delta_k$），则恢复上一轮发送的值 w_k，并通过 $\mathrm{w}_k|_D$将其定义域限制为当前邻域，这相当于将 w 中任意不属于 D 的设备对应的值设为 w 的默认值。旧值 $\mathrm{w}^{\mathrm{old}}$ 和邻居消息集合 $\mathrm{w}^{\mathrm{nbr}}$ 都会传递给更新函数 $\mathrm{w}^{\mathrm{fun}}$。结果元组的第一个元素作为交换表达式的结果返回，第二个元素则存储在值树的根节点中，以便在下一轮共享。其他内置函数的语义属于标准定义，且独立于 Θ，因此为简洁起见不再赘述。

2.5.2 网络级语义

网络执行程序 $\mathrm{e}_{\texttt{main}}$ 的行为通过事件来建模，这些事件表示各个设备按照设备级语义执行的原子轮次。将这些事件综合起来考量——涵盖空间维度（即不同设备）和时间进程（即事件的先后顺序）——便可定义程序的执行轨迹，仿佛它运行在一个统一的“聚合机器” [19]上。在此框架中，设备间的通信通过消息传递 关系 $\rightsquigarrow$来描述，该关系规定了信息如何从一个事件传递到另一个事件。为形式化这一概念，我们采用增强事件结构这一概念，它在经典的事件结构基础上 [35] 扩展而来，属于并发理论的一部分，并通过为每个事件添加上下文元数据予以丰富：相关联的设备标识符，以及事件发生时的本地传感器状态。

定义 2.8（增强事件结构）。一个增强事件结构 $\mathbb{E} = \langle E \rightsquigarrow, d, s \rangle$是一个元组，其中：

- E 是一个可数的事件 ε集合，
- $\rightsquigarrow \subseteq E \times E$ 是一种消息关系，
- $d: E \to \Delta$ 是从事件到其发生所在设备的映射，并且
- $s: E \to S$ 将每个事件 ε 映射到传感器状态 σ（如设备级语义中所示），

满足以下条件:

- $\varepsilon_1 = \varepsilon_2$ 当$d(\varepsilon_1) = d(\varepsilon_2)$，且 $\varepsilon_i \rightsquigarrow \varepsilon$ 对于 $i = 1, 2$（即，事件的前驱必须发生在不同的设备上）；
- 不存在序列 $\varepsilon_1 \rightsquigarrow \ldots \rightsquigarrow \varepsilon_n \rightsquigarrow \varepsilon_1$（即，$\rightsquigarrow$ 关系是无环的）；
- 对于所有 ε，在 $\rightsquigarrow$ 中能够到达 ε 的事件集合 $X_\varepsilon = \{\varepsilon' \in E \mid \varepsilon' \rightsquigarrow \ldots \rightsquigarrow \varepsilon\}$ 都是有限的（即，$\rightsquigarrow$ 是良基且局部有限的）。

若 $\varepsilon' \rightsquigarrow \varepsilon$，则称事件 ε' 是事件 ε的供应者 。我们还将 $\rightsquigarrow$ 的传递闭包称为因果偏序 $< \subseteq E \times E$。

图 10（左上）给出了一个示例，该示例基于图 10（右上）所示的动态网络。此例展示了因果关系如何相对于参考事件 ε划分事件空间，界定出“因果过去”、“因果未来”，以及与 ε 并发的一组事件（即其“现在”）。在执行过程中，ε 处的计算可以依赖其过去产生的数据，并可能进而影响其未来。消息关系 $\rightsquigarrow$ 定义了这种影响何时能够直接发生，即一个事件发送的消息何时能够被另一个事件接收。

该模型反映了由相互作用的物理设备组成的分布式系统的运行动态。单个设备的演化对应于由 $\rightsquigarrow$ 连接、且带有相同设备 ID 标记的一串事件；设备间通信则类似地表示为连接带有不同设备标记的事件的 $\rightsquigarrow$链接。重要的是，该模型不依赖于全局时钟、同步执行或事件按规律定时发生等假设。遵循先前文献 [36, 39, 48], ，我们将值场定义为从设备到值的映射，用于表示全局快照，其中包含每台设备上最近一次事件产生的输出。此类值场随时间的演化构成一种称为时空值的全局结构，它为增强事件结构中的每个事件赋予一个值。

定义 2.9（时空值）。令 $\mathbb{E} = \langle E \rightsquigarrow, d, s \rangle$ 为一个增强事件结构，且$\mathbf{V}_T$为类型T的 n值的定义域。一个时空值（ $\mathbb{E}$ 中类型为T的时空值）是一个将事件映射到 n值的函数$\Phi: E \to \mathbf{V}_T$。

在此设定下，对增强事件结构 $\mathbb{E}$ 上的程序 $\mathsf{e}_{\mathtt{main}}$ 进行求值，会通过迭代应用设备级操作规则诱导出一个时空值。

定义 2.10（事件结构上的程序求值）。设 e 为在假设 $\mathscr{A}$下类型为 T 的 XC 表达式。设 $\mathbb{E}$ 为一个增强事件结构，其 s 包含 e 中出现的每个传感器和自由变量的适当类型的值。

设 $\theta^{\mathbb{E}}_{\mathsf{e}}: E \to \Omega$（其中 Ω 是所有值树的集合）和 $\Phi^{\mathbb{E}}_{\mathsf{e}}: E \to \mathbf{V}_T$ 通过对 E 中的ε进行归纳来定义，使得 $d(\varepsilon)$；$s(\varepsilon)$；$\Theta_\varepsilon \vdash \mathsf{e} \Downarrow \Phi^{\mathbb{E}}_{\mathsf{e}}(\varepsilon)$；$\theta^{\mathbb{E}}_{\mathsf{e}}(\varepsilon)$，其中 $\Theta_\varepsilon = \{d(\varepsilon') \mapsto \theta^{\mathbb{E}}_{\mathsf{e}}(\varepsilon') : \varepsilon' \rightsquigarrow \varepsilon\}$。于是我们说 $\Phi^{\mathbb{E}}_{\mathsf{e}}: E \to \mathbf{V}_T$ 是表达式 e 在 $\mathbb{E}$上的求值。

需要注意的是，每个事件 ε 都是相对于一个固定的值树环境 Θ_ε进行求值的，该环境在整个事件执行过程中保持不变。[3] 此外，并非所有程序都能保证

[3]这为传入消息建模了一种缓冲抽象，通过假定一致的输入快照来确保每一轮次内的确定性。

对事件结构 $\mathbb{E}$产生有效解释。具体而言，如果 $\mathtt{e}_{\mathtt{main}}$ 的执行在某个事件 ε中发散，则该结构的语义变为未定义。在现实场景中，这对应于某设备未能完成其计算：从网络的角度来看，这种情况与设备已掉线无法区分。由于非终止执行从不发出输出消息，它们无法对通信图做出贡献，因此在我们的模型中不构成有效事件。因此，语义自然地排除了此类病态情形。

因此，程序 $\mathtt{e}_{\mathtt{main}}$ 的语义可以从局部和全局这两个互补视角来理解，这呼应了自组织系统 [18, 32, 41] 中常见的微观／宏观二分法。从局部（或微观）视角来看，基于设备级语义，在特定轮次 ε 中对 $\mathtt{e}_{\mathtt{main}}$ 求值，意味着在设备 δ 上执行该表达式，使用当前的传感器状态 σ 和消息环境 Θ——后者编码了来自相邻事件的输入——并生成一个输出值 $\mathtt{w}$ 及与之关联的消息结构 θ，以供传播。从全局（或宏观）视角来看，按照网络级语义的形式化定义，程序 $\mathtt{e}_{\mathtt{main}}$ 是作用于增强事件结构之上的部分函子。给定输入 $\mathbb{E} = \langle E \rightsquigarrow, d, s \rangle$,[4]，其语义会得出一个时空值 Φ，该值封装了程序在时空范围内所有事件上的输出。

3 XC 设计决策与性质

3.1 设计决策

XC 旨在鼓励开发者通过定义和组合天然具备故障韧性的函数块来编程。具体而言，XC 具备以下三个总体特征。

3.1.1 集体视角

该交换机制允许程序员指定一种逻辑，以集体地对邻居的消息进行运算，通过逐点操作和 `nfold`以同质化的方式表达行为，而无需假设邻居的数量或身份；同时，基于轮次的执行模型支持进展以及局部变化的纳入（最终将变化从一个邻域传播到另一个邻域，直至覆盖全局系统）。因此，每种行为都可以描述为一个函数，该函数封装了实现协调结果所需的处理和通信。

3.1.2 自动故障吸收

如果某个设备发生故障或某条消息丢失，交换 会以对程序员透明的方式处理该故障：该设备可能不会出现在某个设备下一次触发时的邻居之中。因此，XC 支持编写固有地容错且对设备动态加入和离开邻居集合（对程序员而言是透明的）具有弹性的程序（例如，因为它们物理位置发生变化或重启）。尽管如此，需要注意的是，XC 作为一种图灵完备语言，本身并不会强制实现容错行为。程序员仍然可以表达非弹性逻辑，而且在具有这种表达能力的语言中，此类行为事实上是不可避免的。然而，这类非弹性程序通常并不符合惯用写法。相反，对于该语言中定义明确的惯用子集，可以建立更强的保证，如第 3.2 节所述。

[4]该函子是部分的，因为如果程序在 $\mathbb{E}$中的任何事件上发散，其解释就会失败。

3.1.3 组合性

由对齐所支配的、介于交换 与函数抽象/组合及分支之间的交互，支持弹性集体行为的组合式编程。具体而言，表达集体行为弹性块的函数（参见示例 2.4 中的梯度）可以被定义并组合起来，以构建更复杂的块，同时保留容错和自稳定等性质（见第 3.2 节）。

3.2 性质

本节列出 XC 的主要性质。以下性质已在 [7]中得到证明，适用于一个呈现了更简单版本的 交换的 XC 版本；将证明扩展到第 2 节中交换的版本（该版本在 [5]中引入）是很直接的。

- 类型健全性：类型正确的程序不会在任何一次触发时陷入停滞。
- 自稳定：使用该语言某个惯用子集编写的程序，在发生瞬态变化后能够最终恢复并稳定；也就是说，如果邻接关系和传感器的值趋于稳定，那么任何一次触发所产生的值最终都会稳定。

此外，在 [7] 中还指出，由于每个 FC 程序都可以编码到 XC 中，因此文献中针对 FC 语言子集所证明的以下结果：

- 结果与设备密度无关 [21],
- 实时误差保证 [13],
- 表达所有物理一致的计算的能力（时空普适性） [4], 以及
- 监测时空逻辑公式的能力 [8, 11],

同样适用于 XC 语言的相应子集。

4 FCPP：用于交换演算的 C++ 库和工具链

FCPP 是由都灵大学计算机科学系开发的 XC C++ 库和工具链，旨在实现*(i)* 高效性和*(ii)* 对定制架构的支持。对于*(i)*，FCPP 借助 C++ 的编译时优化，并在裸机上执行。邻近值通过 C++ STL 中的`vector<T>` 实现（这种方案在线性折叠和逐点操作方面比哈希映射更高效）。通信依赖消息序列化，并将消息传递给网络驱动程序（对于低层设备，这通常是设备专用 API）。对于*(ii)*，FCPP 利用适用于定制架构的 C/C++ 编译器，并通过尽量减少依赖项来简化部署。例如，将序列化头文件、编译时类型检查工具、多类型值映射、选项类型、四元数、带标签元组等纳入实现中。该 FCPP 实现已在多种没有外部依赖项的后端上经过测试并投入使用，包括：

- 对分布式网络进行批量仿真，并生成图表以汇总网络在多次运行中的行为 [2];
- 通过 OpenGL 对分布式网络进行图形化交互式 3D 仿真 [14];
- 在高性能计算环境中处理大型基于图的数据上的 XC 算法 [12];

- 部署于运行 Contiki 操作系统或 MIOSIX 的微控制器架构上 [15, 47], , 并可与安卓手机互操作（相关工作仍在进行中）。

以下代码片段展示了第 2.3 节中 XC 函数 *distanceEstimate* 和到……的距离的实现，体现了在 C++ 程序中使用 FCPP 的方式。

```
1 FUN real_t distanceEstimate(ARGS,nvalue<real_t>n){CODE 2
returnmin_hood(CALL, n  +node.nbr_dist(),INF); 3 } 4 5 FUN real_t
distanceTo(ARGS,boolisSource){CODE 6 return 交换(CALL,INF，[&]
(nvalue<real_t>n){ 7
returnmux(isSource,0,distanceEstimate(CALL,n)); 8 } 9 }
```

借助宏（例如 FUN、ARGS、CODE、CALL），FCPP 库提供了作为内部领域特定语言（Do- main Specific Language，DSL）的 XC，也就是说，开发者可以在同一程序中混合使用“XC 代码”和 C++ 。XC 的基本运算符和内置函数（例如min_hood、交换、mux）被定义为 C++ 函数，利用模板实现多态。尽管代码揭示了 C++的一些不可避免的特性，这些特性与 XC 的形式化定义有所不同（例如，显式类型、返回语句等），但该代码片段看起来与第 2.3 节中展示的相应 XC 代码非常接近。

5 FCPPat 工作

本节将介绍，借助我们参与的一些项目，我们如何已将 XC“装上轮子”，应用于自主机器人；以及我们如何努力让它“飞”起来，应用于无人机。

RoboNG RoboNG 项目[5] 由欧盟通过 NextGenerationEU 在 NODES 生态系统框架内资助。项目于 2023 年底启动，历时 14 个月，至 2025 年 2 月结束，涉及两家主要工业合作伙伴，以及都灵大学计算机科学系这一额外的学术合作伙伴。

该项目所针对的用例由 SAGAT 提供。SAGAT 是都灵机场的管理公司；都灵机场是为意大利都灵市提供服务的国际机场。SAGAT 提出的用例是机场面临的一项重要问题，即及时检测并清除外来物体（外来物体碎片）。外来物体是可能散落在飞机停放区域和滑行区域中的小型物体，可能造成灾难性后果，甚至导致人员死亡[6]。由于由人工持续检查相关区域以检测外来物体碎片，既重复又耗费成本，而且容易出错，因此 RoboNG 项目将原型实现一种能够以（半）自主方式进行外来物体碎片检测的系统，设定为主要目标。

所提出的解决方案涉及一组自主移动机器人，负责覆盖相关区域，同时利用机载传感器检测潜在的外来物体，并向人工操作员发送警报。为了能够在环境中导航、感知环境并与操作员通信，每台机器人配备有：

- 用于观测机器人正前方区域的高分辨率摄像机

[5] https://ecs-nodes.eu/en/1-aerospace-and-sustainable-mobility/progetti-imprese/robong [6] https://www.fodcontrol.com/what-is-fod/




图 11：为 RoboNG 配备的 Jackal（左）和 Create3（右）机器人。

- 基于人工智能（在机载计算机上运行）的计算机视觉（CV）流程，用于检测外来物体
- 自主导航软件（支持障碍物规避和碰撞规避），由激光雷达和/或额外摄像头提供支持
- 通过路由器接入本地无线网络的 WiFi 连接

此外，与我们的科学研究兴趣更相关的是，机器人必须能够在机载设备上运行 FCPP 应用程序，以实现机器人之间的通信与协调。作为主要的机器人核心，FCPP 应用程序从计算机视觉流水线获取信息，并访问本地机器人以获取遥测数据和发送控制指令。

图 11 展示了用于原型系统实现的两种机器人。Clearpath Robotics 生产的 Jackal 是一款功能强劲的小型漫游车，适用于户外探索。iRobot 生产的 Create3 是一个基础平台，可用于研究，主要适用于室内环境。在都灵机场举行的 RoboNG 最终演示中，两台机器人协同覆盖了一小片区域，并成功检测和报告了几个外来物体。分布式覆盖算法通过运行在机载 FCPP 应用程序上的 XC 程序实现。在仿真中，使用 Gazebo 仿真器时，XC 程序已证明能够成功协调多达八台 Jackal 机器人执行外来物体检测任务。

RoboApp RoboApp 项目[7], 由欧盟通过 NextGenerationEU 在 NODES 生态系统框架内资助。该项目于 2024年初启动，截至撰写本文时已接近结束（为期 14 个月）。合作方包括经济与统计系，以及都灵大学计算机科学系；两者将部分工作外包给了小型外部企业。

SAGAT 为该项目提出了另一个颇具吸引力的用例，场景涉及登机口，而非室外区域。具体来说，由于种种原因，登机口可能会出现长长的排队人群，队伍从蛇形隔离栏中溢出，延伸至通行区域。人工操作员应及时疏导这些队伍，因为它们会妨碍人们通行；但队伍形成时，登机口往往还没有人值守。RoboAPP 项目的主要目标是实现一个系统原型，使其能够检测登机口处溢出的队伍，并在需要时向人工操作员发出警报。

[7]https://e cs-nodes.eu/en/1-aerospace-and-sustainable-mobility/progetti-accademici/roboapp



图 12：经 RoboAPP 计算机视觉流水线处理的帧。

所提出的解决方案与 RoboNG 的方案有一些相似之处，但也存在关键差异，使其本身成为一项全新的研究挑战。我们再次考虑由一组自主移动机器人（AMR）负责监测闸门。这些机器人配备了传感器和软件，能够导航并通过摄像头观察周围环境，同时运行用于分布式控制和协调的 FCPP 应用程序。当计划表（目前为固定计划）要求监测某个闸门时，机器人团队会执行分布式 XC 算法，选出两台或更多机器人共同执行任务。选出的机器人会从停机区域（空闲时在此集结）移动到闸门附近预先设定的位置，并从那里开始观察。到这一步为止，该方案与 RoboNG 相比并无新意。真正的创新在于机器人相互协作观察现场的方式，以及向人工操作员发出警报时所采用的决策流程。

首先，在闸口附近检测排队人群，是一项比在大致均匀且空旷的区域检测外来物更具挑战性的计算机视觉任务，因此，计算机视觉流水线（在 Jetson Nano 机载计算机上运行）也相应地更加复杂。具体而言，人工智能模型需要检测单个人员及其边界框；随后，还需通过进一步处理距离和时间，检测出稳定聚集群体，这些群体可能表明存在排队人群（见图 12）。更重要的是，每台机器人都从自己的视角感知场景，因此可能对是否存在排队溢出得出不同结论。我们再次利用一个在机载设备上运行的 FCPP 应用程序，该程序实现了分布式 XC 算法，使机器人达成共识（最简单的情况是采用分布式多数表决）。

然而，即使汇总多个视角得出的共识，有时也无法可靠地完成当前任务，因为我们需要识别的情况涉及空间、时间和常识知识，而这些并非计算机视觉模型所具备的。例如，我们希望避免在实际存在队列、但队列并未阻碍他人通行的情况下发送警报（例如，人们排在蛇形隔离栏内，或以恰当的方式排在蛇形隔离栏外）。因此，我们在系统中增加了一个额外组件，即在局域网上的一台个人电脑上运行的人工智能智能体 [51] 应用，该应用基于多模态小语言模型（SLM）。当机器人团队一致认为可能 发生了需要通知人工操作员的紧急情况时，便会调用人工智能应用作出最终判断。这种架构展示了聚合

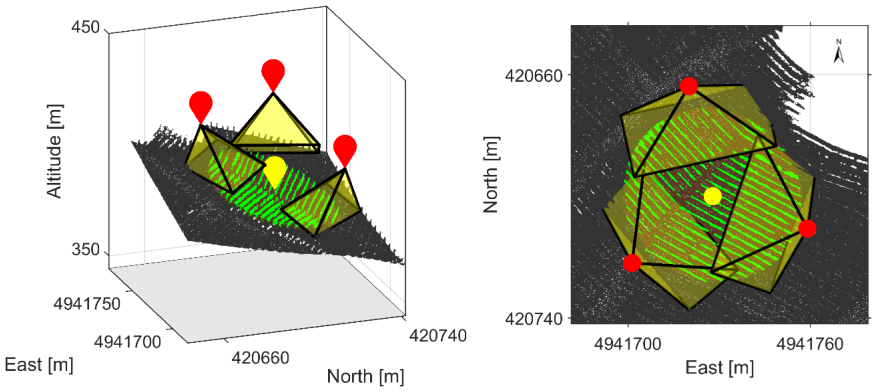

图 13：葡萄园中的一辆无人地面车辆（黄色）和由三架无人机组成的集群（红色）（来自 [45]。

编程与人工智能智能体如何协作，以提升系统的效率和准确性。

农业科技 我们参与了农业科技国家农业技术研究中心分支 6[8], ，该中心由欧盟 - NextGenerationEU 资助。它于2022年年底启动，在撰写本文时即将结束（为期 3 年）。在众多合作伙伴中，我们尤其与都灵大学农业、林业与食品科学系（DISAFA）在任务 6.1.3 针对不同农业场景的新型管理模式的案例研究与基于指标的评估中开展了合作。

与我们科研兴趣相关的任务部分，是研究无人机集群在农业中的应用。在 DISAFA 领域专家的指导下，我们实现了若干自主无人机集群的仿真，这些集群由以 XC 实现的 AP 算法进行控制与协调。更具体地说，我们处理了以下问题：给定一辆名为作业车辆的自主车辆（空中或地面），负责沿预定轨迹穿越农作物区域以执行其任务（例如除草或修剪），设计算法来协调一个无人机集群，使其必须伴随作业车辆沿其轨迹行进，并对周边区域进行监视，以确保安全。我们探讨了以下变体：

- 单作业车辆与多作业车辆：从 AP 的角度来看，当作业车辆不再是单个，而是有多个作业车辆，且每个作业车辆都对应一个集群，同时各架无人机可以在不同集群之间移动以优化监测时，问题就变得有趣得多。
- 嵌入 FCPP 的轻量级仿真（物理现象以高度抽象的方式进行模拟，无人机则表示为球体），以及使用 Gazebo 进行的逼真仿真（物理现象的模拟更加真实，无人机拥有三维模型，甚至可以通过软件在环的方式运行其原始机载控制固件）

在 [45], 中，我们报告了与 DISAFA 合作的研究结果。我们通过实验研究了基于单作业车辆和轻量级仿真的实现方案，在葡萄园中为一辆无人地面车辆提供监视的有效性，并将地形的详细三维地图以及安装在无人机上的摄像机视场纳入考量（图 13）。

虽然农业科技项目中的工作仅限于仿真，但我们最近启动了一个为期六个月、由自筹资金支持的项目，名为 DRONEX。在该项目中，我们将 FCPP 部署到一群纳米无人机上，即 Bitcraze 的 Crazyflies（CF）。CF 的小巧体型对

[8] https://agritechcenter.it/

我们能够安装在无人机上、用于运行 FCPP 和其他自主控制软件的开发板的计算能力（以及能耗）造成了严格限制。我们选择了 Luckfox Pico mini 开发板。尽管体积小巧，它们配备了 ARM Cortex-A7 处理器，可运行嵌入式 Linux，甚至还搭载了可用于机载人工智能的神经处理单元（NPU）。FCPP 对中央处理器，尤其是内存的需求极低，因此我们得以将其部署到如此小巧的开发板上，并成功运行。

6 相关工作

在开发集体自适应系统（例如机器人群体或无线传感器网络（WSN））时，若能在全局层面编程系统，并抽象掉计算节点之间底层的异构性及显式通信机制，将会很有帮助。集体导向的分布式系统工程（DSE）方法，例如面向集体自适应系统（CAS）的方法，以及宏编程，都可以实现这一目标。关于集体系统软件工程方法的全面且最新的综述，请参阅 [24]。

集体自适应系统（CAS）指由大量相互交互的组件构成的系统，这些组件在没有任何形式的集中式控制的情况下协同完成复杂任务，并在动态环境中运行 [22, 34, 43]。为集体自适应系统的工程设计而提出的语言中，有相当一部分侧重于所谓的集成体的规范与分析，即由设备构成的动态集合，这些设备具有丰富的交互模式，并处于复杂环境之中 [52]。一个代表性例子是基于随机过程代数的 *CARMA* [37], 。在集成体编程方面，ASCENS 项目提出了*SCEL* [44], ，这是一种基于过程代数的抽象语言，用于描述单个组件及其聚合体的行为。SCEL 依赖于基于属性的通信 [1] ，并支持针对知识库和自适应策略进行参数化的规范。

另一种与集体导向 DSE 相关的方法是宏编程 [23]。这些方法提供集体层面的抽象，其程序构造定义在整个设备集合之上，从而有效地将分布式系统本身视为执行目标。基于宏编程方法的代表性方案包括：*Buzz* 群体编程领域专用语言 [46], ，它将群体 作为一等实体；*AErlang* [42], ，一种基于 SCEL、利用基于属性的通信的领域专用语言；*Chronus* [50], ，一种面向无线传感器网络的时空编程语言，其程序通过“时空切片”指定数据查询；以及聚合编程 [20],，它依赖计算场 [40]这一概念。

7 结论与未来工作

我们介绍了 AP 范式及其基础模型 XC，并简要概述了若干工业案例研究；这些案例涉及 FCCP C++ 库以及 XC 的内部领域特定语言实现。本文的介绍面向研究兴趣与 λ 演算和类型论相关的研究人员，例如 Stefano Berardi。当前研究正着手解决基础层面的若干问题，例如：

- 为 XC 增添用于利用设备异构性的声明式机制（目前只能依赖`if`-表达式来实现）；
- 进一步研究实时保证；以及

• 设计表达能力更强的XC可监测时空逻辑；

以及实践方面，例如：

• 计划对 FCPP 进行重构；

• 目前，一群来自博洛尼亚大学计算机科学与工程系的研究人员正在开发 XC 的 Kotlin 库和内部领域特定语言版本（https:// collektive.github.io/）；以及

• 计划通过真实世界的用例进一步验证这一方法。

致谢

本出版物是 NODES 项目的一部分，该项目已获得由欧盟 - NextGenerationEU 资助的国家复苏与韧性计划中大学与研究部 – M4C2 1.5 的资助（资助协议编号 ECS00000036）。本研究得到意大利PRIN项目“CommonWears”（2020HCWWLP）的支持，并在农业技术国家研究中心内开展，同时获得了欧盟 NextGenerationEU（国家复苏与韧性计划（PNRR）– 任务4组成部分2，投资1.4 – 第1032号法令（2022年6月17日），CN00000022）的资助。本稿仅反映作者的观点和意见，欧盟和欧盟委员会均不对此承担责任。

参考资料

[1] Yehia Abd Alrahman、Rocco De Nicola与Michele Loreti（2020）：依托基于属性的通信，为集体自适应系统编程交互。科学计算与编程192，第102428页，doi:10.1016/J.SCICO.2020.102428。[2] Giorgio Audrito（2020）：*FCPP*：高效且可扩展的场演算框架。载于：第*1*届国际自主计算与自组织系统会议（*ACSOS*）论文集，IEEE计算机学会，第153–159页，doi:10.1109/ACSOS49614.2020.00037。[3] Giorgio Audrito、Jacob Beal、Ferruccio Damiani、Danilo Pianini与Mirko Viroli（2020）：利用共享算子进行基于场的协调。计算机科学中的逻辑方法16(4)，doi:10.23638/LMCS- 16(4:1)2020。[4] Giorgio Audrito、Jacob Beal、Ferruccio Damiani与Mirko Viroli（2018）：场演算的时空普适性。载于：协调模型与语言，计算机科学讲义10852，施普林格，第1–20页，doi:10.1007/978-3-319-92408-3_1。[5] Giorgio Audrito、Daniele Bortoluzzi、Ferruccio Damiani、Giordano Scarso与Gianluca Torta（2024）：面向*XC*的增强型交换算子。载于Ilaria Castellani与Francesco Tiezzi编：协调模型与语言，施普林格自然瑞士，Cham，第129–145页，doi:10.1007/978-3-031-62697-5_8。[6] Giorgio Audrito、Roberto Casadei、Ferruccio Damiani、GuidoSalvaneschi与Mirko Viroli（2022）：基于*XC*的分布式系统函数式编程。载于：第*36*届欧洲面向对象编程会议（*ECOOP 2022*），*LIPIcs*222，达格施图尔城堡，第20:1–20:28页，doi:10.4230/LIPIcs.ECOOP.2022.20。[7] Giorgio Audrito、Roberto Casadei、Ferruccio Damiani、Guido Salvaneschi与Mirko Viroli（2024）：*eX- change*演算（*XC*）：面向分布式集体系统的函数式编程语言设计。系统与软件杂志210，第111976页，doi:10.1016/j.jss.2024.111976。[8] Giorgio Audrito、Roberto Casadei、Ferruccio Damiani、Volker Stolz与Mirko Viroli（2021）：面向信息物理系统空间属性的自适应分布式监视器。系统与软件杂志175，第110908页，doi:10.1016/j.jss.2021.110908。

[9] Giorgio Audrito、Roberto Casadei、Ferruccio Damiani与Mirko Viroli（2017）：用于最优自愈梯度的组合块。载于：自适应与自组织系统（SASO）2017，IEEE，IEEE计算机学会，第91–100页，doi:10.1109/SASO.2017.18。[10] Giorgio Audrito、Roberto Casadei、Ferruccio Damiani与Mirko Viroli（2023）：面向邻居的计算：通过函数式编程与Scala应对大规模分布和适应性。计算机科学中的逻辑方法19(1)，doi:10.46298/lmcs-19(1:6)2023。[11]Giorgio Audrito、Ferruccio Damiani、Volker Stolz、Gianluca Torta与Mirko Viroli（2022）：利用past-CTL与场演算进行分布式运行时验证。系统与软件杂志187，第111251页，doi:10.1016/j.jss.2022.111251。[12] Giorgio Audrito、Ferruccio Damiani与Gianluca Torta（2022）：推动聚合编程走向云。载于：第11届形式方法、验证与确认应用国际研讨会（ISoLA 2022）论文集，第三部分，LNCS 13703，施普林格，第301–317页，doi:10.1007/978-3- 031-19759-8 19。_[13] Giorgio Audrito、Ferruccio Damiani、Mirko Viroli与Enrico Bini（2018）：利用场演算进行分布式实时最短路径计算。载于：2018年IEEE实时系统研讨会（RTSS），IEEE计算机学会，第23–34页，doi:10.1109/RTSS.2018.00013。[14] Giorgio Audrito、Luigi Rapetta与Gianluca Torta（2022）：利用FCPP对聚合系统进行可扩展三维仿真。载于：协调模型与语言——第24届国际会议COORDINATION 2022论文集，LNCS 13271，施普林格，第55–71页，doi:10.1007/978-3-031-08143-9_4。[15]Giorgio Audrito、Federico Terraneo与William Fornaciari（2023）：FCPP+Miosix：将聚合编程扩展至嵌入式系统。IEEE并行与分布式系统汇刊34(3)，第869–880页，doi:10.1109/TPDS.2022.3232633。[16] Giorgio Audrito与Gianluca Torta（2024）：用FCPP聚合一切。科学计算与编程231，第103026页，doi:10.1016/J.SCICO.2023.103026。[17] Giorgio Audrito、Mirko Viroli、Ferruccio Damiani、Danilo Pianini与Jacob Beal（2019）：计算场高阶演算。ACM计算逻辑汇刊20(1)，第5:1–5:55页，doi:10.1145/3285956。[18] Jacob Beal、Stefan Dulman、Kyle Usbeck、Mirko Viroli与Nikolaus Correll（2013）：组织聚合：空间计算语言。载于：领域专用语言的理论与实践：最新进展，第16章，IGI Global，第436–501页，doi:10.4018/978-1-4666-2092-6.ch016。[19] Jacob Beal、Danilo Pianini与Mirko Viroli（2015）：物联网聚合编程。IEEE计算机48(9)，doi:10.1109/MC.2015.261。[20] Jacob Beal、Danilo Pianini与Mirko Viroli（2015）：物联网聚合编程。计算机48(9)，第22–30页，doi:10.1109/MC.2015.261。[21] Jacob Beal、Mirko Viroli、Danilo Pianini与Ferruccio Damiani（2017）：适应物联网中的设备分布。ACM自治与自适应系统汇刊12(3)，第12:1–12:29页，doi:10.1145/3105758。[22] Giacomo Cabri与Emma Hart（2014）：第二届集体自适应系统基础FOCAS研讨会。载于：第八届IEEE自适应与自组织系统国际会议研讨会（SASOW 2014），英国伦敦，2014年9月8日至12日，IEEE计算机学会，第6–7页，doi:10.1109/SASOW.2014.8。[23] Roberto Casadei（2023）：宏编程：概念、现状及宏观行为建模的机遇。ACM计算调查55(13s)，第275:1–275:37页，doi:10.1145/3579353。[24] Roberto Casadei、Gianluca Aguzzi、Giorgio Audrito、Ferruccio Damiani、Danilo Pianini、Giordano Scarso、Gianluca Torta与Mirko Viroli（2025）：集体网络物理生态系统的软件工程。ACM软件工程与方法汇刊34(5)，doi:10.1145/3712004。[25] Roberto Casadei、Gianluca Aguzzi、Danilo Pianini与Mirko Viroli（2023）：利用ScaFi编程（与学习）自适应与自组织行为：面向群体、边缘云生态系统及其他场景。载于：IEEE国际自主计算与自组织系统会议（ACSOS

2023 - Companion，多伦多（加拿大安大略省），2023年9月25日至29日，IEEE，第33–34页，doi:10.1109/ACSOS- C58168.2023.00032.[26] Roberto Casadei & Mirko Viroli (2016)：迈向 Scala 中的聚合编程。载于：第一届分布式计算编程模型与语言研讨会，PMLDC '16，ACM，美国纽约州纽约市，第5:1–5:7页，doi:10.1145/2957319.2957372.[27] Roberto Casadei、Mirko Viroli、Gianluca Aguzzi & Danilo Pianini (2022)：ScaFi：用于聚合编程的 Scala 领域专用语言与工具包。《SoftwareX》20，页码101248，doi:10.1016/j.softx.2022.101248.[28] Luis Damas & Robin Milner (1982)：函数式程序的主类型方案。载于：编程语言原理研讨会，POPL '82，ACM，第207–212页，doi:10.1145/582153.582176.[29] Ferruccio Damiani、Mirko Viroli & Jacob Beal (2016)：计算场的类型安全演算。《计算机程序科学》117，第17–44页，doi:10.1016/j.scico.2015.11.005.[30] Soura Dasgupta & Jacob Beal (2016)：自适应 Bellman-Ford 算法鲁棒稳定性的李雅普诺夫分析。载于：决策与控制（CDC），2016年 IEEE 第55届会议，IEEE，第7282–7287页，doi:10.1109/CDC.2016.7799393.[31] Simon Dobson、Spyros Denazis、Antonio Fernández、Dominique Gaïti、Erol Gelenbe、Fabio Massacci、Paddy Nixon、Fabrice Saffre、Nikita Schmidt & Franco Zambonelli (2006)：自治通信综述。TAAS 1(2)，第223–259页，doi:10.1145/1186778.1186782.[32] Ramakrishna Gummadi、Omprakash Gnawali & Ramesh Govindan (2005)：使用 Kairos 对无线传感器网络进行宏编程。载于：传感器系统中的分布式计算（DCOSS），第126–140页，doi:10.1007/11502593 12._[33] Atsushi Igarashi、Benjamin C. Pierce & Philip Wadler (2001)：轻量级 Java：一种面向 Java 和 GJ 的最小核心演算。《ACM程序设计语言与系统汇刊》23(3)，doi:10.1145/503502.503505.[34] Serge Kernbach、Thomas Schmickl & Jon Timmis (2011)：集体自适应系统：可进化性之外的挑战。CoRR abs/1108.5643。arXiv:1108.5643.[35] Leslie Lamport (1978)：分布式系统中的时间、时钟与事件排序。《ACM通讯》21(7)，第558–565页，doi:10.1145/359545.359563.[36] Alberto Lluch-Lafuente、Michele Loreti & Ugo Montanari (2017)：基于不动点的计算场的异步分布式执行。《逻辑方法与计算机科学》13(1)，doi:10.23638/LMCS- 13(1:13)2017.[37] Michele Loreti & Jane Hillston (2016)：使用 CARMA 及其工具对集体自适应系统进行建模与分析。载于：集体自适应系统定量评估的形式化方法——第16届计算机、通信与软件系统设计形式化方法国际暑期学校，SFM 2016，高级讲座，LNCS 9700，施普林格，第83–119页，doi:10.1007/978-3-319-34096-8_4.[38] Marco Mamei、Ronaldo Menezes、Robert Tolksdorf & Franco Zambonelli (2006)：计算机科学中的自组织案例研究。《系统架构杂志》52(8-9)，第443–460页，doi:10.1016/j.sysarc.2006.02.002.[39] Marco Mamei & Franco Zambonelli (2004)：使用 TOTA 中间件编程普适与移动计算应用。载于：普适计算与通信，2004年，IEEE，第263–273页，doi:10.1109/PERCOM.2004.1276864.[40] Marco Mamei、Franco Zambonelli & Letizia Leonardi (2002)：Co-Fields：迈向群体智能系统工程的统一方法。载于：第3届智能体世界中的工程化社会国际研讨会，LNCS 2577，施普林格，第68–81页，doi:10.1007/3-540-39173-8_6.[41] Ryan Newton & Matt Welsh (2004)：区域流：面向传感器网络的函数式宏编程。载于：传感器网络数据管理研讨会，第78–87页，doi:10.1145/1052199.1052213.[42] Rocco De Nicola、Tan Duong、Omar Inverso & Catia Trubiani (2018)：AErlang：通过基于属性的通信增强 Erlang。《计算机科学与程序设计》168，第71–93页，doi:10.1016/j.scico.2018.08.006。

[43] Rocco De Nicola、Stefan Jähnichen & Martin Wirsing (2020)：集体自适应系统的严谨工程：专题栏目。《国际软件工具与技术转移杂志》22(4)，第389–397页，doi:10.1007/s10009-020- 00565-0. [44] Rocco De Nicola、Michele Loreti、Rosario Pugliese & Francesco Tiezzi (2014)：自治系统编程的形式化方法：*SCEL*语言。《*ACM*自治与自适应系统汇刊》9(2)，第7:1–7:29页，doi:10.1145/2619998.[45] L Nitti、D Bortoluzzi、A Biglia、D Ricauda Aimonino、G Torta、G Audrito、F Damiani、P Gay、A Rapp & L Comba (2025)：面向自主机器安全功能的无人机群监控系统。载于：精准农业*'25*，Wageningen Academic，第592–598页，doi:10.1163/9789004725232_077. [46] Carlo Pinciroli & Giovanni Beltrame (2016)：*Buzz*：一种面向机器人群体的编程语言。《*IEEE* 软件》33(4)，第97–100页，doi:10.1109/MS.2016.95.[47] Lorenzo Testa、Giorgio Audrito、Ferruccio Damiani & Gianluca Torta (2022)：作为工业物联网分布式自适应服务的聚合进程。《普适与移动计算》85，页码101658，doi:10.1016/j.pmcj.2022.101658.[48] Mirko Viroli、Giorgio Audrito、Jacob Beal、Ferruccio Damiani & Danilo Pianini (2018)：通过自稳定构建具有韧性的集体自适应系统。《*ACM*建模与计算机仿真汇刊》28(2)，第16:1–16:28页，doi:10.1145/3177774.[49] Mirko Viroli、Jacob Beal、Ferruccio Damiani、Giorgio Audrito、Roberto Casadei & Danilo Pianini (2019)：从分布式协调到场演算与聚合计算。《逻辑与代数方法程序设计杂志》109，doi:10.1016/j.jlamp.2019.100486.[50] Hiroshi Wadaa、Pruet Boonmab & Junichi Suzukic (2010)：*Chronus*：一种面向自治无线传感器网络的时空宏编程语言。167，学术出版社，doi:10.1016/B978-0-12-382190- 4.00008-5.[51] Lei Wang、Chen Ma、Xueyang Feng、Zeyu Zhang、Hao Yang、Jingsen Zhang、Zhiyuan Chen、Jiakai Tang、Xu Chen、Yankai Lin 等 (2024)：基于大语言模型的自主智能体综述。《计算机科学前沿》18(6)，页码186345，doi:10.1007/s11704-024-40231-1.[52] Martin Wirsing、Matthias M. Hölzl、Mirco Tribastone & Franco Zambonelli (2011)：*ASCENS*：自治服务组件群的工程化。载于：组件与对象的形式化方法，第*10*届国际研讨会，*FMCO 2011*，修订版精选论文，*LNCS*7542，施普林格，第1–24页，doi:10.1007/978- 3-642-35887-6 1.