arXiv:2303.03170v2 [cs.PL] 3 Jul 2023

异步模态函数式响应式编程

PATRICK BAHR，哥本哈根信息技术大学，丹麦RASMUS EJLERS MØGELBERG，哥本哈根信息技术大学，丹麦

过去十年间，人们提出了多种用于函数式响应式编程（FRP）的语言，它们利用模态类型来确保因果性、生产性和不会发生空间泄漏等性质。迄今为止，这些语言几乎都包含一个基于全局时钟的延迟模态算子。然而，对于某些应用而言，全局时钟并不自然，会导致泄漏抽象以及低效的实现。虽然已有不使用全局时钟的模态语言被提出，但迄今尚未证明其任何操作性质。

本文提出 Async RaTT，这是一种用于异步 FRP 的新型模态语言，并配备操作语义，可将完整程序映射为机器；这些机器接收异步输入信号并生成输出信号。Async RaTT 的主要创新在于引入了一种用于异步延迟的新模态，使每个输出通道都能在运行时关联到其所依赖的输入通道集合，从而使机器仅在必要时才计算新的输出。我们证明了一系列操作性质，包括因果性、生产性和不会发生空间泄漏。我们还表明，尽管与某个输出通道关联的输入通道集合会在执行期间发生变化，但类型系统可以静态确定这些集合大小的上界。

CCS 概念： • **软件及其工程** → **函数式语言**； **数据流语言**； 递归； • **计算理论** → 操作语义。

附加关键词和短语：函数式响应式编程、模态类型、线性时序逻辑、同步数据流语言、类型系统

1 引言

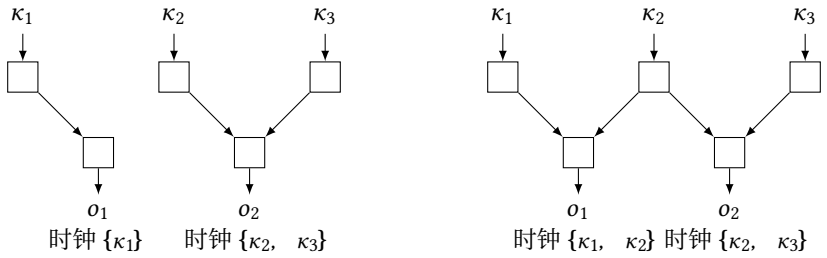
响应式程序是与其环境进行交互的程序：接收输入并产生输出，而且通常永不终止。如今使用的许多安全关键型软件都属于此类，例如控制软件和服务器，以及图形用户界面。大多数响应式软件都是用命令式语言编写的，并结合使用回调和共享内存等复杂特性，因此容易出错且难以推理。

函数式响应式编程（FRP）的理念 [12] 是为程序员提供恰当的抽象，使其能够以函数式风格编写响应式程序，从而实现简短的模块化程序，并对其进行模块化推理。要使这类抽象有用，重要的是将其设计为能够根据程序自动生成高效底层实现。

函数式响应式编程的主要抽象是信号，即与时间相关的值。在由全局时钟定义的离散时间情形下，信号可以看作数据流。响应式程序本质上只是一个接收输入信号并生成输出信号的函数。然而，要使其可实现，它必须是因果的：当前输出只能依赖于当前及过去的输入。此外，由高级程序生成的底层实现也应避免（隐式的）空间和时间泄漏。这意味着响应式程序不应无限期地存储数据，以致最终耗尽空间；也不应以某种方式重复计算，导致每一步的执行都越来越慢。

这些要求促成了模态函数式响应式编程 [2, 3, 16–21], 的发展，这是一类使用模态类型来确保所有程序都能高效实现的语言。最重要的模态类型构造子是 ○，用于对全局离散时钟上下一时间步可用的数据进行分类。例如，信号的类型应满足以下类型同构

作者地址：Patrick Bahr，哥本哈根信息技术大学，丹麦，paba@itu.dk；Rasmus EjlersMøgelberg，哥本哈根信息技术大学，丹麦，mogel@itu.dk。



(a) 带有计算所得时钟的数据流图。 (b) 随着依赖项发生变化，时钟也会更新。

图 1. 输入通道为 κ_1、κ_2、κ_3 且输出通道为 o_1、o_2的 Async RaTT 程序的动态变化数据流图。

Sig $A \cong A \times \bigcirc(\text{Sig } A)$，该同构表明，信号的当前值现在即可获得，而未来值只有在下一个时间步之后才能获得。采用这种信号编码，可以确保所有响应式程序都具有因果性。许多模态 FRP 语言还包含 Nakano[22] 守卫不动点算子，其类型为 $(\bigcirc A \to A) \to A$。该类型确保递归调用只会在未来的步骤中执行，从而确保每一步计算都能终止；这一性质称为 生产性。这些语言通常还包含一种 $\square$ 模态，用于对稳定的数据进行分类，也就是说，这类数据可以保留到下一个时间步，而不会造成空间泄漏。其他模态构造子，例如 $\Diamond$ （最终）也可以编码，这表明线性时序逻辑 [25] 与模态 FRP [4, 9, 16, 18]之间存在 Curry-Howard 对应。

然而，对于许多应用而言，与 $\bigcirc$ 模态运算符相关联的全局时钟这一概念可能并不自然，也可能导致实现效率低下。例如，考虑一个图形用户界面，它接收用户按键这一输入信号，以及其他更新频率更高的信号，例如鼠标指针坐标。全局时钟的滴答频率至少必须与更新最快的信号一样快，而按键信号只会在全局时钟的少数几个时钟滴答中更新。因此，对按键信号进行建模，最自然的方式或许是使用类型为 Maybe(Char) 的信号。在 Bahr 等人 [3], Krish- naswami [19], 提出的模态 FRP 语言中，该信号的处理器必须在全局时钟的每个时钟滴答时唤醒、检查是否有输入，并且通常还要通过递归调用自身，将某些局部状态传递到下一个时间步。然而，更成问题的或许是：当输入信号的处理器能够访问全局时钟时，重要的抽象屏障就被打破了。相反，我们希望将图形用户界面编写为一组异步输入信号处理器，它们仅在所依赖的信号发生更新时才被激活。

1.1 Async RaTT

本文介绍 Async RaTT，这是一种 RaTT 系列中的模态函数式响应式编程语言 [2–4], , 旨在处理异步输入。Async RaTT 中的响应式程序从一组输入通道 读取信号，并据此向一组输出通道发送信号。在图形用户界面应用程序中，典型的输入通道包括鼠标位置和按键事件，而输出通道则可以包括文本字段的内容或颜色。

对于每个输出通道 o，响应式程序都会跟踪输入通道集合 θ，这些通道是 o 所依赖的（参见图 1a）。我们将这样的输入通道集合 θ 称为一个时钟。当输入通道 κ 上的信号更新时，只有时钟 θ 包含 κ 的那些输出通道才会更新。例如，按键输入通道可能属于文本字段内容的时钟，但不属于文本字段颜色的时钟。由于程序可以动态更改其内部数据流图，

与输出通道关联的时钟可能会在执行过程中发生变化（参见图 1b），因此在编译时无法确定。例如，文本字段可能会失去焦点，从而不再响应按键。我们将时钟 θ 上输入通道有新数据到达称为一次时钟滴答，发生在时钟 θ 上。

Async RaTT 含有一个模态算子 $\square$，用于对稳定数据进行分类，并引入了两种新模态：$\Circled{\exists}$，用于表示异步延迟；以及 $\Circled{\forall}$，用于表示全局时钟上的延迟。类型为 $\Circled{\exists}A$ 的值是一个二元组，由一个时钟 θ 和一个计算组成；该计算可在 A 上的第一个时钟滴答时执行，并返回类型为 θ 的数据。因此，类型 $\Circled{\exists}A$ 可以被视为存在类型。如图 1所示，输出通道的时钟存储在这个存在类型的第一个分量中。我们的信号概念在类型中编码为递归类型信号 $A \cong A \times \Circled{\exists}$信号 A。这意味着，与信号尾部关联的时钟可能会随着每一步而变化，从而允许动态更新与输出通道关联的时钟，如图 1所示。

与同步的 $\bigcirc$ 不同，异步的 $\Circled{\exists}$ 没有类型为$\Circled{\exists}(A \to B) \to \Circled{\exists}A \to \Circled{\exists}B$ 的应用式作用，因为延迟函数和延迟输入可能不会同时到达；而且为了避免空间泄漏，Async RaTT 不允许在第二个输入到达之前存储第一个输入。相反，Async RaTT使用一种算子来同步延迟数据

$$\mathsf{sync} : \Circled{\exists}A_1 \to \Circled{\exists}A_2 \to \Circled{\exists}((A_1 \times \Circled{\exists}A_2) + (\Circled{\exists}A_1 \times A_2) + (A_1 \times A_2))$$

给定分别与时钟 θ_1 和 θ_2关联的两个延迟计算，sync 返回与并时钟 $\theta_1 \sqcup \theta_2$关联的延迟计算。该延迟计算会等待任意输入通道 $\kappa \in \theta_1 \sqcup \theta_2$上的输入，然后根据 $\kappa \in \theta_1$、 $\kappa \in \theta_2$或两者的情况，计算能够被计算的计算。例如，如果输入到达通道 $\kappa \in \theta_1 \setminus \theta_2$，则只计算第一个延迟计算。sync 算子可用于实现诸如以下算子

$$\mathsf{switch} : \mathrm{Sig}\, A \to \Circled{\exists}(\mathrm{Sig}\, A) \to \mathrm{Sig}\, A$$

它们会动态更新程序的数据流图。

请注意，sync 可以理解为线性时间公理：给定两个时钟，要么一个先于另一个滴答，要么二者同时滴答。因此，Async RaTT 程序依赖运行时环境来安排处理输入的顺序。所谓异步，是指输出通道会异步更新。这一点在类型系统中体现为，如上所述， $\Circled{\exists}$不是应用函子。

模态类型 $\Circled{\forall}A$ 对可在未来任意时刻运行、但不能现在运行的计算进行分类。它用于守卫不动点算子，该算子在 Async RaTT 中的类型为

$$\square(\Circled{\forall}A \to A) \to A$$

不动点算子的输入必须是稳定函数（由 $\square$分类），因为它将在未来任意时刻用于展开。使用 $\Circled{\forall}$ 将不动点限制为只能在未来展开，从而确保每个计算步骤都会终止。

1.2 操作语义与结果

我们给出一种操作语义，它将每个完整的 Async RaTT 程序映射为一台机器，将输入通道上接收到的输入序列转换为输出通道上的输出序列。该转换分步进行，每次处理一个输入，并在受影响的输出通道上产生新的输出。

操作语义由两部分组成。第一部分是求值语义，它描述每一步中项的求值过程。在当前输入信号值的上下文中，它接收一个项和一个存储，并返回一个值和更新后的存储。存储包含

位置 l $\in \mathsf{Loc}$ 输入通道 κ $\in \mathsf{Chan}$ 时钟表达式 $\theta ::= \mathsf{cl}\ (v) \mid \theta \sqcup \theta'$类型 $A, B ::= \alpha \mid 1 \mid$自然数 $\mid A \times B \mid A + B \mid A \to B \mid \exists A \mid \forall A \mid$不动点 $\alpha.A \mid \Box A$稳定类型 $S, S' ::= 1 \mid$自然数 $\mid S \times S' \mid S + S' \mid \forall A \mid \Box A$值类型 $T, T' ::= 1 \mid$自然数 $\mid T \times T' \mid T + T'$ 值 $v, w ::= x \mid \mathsf{hi} \mid 0 \mid$后继 $v \mid \lambda x.t \mid (v, w) \mid \mathsf{in}_i\, v \mid l \mid$等待$_\kappa \mid \mathsf{box}\ t \mid \mathsf{dfix}\ x.t \mid \mathsf{into}\ v$项 $s, t ::= v \mid$后继 $t \mid \mathsf{rec}_{\mathsf{Nat}}(s, x\, y.t, u) \mid (s, t) \mid \mathsf{in}_i\, t \mid \pi_i\, t \mid t_1 t_2 \mid \mathsf{let}\ x = s \mathsf{in}\ t \mid$情形 $t \mathsf{of} \mathsf{in}_1\, x.t_1; \mathsf{in}_2\, x.t_2 \mid$延迟$_\theta\, t \mid \mathsf{adv}\ t \mid \mathsf{select}\ v_1\, v_2 \mid \mathsf{unbox}\ t \mid$不动点 $x.t \mid$永不 $\mid \mathsf{into}\ t \mid \mathsf{out}\ t \mid$读取$_\kappa$

图 2. 语法。

延迟计算，而求值语义既可以运行先前存储的延迟计算，也可以存储新的延迟计算，以便在后续步骤中求值。另一方面，反应式语义描述的是这样一台机器：它在每个步骤中定位需要更新的输出信号，并执行相应的延迟计算以产生输出。

操作语义所描述的从输入到输出的转换从构造上就是因果的。我们证明，它也是确定性的且具有产出性（即每个步骤都会终止，且永不陷入停滞）。我们还证明，Async RaTT 程序的执行不会产生（隐式）空间泄漏。这是通过采用 Krishnaswami 最初提出的一种技术实现的[19]：在每个执行步骤结束时，机器会删除所有原则上可能在当前步骤中运行的延迟计算——无论它们实际上是否运行过。所有输入也会被删除，具体是在步骤结束时删除，还是等到同一信号的下一个输入到达时删除，取决于该输入信号的类型。我们的结果表明，这种激进的垃圾回收策略是安全的。当然，程序员仍然可以编写会累积空间的程序，但这类泄漏会明确地出现在源程序中，而不是由语言实现隐式引入。（关于隐式与显式空间泄漏的进一步讨论，请参见 Krishnaswami [19] 。）

最后，我们证明，与输出信号相关联的动态时钟的上界可以静态计算。更准确地说，给定一个由若干输出信号组成、且处于输入通道上下文 Δ 中的 Async RaTT 程序，如果某个输出信号能够在更小的上下文 $\Delta' \subseteq \Delta$ 中进行类型检查，那么该信号永远不需要因通道 $\Delta \backslash \Delta'$ 上到达的输入而更新。值得注意的是，尽管存在 switch 这类会动态改变程序数据流图的运算符，这一结论依然成立。

1.3 概述

本文结构安排如下：第2节介绍 Async RaTT 及其类型规则；第 3 节通过开发一个小型信号组合子库，展示 Async RaTT 的表达能力，并给出使用该库进行图形用户界面编程以及计算信号积分和导数的示例。第4节定义操作语义，并通过示例加以说明，同时介绍主要结果。第5 节概述主要结果的证明，尤其定义了证明中使用的 Kripke 逻辑关系。最后，第 6节和第 7 节讨论相关工作、结论和未来工作。此外，附录A 详细介绍了 Kripke 逻辑关系基本性质的证明。

$$\frac{}{\cdot \vdash_\Delta} \qquad \frac{\Gamma \vdash_\Delta \qquad x \notin \mathrm{dom}(\Gamma)}{\Gamma,\ x :\ A \vdash_\Delta} \qquad \frac{\Gamma \vdash_\Delta \qquad \Gamma \vdash_\Delta \theta :\ \text{时钟} \qquad \Gamma \text{无tick}}{\Gamma,\ \checkmark_\theta \vdash_\Delta}$$

$$\frac{\Gamma \vdash_\Delta \theta :\ \text{时钟} \qquad \Gamma \vdash_\Delta \theta' :\ \text{时钟}}{\Gamma \vdash_\Delta \theta \sqcup \theta' :\ \text{时钟}} \qquad \frac{\Gamma \vdash_\Delta v :\ \Diamond A}{\Gamma \vdash_\Delta \text{闭包}(v) :\ \text{时钟}}$$

$$\frac{\Gamma' \text{无tick或} A \text{稳定} \qquad \Gamma, x : A, \Gamma' \vdash_\Delta}{\Gamma,\ x :\ A,\ \Gamma' \vdash_\Delta x :\ A} \qquad \frac{}{\Gamma \vdash_\Delta \text{嗨} :\ 1} \qquad \frac{\Gamma \vdash_\Delta s :\ A \qquad \Gamma,\ x :\ A \vdash_\Delta t :\ B}{\Gamma \vdash_\Delta \text{令} x = s,\ \text{其中} t :\ B}$$

$$\frac{\Gamma,\ x :\ A \vdash_\Delta t :\ B \qquad \Gamma \text{无tick}}{\Gamma \vdash_\Delta \lambda x.t :\ A \to B} \qquad \frac{\Gamma \vdash_\Delta t :\ A \to B \qquad \Gamma \vdash_\Delta t' :\ A}{\Gamma \vdash_\Delta t\, t' :\ B} \qquad \frac{\Gamma \vdash_\Delta t : A_i \qquad i \in \{1,\ 2\}}{\Gamma \vdash_\Delta \text{在}_i\, t :\ A_1 + A_2}$$

$$\frac{\Gamma,\ x :\ A_i \vdash_\Delta t_i :\ B \qquad \Gamma \vdash_\Delta t :\ A_1 + A_2 \qquad i \in \{1,2\}}{\Gamma \vdash_\Delta \text{情形 } t \text{的 } \mathrm{in}_1\, x.t_1; \mathrm{in}_2\, x.t_2 :\ B} \qquad \frac{\Gamma \vdash_\Delta t : A \qquad \Gamma \vdash_\Delta t' : B}{\Gamma \vdash_\Delta (t,\ t') :\ A \times B}$$

$$\frac{\Gamma \vdash_\Delta t :\ A_1 \times A_2 \qquad i \in \{1,\ 2\}}{\Gamma \vdash_\Delta \pi_i\, t :\ A_i} \qquad \frac{\Gamma,\ \checkmark_\theta \vdash_\Delta t :\ A \qquad \Gamma \vdash_\Delta \theta :\ \text{时钟}}{\Gamma \vdash_\Delta \text{延迟}_\theta\, t :\ \Diamond A} \qquad \frac{}{\Gamma \vdash_\Delta \text{永不} :\ \Diamond A}$$

$$\frac{\kappa :\ {}_c A \in \Delta \qquad c \in \{\mathrm{p},\ \mathrm{bp}\}}{\Gamma \vdash_\Delta \text{等待}_\kappa :\ \Diamond A} \qquad \frac{\kappa :\ {}_c A \in \Delta \qquad c \in \{\mathrm{b},\ \mathrm{bp}\}}{\Gamma \vdash_\Delta \text{读取}_\kappa :\ A} \qquad \frac{\Gamma \vdash_\Delta v :\ \Diamond A \qquad \Gamma,\ \checkmark_{\mathrm{cl}(v)},\ \Gamma' \vdash_\Delta}{\Gamma,\ \checkmark_{\mathrm{cl}(v)},\ \Gamma' \vdash_\Delta \mathrm{adv}\, v :\ A}$$

$$\frac{\Gamma \vdash_\Delta v_1 :\ \Diamond A_1 \qquad \Gamma \vdash_\Delta v_2 :\ \Diamond A_2 \qquad \vdash \theta_1 \sqcup \theta_2 = \text{闭包}(v_1) \sqcup \text{闭包}(v_2) \qquad \Gamma,\ \checkmark_{\theta_1 \sqcup \theta_2},\ \Gamma' \vdash_\Delta}{\Gamma, \checkmark_{\theta_1 \sqcup \theta_2}, \Gamma' \vdash_\Delta \mathrm{select}\, v_1\, v_2 : ((A_1 \times \Diamond A_2)\ +(\ \Diamond A_1 \times A_2))\ +(A_1 \times A_2)}$$

$$\frac{}{\Gamma \vdash_\Delta 0 : \text{自然数}} \qquad \frac{\Gamma \vdash_\Delta t : \text{自然数}}{\Gamma \vdash_\Delta \text{后继}\, t : \text{自然数}} \qquad \frac{\Gamma \vdash_\Delta s : A \qquad \Gamma,\ x : \text{自然数}\ y :\ A \vdash_\Delta t :\ A \qquad \Gamma \vdash_\Delta n : \text{自然数}}{\Gamma \vdash_\Delta \text{递归}_{\mathrm{Nat}}(s,\ x\, y.t,\ n) :\ A}$$

$$\frac{\Gamma^\square, x :\ \Box A \vdash_\Delta t :\ A}{\Gamma \vdash_\Delta \text{不动点}\ x.t :\ A} \qquad \frac{\Gamma \vdash_\Delta x :\ \Box A \qquad \Gamma, \checkmark_\theta, \Gamma' \vdash_\Delta}{\Gamma, \checkmark_\theta, \Gamma' \vdash_\Delta \mathrm{adv}\, x :\ A} \qquad \frac{\Gamma^\square \vdash_\Delta t :\ A}{\Gamma \vdash_\Delta \text{装箱}\, t :\ \Box A}$$

$$\frac{\Gamma \vdash_\Delta t :\ \Box A}{\Gamma \vdash_\Delta \text{拆箱}\, t :\ A} \qquad \frac{\Gamma \vdash_\Delta t :\ \text{不动点}\ \alpha.A}{\Gamma \vdash_\Delta \text{输出}\, t :\ A[\Diamond(\text{不动点}\ \alpha.A)/\alpha]} \qquad \frac{\Gamma \vdash_\Delta t :\ A[\Diamond(\text{不动点}\ \alpha.A)/\alpha]}{\Gamma \vdash_\Delta \text{进入}\, t :\ \text{不动点}\ \alpha.A}$$

$$\cdot^\square = \cdot \qquad (\Gamma,\ \checkmark_\theta)^\square = \Gamma^\square \qquad (\Gamma,\ x :\ A)^\square = \begin{cases} (\Gamma^\square,\ x :\ A) & \text{如果 } A \text{ 稳定} \\ \Gamma^\square & \text{否则} \end{cases}$$

图3. 类型规则。

2 ASYNC RATT

本节概述 Async RaTT，并参见图2 和3，其中完整给出了其语法和类型规则。

Async RaTT 程序可以访问一组输入通道，每个输入通道都会异步接收彼此独立的更新。为此，类型判断相对于输入通道上下文 Δ 或简写为输入上下文而言。此类上下文的一个示例如下：

$$\mathsf{keyPressed} :_{\mathsf{p}} \mathsf{Nat}, \mathsf{mouseCoord} :_{\mathsf{bp}} \mathsf{Nat} \times \mathsf{Nat}, \mathsf{time} :_{\mathsf{b}} \mathsf{Float}$$

输入通道分为三类，分别对应于上例中的下标 p、b 和 bp。仅推送输入通道用 p 表示，其更新会通过程序推送，可能导致输出通道更新。在上述示例上下文中，程序员希望立即响应用户的按键事件，因此应推送对该通道的更新。另一方面，我们可能希望访问一个时间输入通道，可以随时从中读取，但又不希望每当时间发生变化时程序都被唤醒。因此，时间被视为仅缓冲输入通道，用 b 表示；其最新值会被缓冲，但其变化不会触发程序更新任何输出通道。最后，输入通道也可以同时进行缓冲并推送，用 bp 表示。这意味着更新会被推送，同时我们还会将值保存在缓冲区中，以便程序始终能够读取最新值。这不同于仅推送输入通道：出于节省空间的原因，一旦某次更新推送得到处理，其值就会被删除。例如，我们可能希望在鼠标坐标更新时收到通知，同时保留这些坐标，以便在按键事件发生时读取鼠标坐标，即使鼠标并未移动。我们将仅推送或缓冲并推送（p 或 bp）的输入通道称为**推送通道**，类似地，将仅缓冲或缓冲并推送的输入通道称为**缓冲通道**。

我们假设所有信号都具有值类型，即任何声明 $\kappa :_c A$ 位于 Δ 都必须具有值类型 A。值类型的语法见图 2。

2.1 时钟与 $\Im$

直观地说，时钟是一组程序可能需要对此作出反应的推送通道（p 或 bp）。例如，$\emptyset$、{keyPressed} 以及 {keyPressed、mouseCoord} 都是前文所述示例输入上下文中的时钟。类型 $\Im A$ 是在一个存在量化时钟上的延迟计算类型。换言之，类型为 $\Im A$ 的值是一个时钟 θ 和一个计算组成的对；当 θ 中某个输入通道收到更新后，该计算将产生一个类型为 A 的值。我们将时钟 θ 上的此类更新称为一次**时钟滴答**。例如，如果关联的时钟是 {keyPressed、mouseCoord}，那么类型为 A 的数据可以在 keyPressed 或 mouseCoord 接收到新输入后计算得到。

由于 $\Im A$ 是存在类型，因此可以为这些类型的任意值 获取时钟 $\mathsf{cl}(v)$。类型为 $\Im A$ 的值是变量和 wait_κ，其中 κ 是推送通道之一。后者充当对推送到 κ上的下一个值的引用，因此直观地说，$\mathsf{cl}\,(\mathsf{wait}_\kappa) = \{\kappa\}$。时钟也可以使用并运算符 $\sqcup$组合。我们还引入了一个元素 never，它与空时钟相关联。

我们采用 Fitch 风格 [10],，而不是较为传统的双重上下文风格 [11]，来使用 Async RaTT 的模态类型构造器进行编程。对于 $\Im$，这意味着引入规则和消去规则会在上下文中使用一个特殊符号 $\checkmark_\theta$，称为时钟滴答。可以将时钟滴答 $\checkmark_\theta$ 理解为表示时钟 θ 的滴答，它将判断分为两部分：在时钟滴答之前收到的变量（位于 $\checkmark_\theta$ 左侧），以及其余所有内容，这些都发生在时钟滴答之后。例如，消去规则应读作：如果 v 现在具有类型 $\Im A$，那么在时钟 $\mathsf{cl}\,(v)$ 上经过一次时钟滴答后，$\mathsf{adv}(v)$ 具有类型 A。类似地，$\Im$ 的引入规则应读作：如果 t 在时钟 θ 上经过一次时钟滴答后具有类型 A，那么 $\mathsf{delay}_\theta\, t$ 现在具有类型 $\Im A$。请注意，一个上下文中至多只能有一个时钟滴答。这一限制是证明生产性定理（定理 4.1）所必需的，而且

也出现在 RaTT 系列的其他语言中 [3, 4]。不过，Bahr [2] 证明了，可以通过程序变换解除这一限制：该变换将一个可在含有多个时钟滴答的上下文中定型的程序，转换为一个仅含有一个时钟滴答的程序，并且其中 adv 只应用于变量。

在操作层面上，项 $\mathsf{delay}_\theta\, t$ 会创建一个延迟计算，并将其存储在堆中，直到可用于对其求值的输入数据就绪。因此，它不被视为值，而是$\mathsf{delay}_\theta\, t$ 求值得到一个指向该延迟计算的堆引用 l。尽管堆引用是 Async RaTT 的组成部分，甚至也被视为值（图 2），但程序员不能直接使用它们，因此也没有针对它们的类型规则。

在并时钟 $\mathsf{cl}\,(v_1) \sqcup \mathsf{cl}\,(v_2)$ 上收到一个时钟滴答后，可以使用 select 同步两个延迟值 $v_1 : \circledcirc A_1$ 和 $v_2 : \circledcirc A_2$。$\mathsf{select}\, v_1\, v_2$ 的类型反映了该时钟滴答可能出现的三种情形：它可能属于一个 $\mathsf{cl}\,(v_i)$，但不属于另一个；也可能同时属于两者。例如，如果输入属于 $\mathsf{cl}\,(v_1)$，但不属于 $\mathsf{cl}\,(v_2)$，那么就可以计算出类型为 $A_1 \times \circledcirc A_2$ 的数据。第 1.1 节中展示的同步算子可以用 select 定义。使用类似 select 的项来区分这些情形的想法源自 Graulund 等人 [14], , 他们只要求定义两种情形，并在时钟滴答位于时钟交集中的情形下采用非确定性选择。在 Async RaTT 中，提供全部三种情形对于第 4 节的操作性结果至关重要。

请注意，select 和 adv 的规则将这些构造的应用限制为值。这样做的原因之一是，它可以通过防止任意项经由时钟出现在上下文中来简化元理论。这也意味着时钟表达式始终是无需求值的值。例如，对 $\mathsf{delay}^{\mathsf{cl}\,(t)}(\mathsf{adv}(t))$ 求值需要将 t 求值两次：先计算时钟，再计算项。结合使用 let 绑定和 adv，可以对更一般的项 t 进行 $\circledcirc$ 的消除。

2.2 稳定类型与不动点

Async RaTT 中的一般值可以包含对依赖时间的数据的引用，例如存储在堆中的延迟计算。类型系统的主要目的之一，是防止在未来某个时刻解引用此类引用，而此时延迟计算可能已从堆中删除。因此，不应让任意数据跨越时间步保留，这一点体现在变量引入规则中：该规则禁止在时钟滴答之间引入一般变量。

然而，对于某些类型，值不能包含此类引用。我们将这些类型称为稳定类型，其语法见图 2。稳定类型包括所有形如 $\Box A$ 的类型，它们用于分类生成类型为 A 的值且不访问延迟计算的计算。$\Box$ 的引入规则会构造一个延迟计算框 $\mathsf{box}(t)$，该计算可在未来任意时刻求值。这要求 t 必须在稳定上下文中进行类型判断，因此，类型规则的假设会从上下文中移除所有时钟滴答以及所有非稳定类型的变量。$\Box$ 模态具有余单位和余乘法 $\Box \to \Box\Box$。请注意，wait_κ 和 read_κ 是稳定的，因为对于任意 $\kappa :_c A \in \Delta$，都有 $\vdash_\Delta \mathsf{box}(\mathsf{wait}_\kappa) : \Box(\circledcirc A)$，其中 $c \in \{\mathsf{p}, \mathsf{bp}\}$；对于任意 $\kappa :_c A \in \Delta$，都有 $\vdash_\Delta \mathsf{box}(\mathsf{read}_\kappa) : \Box A$，其中 $c \in \{\mathsf{b}, \mathsf{bp}\}$。

Async RaTT 是一种终止演算，即每一步计算都会终止。不过，它仍然允许通过不动点算子进行递归定义，其类型可确保递归调用只会在之后的时间步进行。更准确地说，$\mathsf{fix}\ x.t$ 中的递归变量 x 具有类型 $\textcircled{\scriptsize\forall} A$，这意味着递归定义可以在未来任意时刻展开，生成一个类型为 A 的项，但不能在现在展开。这是通过 $\textcircled{\scriptsize\forall}$ 的消去规则来保证的；该规则允许使用当前上下文中可定型的任意时钟上的时钟滴答来推进它。由于不动点可以在未来任意时刻被递归调用，因此它们必须是稳定的，所以 t 必须能在稳定上下文中定型。

类型 $\mathsf{Fix}\,\alpha.A$ 是受保护递归类型，可通过项 into 和 out 展开为 $A[\circledcirc(\mathsf{Fix}\ \alpha.A)/\alpha]$。其中最重要的类型是 $\mathsf{Sig}\ A$，其定义为 $\mathsf{Fix}\,\alpha.(A \times \alpha)$，展开后为 $A \times \circledcirc(\mathsf{Sig}\,A)$。信号由当前值和一个延迟的尾部组成；在未来某个时刻，该尾部可能会返回一个新信号。任何推送通道 $\kappa :_c A \in \Delta$，其中 $c \in \{\mathrm{p}, \mathrm{bp}\}$，都会导出一个稳定信号：

$$\mathrm{box}\left(\mathrm{fix}\,x.\mathrm{delay}_{\mathrm{cl}(\mathrm{wait}_\kappa)}\ (\mathrm{into}\ (\mathrm{adv}(\mathrm{wait}_\kappa), \mathrm{adv}(x)))\right) : \square(\circledcirc(\mathsf{Sig}\,A))$$

其中递归变量 x 的类型为 $\textcircled{\scriptsize V}(\circledcirc A)$。当然，这些信号都作用于固定时钟 $\{\kappa\}$，但一般来说，与信号尾部关联的时钟可能在每一步发生变化，我们将在第 3 节中看到相关示例。

除了上述所有构造之外，Async RaTT 还包含函数式编程中的许多标准构造：和类型、积类型、自然数和函数类型。这些类型的规则完全遵循标准形式，唯一的例外是，函数类型只能在不含时钟滴答的上下文中构造。RaTT 系列中的其他演算系统也存在类似限制 [3, 4], ，而且这是得到第 4 节结果所必需的。Bahr 提到的程序变换 [2] 也会移除此限制。请注意，函数类型并不稳定，因为与时间相关的引用可以存储在闭包中。

3 Async RaTT 编程

本节通过若干示例展示 Async RaTT 的表达能力。为此，我们假设有一种表层语言，它通过模式匹配、递归和顶层定义的语法糖扩展了 Async RaTT。这些内容可以轻松地 elaboration 为 Async RaTT 演算，具体如第 3.5 节所述。

3.1 简单信号组合子

我们首先实现一组简单的组合子，用于操作信号，即受保护递归类型 $\mathsf{Sig}\ A$ 定义为 $\mathsf{Fix}\,\alpha.(A \times \alpha)$ 的元素。为便于阅读，我们用简写 $s :: t$ 表示 into (s, t)；因此，给定 $s : A$ 和 $t : \circledcirc(\mathsf{Sig}\ A)$，有 $s :: t : \mathsf{Sig}\ A$。

我们从或许最简单的信号组合子开始：

映射 : $\square\,(A \to B) \to$ 信号 $A \to$ 信号 B 映射 f
$(x :: xs) =$ 拆箱 $f\,x$ ::延迟 (映射 f (adv xs))

该映射 组合子接受一个稳定函数 f，并将其逐点应用于给定信号。f的类型为 $\square(A \to B)$ 而不仅仅是 $A \to B$，这一事实至关重要：由于 $A \to B$ 不是稳定类型，在延迟计算的作用域中，f原本将不在作用域内，而递归调用需要在那里使用 f。这也有直观的理由：该函数将被应用于输入信号未来任意远处的值，但 $A \to B$ 类型的闭包可能包含对延迟计算的引用，而这些延迟计算在未来可能已经被垃圾回收。

映射 组合子是无状态的，即输出信号的当前值只取决于输入信号的当前值。我们可以将这个组合子推广为扫描，它生成的输出信号除了可能依赖输入信号的当前值外，还可能依赖输出信号的先前值：

扫描: 稳定 $B \Rightarrow \square\,(B \to A \to B) \to B \to$ 信号 $A \to$
信号 B 扫描 f 累加器 $(a :: as) =$ 累加器′ ::延迟 (扫描 f 累
加器′ (adv as))**其中**累加器′ = 解包 f 累加器 a

每当输入信号更新时，输出信号就会根据输入信号的当前值和输出信号的先前值产生一个新值。由于输出信号的先前值

会被访问，因此 B 必须是稳定类型。我们使用 $\Rightarrow$ 符号将此类约束与类型签名区分开来。

例如，我们可以使用 扫描 来计算数字输入信号的总和：

sum : Sig Nat $\to$ Sig Natsum = 扫描 (box ($\lambda m\ n \to m + n$)) 0

通常我们只能访问一个延迟 信号。例如，对于每个推送通道 $\kappa :_c A \in \Delta, c \in \{\mathrm{p}, \mathrm{bp}\}$，我们都有信号

$sigAwait_\kappa$: ⋑ (Sig$A$)$sigAwait_\kappa$ = 延迟 (adv wait$_\kappa$ :: $sigAwait_\kappa$)

例如，我们可能有仅推送通道 mouseClick :$_{\mathrm{p}}$ 1 或 keyPress :$_{\mathrm{p}}$ KeyCode 可用。我们可以为这类信号推导出一个版本的扫描：

$scanAwait$: stable $B \Rightarrow \square\ (B \to A \to B) \to B \to$ ⋑ (Sig A) $\to$ Sig B
$scanAwait\ f\ acc\ as$ = acc ::延迟 ($scan\ f\ acc$ (adv as))

$scanAwait$ 的一个简单用例是一个用于计算给定延迟信号更新次数的组合子，例如按键次数：

$count$: ⋑ (信号 A) $\to$ 自然数 $\to$ SigNat$count$
$s\ n$ = $scanAwait$ (box ($\lambda m \to m + 1$)) $n\ s$

最后，我们有一个最简单的组合子，它只生成一个常量信号：

$const$: $A \to$ 信号
$A\ const\ x$ = x ::永不

单独来看，这个组合子似乎用处不大。引入下一节中的切换组合子后，它的实用性就显而易见了。

3.2 并发信号组合子

到目前为止，我们研究的组合子只接收单个信号，因此无需考虑两个或更多时钟的并发行为。例如，我们可能有两个输入信号，它们由两个冗余传感器生成，传感器各自独立地提供我们感兴趣的读数。要组合这两个信号，我们可以使用以下组合子将它们交错：

$interleave$: $\square\ (A \to A \to A) \to$ ⋑ (Sig A) $\to$ ⋑ (Sig A) $\to$ ⋑ (Sig A)
交错 $f\ xs\ ys$ = 并情形 select $xs\ ys$ 或 f
$Left\ (x :: xs')\ \ ys'$.$x :: interleave\ f\ xs'\ ys'$
$Right\ \ xs'\ \ (y :: ys')$.$y :: interleave\ f\ xs'\ ys'$
$Both\ (x :: xs')\ (y :: ys')$.unbox $f\ x\ y :: interleave\ f\ xs'\ xs'$)

在此定义及后续定义中，我们按预期方式使用简写左、右和两者。例如，左 $s\ t$ 是 $\mathrm{in}_1(\mathrm{in}_1(s,\ t))$ 的简写，也就是说，左侧时钟先滴答的情形。交错 组合子使用 select 等待至少一个输入信号发生时钟滴答，然后相应地更新输出信号。如果两个信号同时发生时钟滴答，则应用提供的合并函数 f。例如，f可以始终采用第一个信号的值，或者取平均值。请注意，生成的信号组合了输入信号的时钟，也就是说，只要任一输入信号发生时钟滴答，它就会发生时钟滴答。

我们也可能希望同时获取两者输入信号的值，在这种情形下，我们会使用zip：

```
zip : 稳定的A, B ⇒ 信号A → 信号B → 信号 (A × B)
zip (x :: xs) (y :: ys) = (x, y) :: 延迟 (情形 select xs ys of
                                        左 xs′ ys′.zip xs′ (y ::ys′)右
                                        xs′ ys′.zip (x ::xs′) ys′两者
                                        xs′ ys′.zip xs′ ys′)
```

与交错类似，*zip* 生成的输出信号会在任一输入信号发生时钟滴答时产生时钟滴答。不过请注意，在左 和右 两种情形下，我们会将未发生时钟滴答的信号先前观测到的值复制到未来。因此，我们需要这两种类型，*A* 和 *B*,都是稳定的。

最后，我们考虑信号切换。我们希望生成一个信号，它最初的行为类似于给定的输入信号，但一旦发生某个事件，就切换到另一个信号。这个想法通过*switch* 函数实现：

```
switch : 信号 A → ◁ (信号 A) → 信号 A
switch (x :: xs) d = x :: 延迟 (情形 selectxs d of
                                  Left  xs′ d′.switch xs′ d′
                                  Right _   d′.d′
                                  Both  _   d′.d′)
```

表示信号未来变化的事件会表示为延迟信号；一旦这个延迟信号发生时钟滴答，就会接管，如同右 和两者 情形一样。借助*switch*，我们可以构建动态数据流图，因为我们会用一个全新的信号替换给定信号；与原信号相比，这个新信号可能依赖于不同的输入通道和中间信号。

我们将在下一节展示这种动态行为的一个示例。为此，我们先准备一个*switch*的变体，其中新信号取决于前一个信号的值：

```
switchf : 稳定 A ⇒ 信号 A → ◁ (A → 信号 A) → 信号 A
switchf (x :: xs) d = x :: 延迟 (情形 selectxs d of
                                   Left       xs′  d′.switchf xs′ d′
                                   Right      _    d′.d′ x
                                   Both (x′ :: xs′) d′.d′ x′)
```

这个组合子不是等待一个新信号，而是等待一个函数 来生成新信号，并将第一个信号的最后一个值传给这个函数。

3.3 一个简单的 SimpleGUI 示例

为了演示如何使用我们的信号组合子，我们考虑一个非常简单的图形用户界面应用程序示例：我们的目标是编写一个具有两个输出通道的响应式程序，用于描述两个文本字段的内容。为此，两个输出通道的类型被指定为 SigNat。每当用户点击一个按钮时，文本字段中显示的数字都应递增；该按钮可作为输入通道 up :$_{\mathrm{p}}$ 1 ∈ Δ 使用。不过，只有一个“up”按钮，用户可以通过“toggle”按钮更改“up”按钮应修改哪个文本字段；“toggle”按钮可作为输入通道 toggle :$_{\mathrm{p}}$ 1 ∈ Δ 使用。

这意味着，第一个文本字段的内容可以用 *count* 组合子描述，但按下“切换”后，就会切换为由 *const* 组合子描述的信号。另一个文本字段的行为则相反：先是 *const*，然后是 *co unt*。这种在两种行为之间持续切换的过程，可以用下面这个组合子简洁地描述：

```
积分: 浮点数 → 信号 浮点数 → 信号 浮点数
积分 cur (0:: xs)  = cur :: 延迟 (积分 cur (adv xs))
积分 cur (x :: xs)  = cur :: 延迟 (情形 selectxs 等待sample of
                                    左 xs′ _.        积分 cur xs′
                                    右 xs′ dt。      积分 (当前 +x × dt) (x :: xs′)
                                    两者 (x′ :: xs′) dt.积分 (当前 +x′ × dt) (x′ :: xs′))

导数 : 信号 浮点数 → 信号 浮点数 derivative xs =求导 0 (头部 xs) xs where 求
导 : 浮点数 → 浮点数 → 信号 浮点数 → 信号 浮点数 求导 0 last (x :: xs) = 0 ::
延迟 ( let x′ :: xs′ = adv xs in 求导 ((x′ −x) / 读取sample) x (x′ :: xs′)) 求导 d last (x ::
xs) = d :: 延迟 ( 情形 select xs 等待 sample of 左 xs′ _. 求导 d last xs′ 右 xs′ dt. 求
导 ((x −last) / dt) x (x :: xs′) 两者 (x′ :: xs′) dt.求导 ((x′ −last) / dt) x′ (x′ :: xs′))
```

图 4. 积分和导数信号组合子。

```
toggleSig : 稳定 A ⇒ □ ( ∋ 1) → □ (A → 信号 A) → □ (A → 信号 A) → A → 信
号 A toggleSig tog f g x = switchf (解包 f x) (延迟 (adv 时钟滴答;toggleSig tog g f ))
  其中tick = 解包 tog
```

第一个参数提供了用于确定何时在两种行为之间切换的事件，而这两种行为分别作为接下来的两个参数给出。在实现中，我们使用记法 $s; t$ 作为 let hi = s in t 的简写。*toggleSig* 组合子使用 *switchf*，先从 f 提供的第一个信号开始，然后通过递归调用交换两个参数 f 和 g 的顺序，在切换 *tog* 发生时钟滴答后立即切换到 g。

现在，只需向*toggleSig*提供适当的输入信号，即可实现描述这两个文本字段的输出通道：

```
字段1，字段2： Sig Nat
字段1 =toggleSig (盒 等待toggle) (盒 (计数 信号等待up)) (盒 常量) 0字段2 =
toggleSig (盒 等待toggle) (盒 常量) (盒 (计数 信号等待up)) 0
```

请注意，程序执行期间数据流图会发生变化，以及这种变化如何反映在与输出通道关联的时钟上：第一个文本字段的输出通道最初的时钟为 {up,toggle} 因为它既要统计“up”按钮被点击的次数，也要在“toggle”按钮被点击时改变其行为。点击“toggle”按钮后，文本字段输出通道的时钟便变为 {toggle} 因为此时它会忽略“up”按钮。我们将在 第 4.3 节中更详细地考察此示例的运行时行为。

3.4 积分与导数

缓冲输入通道可用于表示在离散时间点发生变化、但其当前值可以随时访问的输入信号。例如，给定一个缓冲推送通道 κ： $_{\text{bp}}\ A \in \Delta$，我们可以构造如下信号（使用*sigAwait*$_\kappa$ 自第 3.1 节）：

sig_κ : 一个信号$A\ sig$
$_\kappa$ = 读取$_\kappa$ ::$sigAwait_\kappa$
为了说明我们可以如何处理这类输入信号，我们假设 Async RaTT 具有稳定类型浮点数，并支持浮点数的常见运算。图4 给出了两个信号组合子的定义，它们各自接收一个值为浮点数的信号，并生成该信号的积分和导数。为此，我们假设有一个缓冲推送通道sample：$_{\mathsf{bp}}$ 浮点数 $\in \Delta$，它会 s 以固定间隔（例如每秒 10 次）产生一个新的浮点数。这个数 s 表示通道上次更新以来经过的秒数，例如， $s = 0.1$ 如果sample 每秒产生 10 次时钟滴答。

积分组合子从给定的常数开始，生成给定信号的积分；该常数作为第一个参数提供。其实现采用一种简单的近似方法：每当采样通道产生一个值时，就对底层信号的值进行采样，并加上由该信号值与自上次采样以来经过的时间构成的矩形面积。

定义中的第一个方程是一种优化，可以省略。它表示：如果底层信号的当前值为 0，我们只需等待底层信号更新，因为在底层信号取得非零值之前，积分的值不会改变。因此，我们不必在采样通道每次时钟滴答时都进行采样。

与积分类似，我们可以实现一个导数 函数：给定一个浮点数值信号，生成其导数。与积分 函数一样，导数 也会在采样通道每次时钟滴答时对底层信号进行采样。为此，它使用辅助函数求导，该函数还接收两个参数：导数的当前值，以及采样通道最近一次输入时底层信号的值。与积分类似，求导 的第一行执行了一项优化：如果计算出的导数值为 0，采样将暂停，直到底层信号更新。一旦底层信号更新，我们就假定采样通道发生了时钟滴答，从而及时更新导数。

这两个组合子可以轻松地从浮点数推广到任意向量空间。这样便可以用来描述对多维传感器数据作出反应的复杂行为。

3.5 将表层语法展开为核心演算

为了说明表层语言如何展开为 Async RaTT 核心演算，我们重新来看映射的定义

映射 : □ $(A \to B) \to$ 信号 $A \to$ 信号 B 映射 $f\,(x ::$
$xs) =$ 拆箱 $f\,x ::$ 延迟 (映射 $f\,(\mathsf{adv}\ xs)$)
它会展开为普通 Async RaTT 中的以下项：

$$map = \mathsf{fix}\, r.\lambda f.\lambda s.\mathsf{let}\, x = \pi_1(\mathsf{out}\, s)\ \mathsf{in}\ \mathsf{let}\, xs = \pi_2(\mathsf{out}\, s)$$
$$\mathsf{in}\ \mathsf{into}\Big(unbox\ f\ x, \mathsf{delay}_{\mathsf{cl}(xs)}\,(\mathsf{adv}\, r\ f\ (\mathsf{adv}\ xs))\Big)$$

回顾一下， $s ::\ t$ 是 into (s, t) 的简写。模式匹配会被翻译为相应的消去形式：递归类型使用 out，map产品类型使用 π_i，和类型使用 case。递归语法——map出现在其定义体中——会被翻译为不动点 $\mathsf{fix}\, r.t$，因此递归出现的 map会被 adv r 替代。因此，递归调用必须始终位于 ✓ 的作用域内；map 的定义正是如此，因为它位于延迟的作用域内。此外，我们省略 delay 的下标 cl（xs），因为从延迟作用域内出现了项 advxs 这一事实，可以唯一推断出该下标。

此外，我们会使用 *map* 和 *scan* 这样的顶层定义，它们之后可以在任何上下文中使用。例如，在 ✓ 的作用域内，*scan* 被用于 *scanAwait* 的定义中。我们可以认为，这些顶层定义在定义时会被隐式装箱，在之后使用时再拆箱。也就是说，这些定义在核心演算中会被翻译如下：

$$\begin{aligned}&\text{let } \mathit{scan} = \mathsf{box}(\dots) \text{ in}\\&\text{let } \mathit{scanAwait} = \mathsf{box}(\dots(\mathsf{unbox}\ \mathit{scan})\dots)) \text{ in}\\&\dots\end{aligned}$$

4 操作语义与操作性保证

我们分两个阶段描述 Async RaTT 的操作语义：首先在第 4.1 节介绍求值语义，说明如何在某个特定时刻对 Async RaTT 项求值。除此之外，求值语义还描述了在推送通道上有新输入到达时，为响应这一到达而进行更新所需的计算。随后，我们在第 4.2 节介绍反应式语义，以刻画 Async RaTT 程序随时间推移的动态行为。反应式语义是一台等待新输入到达的机器；新输入到达后，它会为依赖于该输入的输出通道计算新值。对于后者，反应式语义会调用求值语义来执行必要的更新计算。

最后，在第 4.3 节通过一个示例展示操作语义之后，我们将在第 4.4 节结束对操作语义的讨论，并精确阐述关于其性质的主要技术结果：生产性、因果性、信号独立性，以及不存在隐式空间泄漏。为证明最后一点，求值语义使用一个存储区，其中保存外部输入和延迟计算。依赖的数据一旦到达，延迟计算便会立即被垃圾回收。通过这种方式，只要能够证明操作语义永不陷入停滞，Async RaTT 就能从设计上避免隐式空间泄漏。

4.1 求值语义

图 5 将求值语义定义为确定性大步操作语义。我们用 $\langle t;\sigma\rangle \Downarrow^{\iota}\langle v;\tau\rangle$ 表示：给定一个项 t、一个存储 σ和一个输入缓冲区 ι后，机器计算出一个值 v 和一个新的存储 τ。在计算过程中，机器可以将尚未求值的项存入存储 σ，将计算推迟到未来，以便稍后取出并求值。反过来，机器也可以取出先前推迟求值的项并立即对其求值。此外，机器还可以从存储 σ 中读取最近更新的推送通道的新值，并从输入缓冲区读取任意带缓冲通道的当前值 ι。

为了实现计算的延迟，语言的语法包含堆位置 l。这些位置在演算中不可定型，但机器可以在求值过程中引入它们。堆位置表示一个延迟计算；当特定时钟发生时钟滴答后，该计算即可恢复，这意味着延迟计算正在等待的数据已经到达。为此，每个堆位置 l 都与一个时钟相关联，记作 $\mathsf{cl}\,(l)$。时钟 $\mathsf{cl}\,(l)$一旦发生时钟滴答，就可以通过取出存储在堆位置 l 中的未求值项并对其求值，来恢复 l 所表示的延迟计算。我们用 Loc 表示所有堆位置的集合，并假设对于每个时钟 Θ，都有可数无穷多个位置 l 满足 $\mathsf{cl}\,(l) = \Theta$。时钟 Θ 是取自 $\mathsf{dom}\,(\Delta)$ 的有限个推送通道组成的集合；只要其中任一通道 $\kappa \in \Theta$ 更新，时钟就会发生时钟滴答。例如，假设图形用户界面的输入上下文为 Δ，那么时钟 {keyPressed，mouseCoord} 会在用户按键或移动鼠标时发生时钟滴答。请注意，我们现在更精确地区分了时钟表达式（通常记作 θ）和时钟（通常记作 Θ）。一个闭合的

$$\frac{}{\langle v;\sigma\rangle \Downarrow^\iota \langle v;\sigma\rangle} \qquad \frac{\langle t;\sigma\rangle \Downarrow^\iota \langle v;\sigma'\rangle \qquad \langle t';\sigma'\rangle \Downarrow^\iota \langle v';\sigma''\rangle}{\langle (t,t');\sigma\rangle \Downarrow^\iota \langle (v,v');\sigma''\rangle} \qquad \frac{\langle t;\sigma\rangle \Downarrow^\iota \langle (v_1,v_2);\sigma'\rangle \qquad i\in\{1,2\}}{\langle \pi_i(t);\sigma\rangle \Downarrow^\iota \langle v_i;\sigma'\rangle}$$

$$\frac{\langle t;\sigma\rangle \Downarrow^\iota \langle v;\sigma'\rangle \qquad i\in\{1,2\}}{\langle \mathsf{in}_i(t);\sigma\rangle \Downarrow^\iota \langle \mathsf{in}_i(v);\sigma'\rangle} \qquad \frac{\langle t;\sigma\rangle \Downarrow^\iota \langle \mathsf{in}_i(v);\sigma'\rangle \qquad \langle t_i[v/x];\sigma'\rangle \Downarrow^\iota \langle v_i;\sigma''\rangle \qquad i\in\{1,2\}}{\langle \mathsf{case}\ t\ \mathsf{of}\ \mathsf{in}_1\ x.t_1;\mathsf{in}_2\ x.t_2;\sigma\rangle \Downarrow^\iota \langle v_i;\sigma''\rangle}$$

$$\frac{\langle t;\sigma\rangle \Downarrow^\iota \langle \lambda x.s;\sigma'\rangle \qquad \langle t';\sigma'\rangle \Downarrow^\iota \langle v;\sigma''\rangle \qquad \langle s[v/x];\sigma''\rangle \Downarrow^\iota \langle v';\sigma'''\rangle}{\langle t\ t';\sigma\rangle \Downarrow^\iota \langle v';\sigma'''\rangle}$$

$$\frac{\langle s;\sigma\rangle \Downarrow^\iota \langle v;\sigma'\rangle \qquad \langle t[v/x];\sigma'\rangle \Downarrow^\iota \langle w;\sigma''\rangle}{\langle \mathsf{let}\ x=s\ \mathsf{in}\ t;\sigma\rangle \Downarrow^\iota \langle w;\sigma''\rangle} \qquad \frac{\kappa\in \mathsf{dom}\,(\iota)}{\langle \mathsf{read}_\kappa;\sigma\rangle \Downarrow^\iota \langle \iota(\kappa);\sigma\rangle}$$

$$\frac{l=\mathsf{alloc}^{|\theta|}\,(\sigma)}{\langle \mathsf{delay}_\theta\ t;\sigma\rangle \Downarrow^\iota \langle l;(\sigma,l\mapsto t)\rangle} \qquad \frac{l=\mathsf{alloc}^{\emptyset}\,(\sigma)}{\langle \mathsf{never};\sigma\rangle \Downarrow^\iota \langle l;\sigma\rangle}$$

$$\frac{}{\langle \mathsf{adv}\,\mathsf{wait}_\kappa;\eta_N\,\langle\kappa\mapsto v\rangle\,\eta_L\rangle \Downarrow^\iota \langle v;\eta_N\,\langle\kappa\mapsto v\rangle\,\eta_L\rangle} \qquad \frac{\langle \eta_N(l);\eta_N\,\langle\kappa\mapsto v\rangle\,\eta_L\rangle \Downarrow^\iota \langle w;\sigma\rangle}{\langle \mathsf{adv}\,l;\eta_N\,\langle\kappa\mapsto v\rangle\,\eta_L\rangle \Downarrow^\iota \langle w;\sigma\rangle}$$

$$\frac{\kappa\in |\mathsf{cl}\,(v_i)|\setminus|\mathsf{cl}\,(v_{3-i})| \qquad \langle \mathsf{adv}\,v_i;\eta_N\,\langle\kappa\mapsto w\rangle\,\eta_L\rangle \Downarrow^\iota \langle u_i;\sigma\rangle \qquad u_{3-i}=v_{3-i}}{\langle \mathsf{select}\,v_1\,v_2;\eta_N\,\langle\kappa\mapsto w\rangle\,\eta_L\rangle \Downarrow^\iota \langle \mathsf{in}_1(\mathsf{in}_i\,(u_1,u_2));\sigma\rangle}$$

$$\frac{\kappa\in |\mathsf{cl}\,(v_1)|\cap|\mathsf{cl}\,(v_2)| \qquad \langle \mathsf{adv}\,v_1;\eta_N\,\langle\kappa\mapsto w\rangle\,\eta_L\rangle \Downarrow^\iota \langle u_1;\sigma\rangle \qquad \langle \mathsf{adv}\,v_2;\sigma\rangle \Downarrow^\iota \langle u_2;\sigma'\rangle}{\langle \mathsf{select}\,v_1\,v_2;\eta_N\,\langle\kappa\mapsto w\rangle\,\eta_L\rangle \Downarrow^\iota \langle \mathsf{in}_2\,(u_1,u_2)\,;\sigma'\rangle}$$

$$\frac{\langle t;\sigma\rangle \Downarrow^\iota \langle v;\sigma'\rangle}{\langle \mathsf{suc}\,t;\sigma\rangle \Downarrow^\iota \langle \mathsf{suc}\,v;\sigma'\rangle} \qquad \frac{\langle n;\sigma\rangle \Downarrow^\iota \langle 0;\sigma'\rangle \qquad \langle s;\sigma'\rangle \Downarrow^\iota \langle v;\sigma''\rangle}{\langle \mathsf{rec}_{\mathsf{Nat}}(s,x\,y.t,n);\sigma\rangle \Downarrow^\iota \langle v;\sigma''\rangle}$$

$$\frac{\langle n;\sigma\rangle \Downarrow^\iota \langle \mathsf{suc}\,v;\sigma'\rangle \qquad \langle \mathsf{rec}_{\mathsf{Nat}}(s,x\,y.t,v);\sigma'\rangle \Downarrow^\iota \langle v';\sigma''\rangle \qquad \langle t[v/x,v'/y];\sigma''\rangle \Downarrow^\iota \langle w;\sigma'''\rangle}{\langle \mathsf{rec}_{\mathsf{Nat}}(s,x\,y.t,n);\sigma\rangle \Downarrow^\iota \langle w;\sigma'''\rangle}$$

$$\frac{\langle t[\mathsf{dfix}\,x.t/x];\sigma\rangle \Downarrow^\iota \langle v;\sigma'\rangle}{\langle \mathsf{adv}\,(\mathsf{dfix}\,x.t);\sigma\rangle \Downarrow^\iota \langle v;\sigma'\rangle} \qquad \frac{\langle t[\mathsf{dfix}\,x.t/x];\sigma\rangle \Downarrow^\iota \langle v;\sigma'\rangle}{\langle \mathsf{fix}\,x.t;\sigma\rangle \Downarrow^\iota \langle v;\sigma'\rangle}$$

$$\frac{\langle t;\sigma\rangle \Downarrow^\iota \langle \mathsf{box}\,t';\sigma'\rangle \qquad \langle t';\sigma'\rangle \Downarrow^\iota \langle v;\sigma''\rangle}{\langle \mathsf{unbox}\,t;\sigma\rangle \Downarrow^\iota \langle v;\sigma''\rangle} \qquad \frac{\langle t;\sigma\rangle \Downarrow^\iota \langle v;\sigma'\rangle}{\langle \mathsf{into}\,t;\sigma\rangle \Downarrow^\iota \langle \mathsf{into}\,v;\sigma'\rangle} \qquad \frac{\langle t;\sigma\rangle \Downarrow^\iota \langle \mathsf{into}\,v;\sigma'\rangle}{\langle \mathsf{out}\,t;\sigma\rangle \Downarrow^\iota \langle v;\sigma'\rangle}$$

图 5. 操作语义。

时钟表达式 θ 按如下方式求值为时钟 $|\theta|$：

$$|\mathsf{cl}\,(l)| = \mathsf{cl}\,(l) \qquad |\mathsf{cl}\,(\mathsf{wait}_\kappa)| = \{\kappa\} \qquad |\theta \sqcup \theta'| = |\theta| \cup |\theta'|$$

延迟计算存放在堆中，堆只是一个从堆位置到项的有限映射 η。我们特别关注这样一类堆 η：其位置记作 $\mathsf{dom}\,(\eta)$，每个位置都有一个包含给定输入通道 κ的时钟：

$$\mathsf{Heap}^\kappa = \{\eta\in\mathsf{Heap}\mid \forall l\in\mathsf{dom}\,(\eta)\,.\,\kappa\in\mathsf{cl}\,(l)\,\}$$

一旦输入通道κ上有新值到达，就可以安全地计算存储在堆 $\eta \in \mathrm{Heap}^{\kappa}$中的项。这一直觉体现在存储 σ的表示中，存储可以采用两种形式之一：单堆存储 η_L，或双堆存储 $\eta_N \mathrm{h}\kappa \mapsto v \mathrm{i}\ \eta_L$，其中包含 $\eta_N \in \mathrm{Heap}^{\kappa}$。我们通常将 η_L 称为稍后堆，用于存储延迟计算以供稍后使用；将 η_N称为现在堆，其中存储的项可以立即安全地计算。双堆存储的 $\mathrm{h}\kappa \mapsto v\mathrm{i}$ 组件表示输入通道 κ 已更新为新值 v。因此，机器可以安全地从 η_N恢复计算，因为延迟计算在 η_N中等待的数据已经到达。

我们先来考虑延迟的语义：为了在存储中分配新位置，我们假设存在一个函数 $\mathrm{alloc}^{\Theta}(\cdot)$。给定存储 $\eta_L \mathrm{or}\ \eta_N \mathrm{h}\kappa \mapsto v \mathrm{i}\ \eta_L$ 时，该函数会生成一个位置 $l \notin \mathrm{dom}\,(\eta_L)\ \mathrm{with}\ \mathrm{cl}\,(l) = \Theta$。这样分别得到存储 $\eta_L,\ l \mapsto t \mathrm{or}\ \eta_N \mathrm{h}\kappa \mapsto v \mathrm{i}\ \eta_L,\ l \mapsto t$，其中$\eta_L,\ l \mapsto t$表示在堆 η_L中扩展映射 $l \mapsto t$。

反过来，$\mathrm{adv}\,l$会取回先前延迟的计算。类型规则确保 $\mathrm{adv}\,l$只会在形如 $\eta_N \mathrm{h}\kappa \mapsto v \mathrm{i}\ \eta_L$ 且满足 $l \in \mathrm{dom}\,(\eta_N)$、因而也满足 $\kappa \in \mathrm{cl}\,(l)$ 的存储上下文中求值。此外，adv 还可以应用于 wait_{κ}，它只会从通道 κ中查找新值 v。

select 组合子使我们能够同时与两个延迟计算交互。其语义会检查三种可能的情况，即两个延迟计算中有哪些非空子集已被触发。两个参数值v_1或 v_2要么是堆位置，要么是 wait_{κ}，因此机器只需检查当前输入通道 κ 是否属于与 v_1、v_2或两者相关联的时钟。根据检查结果，机器会推进相应的值。

最后，不动点组合子 fix 借助组合子 dfix 求值。与堆位置类似，dfix 在演算中不可定型，而是由机器引入的。直观地说，我们可以将形如 dfix $x.t$的值看作 $\Lambda\theta.(\mathrm{delay}_{\theta}(\mathrm{fix}\ x.t))$ 的简写。也就是说，dfix $x.t$是一个延迟求值闭包，给定时钟 θ后会产生一个关于 θ的延迟计算；该计算会在经历一次 θ 时钟滴答后求出一个不动点。因此，adv 组合子对 ⓥ 的作用也可以理解为：先提供时钟 θ，再推进延迟计算 $\mathrm{delay}_{\theta}(\mathrm{fix}\ x.t)$，这意味着对 fix $x.t$求值。

4.2 反应式语义

Async RaTT 程序通过从一组输入通道接收输入并向一组输出通道发送输出，从而与其环境交互。输入上下文 Δ 描述可用的输入通道。此外，我们还有一个输出上下文 Γ_{out}，其中仅包含变量 $x : A$，其中 A 是一个值类型。我们将 Γ_{out} 中的变量称为输出通道。综合起来，我们将由 Δ 和 Γ_{out} 组成的二元组称为反应式接口，记作 $\Delta \Rightarrow \Gamma_{\mathrm{out}}$。

给定一个输出接口 $\Gamma_{\mathrm{out}} = x_1 : A_1, \ldots, x_n : A_n$，我们将类型 $\mathrm{Prod}\,(\Gamma_{\mathrm{out}})$ 定义为 Γ_{out}中所有类型的积，即 $\mathrm{Prod}\,(\Gamma_{\mathrm{out}}) = 信号\, A_1 \times \cdots \times 信号\, A_n$。此处使用的 n 元积类型可以按标准方式使用二元积类型和单位类型进行编码。若 Async RaTT 项 t 满足 $\vdash_{\Delta} t : \mathrm{Prod}\,(\Gamma_{\mathrm{out}})$，则称其为实现反应式接口 $\Delta \Rightarrow \Gamma_{\mathrm{out}}$的响应式程序，记作 $t : \Delta \Rightarrow \Gamma_{\mathrm{out}}$。

响应式程序的操作语义由图 6 中的机器描述。机器的状态可以有两种不同的形式：最初，机器处于以下形式的状态$\langle t; \iota \rangle$，其中 $t : \Delta \Rightarrow \Gamma_{\mathrm{out}}$ 是响应式程序，而 ι是初始输入缓冲区，其中包含所有缓冲输入通道的初始值。随后，机器状态是一个二元组 $\langle N; \sigma; \iota \rangle$，其中 N是形如 $x_1 \mapsto l_1, \ldots, x_n \mapsto l_n$ 的序列，它将输出通道 $x_i \in \mathrm{dom}\,(\Gamma_{\mathrm{out}})$映射到堆位置。也就是说，$N$为每个输出通道记录将要产生该输出通道下一个值的延迟计算所在的位置，并在需要更新时执行该计算。

$$\textsc{init}\ \frac{\langle\langle t\rangle;\emptyset\rangle \Downarrow^{\iota} \langle\langle v_1 :: l_1, \ldots, v_m :: l_m\rangle;\eta\rangle}{\langle t;\iota\rangle \overset{x_1\mapsto v_1,\ldots,x_m\mapsto v_m}{\Longrightarrow} \langle x_1 \mapsto l_1, \ldots, x_m \mapsto l_m;\eta;\iota\rangle}$$

$$\textsc{input}\ \frac{\iota' = \iota[\kappa \mapsto v] \text{ if } \kappa \in \mathrm{dom}\,(\iota) \text{ otherwise } \iota' = \iota}{\langle N;\eta;\iota\rangle \overset{\kappa\mapsto v}{\Longrightarrow} \left\langle N;[\eta]_{\kappa\in}\langle\kappa\mapsto v\rangle[\eta]_{\kappa\notin};\iota'\right\rangle}$$

$$\textsc{output-end}\ \frac{}{\langle\cdot;\eta_N\langle\kappa\mapsto v\rangle\eta_L;\iota\rangle \overset{\cdot}{\Longrightarrow} \langle\cdot;\eta_L;\iota\rangle}$$

$$\textsc{output-skip}\ \frac{\kappa\notin \mathrm{cl}\,(l) \qquad \langle N;\eta_N\langle\kappa\mapsto v\rangle\eta_L;\iota\rangle \overset{O}{\Longrightarrow} \langle N';\eta;\iota\rangle}{\langle x\mapsto l,N;\eta_N\langle\kappa\mapsto v\rangle\eta_L;\iota\rangle \overset{O}{\Longrightarrow} \langle x\mapsto l,N';\eta;\iota\rangle}$$

$$\textsc{output-compute}\ \frac{\kappa\in \mathrm{cl}\,(l) \qquad \langle \mathrm{adv}\,l;\eta_N\langle\kappa\mapsto v\rangle\eta_L\rangle \Downarrow^{\iota} \langle v' :: l';\sigma\rangle \qquad \langle N;\sigma;\iota\rangle \overset{O}{\Longrightarrow} \langle N';\eta;\iota\rangle}{\langle x\mapsto l,N;\eta_N\langle\kappa\mapsto v\rangle\eta_L;\iota\rangle \overset{x\mapsto v',O}{\Longrightarrow} \langle x\mapsto l',N';\eta;\iota\rangle}$$

图 6. 反应式语义。

机器可以进行三种迁移：

an initialisation transition $\langle t;\iota\rangle \overset{O}{\Longrightarrow} \langle N;\eta;\iota\rangle$

an input transition $\langle N;\eta;\iota\rangle \overset{\kappa\mapsto v}{\Longrightarrow} \langle N;\eta_N\langle\kappa\mapsto v\rangle\eta_L;\iota'\rangle$

an output transition $\langle N;\eta_N\langle\kappa\mapsto v\rangle\eta_L;\iota\rangle \overset{O}{\Longrightarrow} \langle N';\eta_L;\iota\rangle$

其中 O 是一个序列 $x_1 \mapsto v_1, \ldots\ldots, x_n \mapsto v_n$，它将输出通道映射到值。初始迁移会初始化所有输出通道的值；此后，机器交替执行输入转换和输出转换。每个输入转换都会更新一个输入通道的值，并可能更新输入缓冲区（如果新输入来自带缓冲通道）；每个输出转换则会为由紧接在前的输入转换触发的所有输出通道提供新值。

初始化转换会在初始输入缓冲区 ι 的上下文中计算响应式程序 t，由此产生一个元组 $\mathbf{h}v_1 :: l_1, \ldots\ldots, v_m :: l_m\mathbf{i}$，其分量 $v_i :: l_i$ 对应于输出通道 $x_i : A_i \in \Gamma_{\mathrm{out}}$。每个 v_i 都是输出通道 x_i 的初始值，而每个l_i都指向堆 η 中的一个延迟计算，该计算会求出 x_i 的未来值。

输入迁移会在输入通道 κ 上接收更新后的值 v，并通过更新输入缓冲区（如果其中已有 κ的值）以及将存储 η 迁移至新存储$[\eta]_{\kappa\in}\mathbf{h}\kappa\mapsto v\mathbf{i}\ [\eta]_{\kappa\notin}$来作出响应。这会将堆 η 划分为包含 κ 的部分和不包含它的部分：

$$[\eta]_{\kappa\in}(l) = \eta(l) \quad \text{if } \kappa\in \mathrm{cl}\,(l) \qquad\qquad [\eta]_{\kappa\notin}(l) = \eta(l) \quad \text{if } \kappa\notin \mathrm{cl}\,(l)$$

也就是说，在后续的输出迁移中，机器可以从 $[\eta]_{\kappa\in}$读取，即恰好读取 η 中等待从 κ接收输入的那些堆位置，并从 κ访问新值 v。

最后，输出转换会检查 $x \mapsto l$ 中的每个元素，判断它是否应该推进，因为它依赖于 κ（输出计算），还是应该保持不变，因为它不依赖于 κ（输出跳过）。只有在输出计算的情形下，才会为 x 生成新的输出值。最后，输出转换会执行所需的垃圾回收，删除

当前堆 η_N 和输入值 v（输出结束）。这也意味着，输出计算执行的更新不仅是可行的（因为所需数据已到达），也是必要的（因为在机器执行此次输出转换后，输入数据以及它们所依赖的延迟计算都会消失）。

4.3 示例

为了了解操作语义如何运作，我们重新考察第 3.3 节中的简单图形用户界面程序，并在机器上运行它。为此，我们首先按照第 *3.5* 节中的描述，将 toggleSig 的定义展开为核心演算中的显式不动点项：

$$\mathit{toggleSig} = \text{不动点 } r.\lambda \mathit{tog}.\lambda f.\lambda g.\lambda x.\text{令 } \mathit{tick} = \text{拆箱 } \mathit{tog} \text{ 于 } \mathit{switchf}\,(\text{拆箱 } f\ x)(\text{延迟}^{\mathrm{cl}(\mathit{tick})}(\mathit{adv}\mathit{tick};\ \mathrm{adv}\ r\mathit{tog}\ g\ f))$$

执行过程中，机器会将 *toggleSig* 这样的不动点转换为延迟不动点，使用 dfix 代替 不动点。我们用 *toggleSig*′ 表示这个延迟不动点，即 *toggleSig*′ 是将 *toggleSig* 中的 不动点替换为 dfix后得到的。对于其他不动点定义，我们也采用相同的记号约定，将 $\mathit{sigAwait}'_\kappa$ 和 *scan*′ 分别表示第 dfix 版本的 $\mathit{sigAwait}_\kappa$ 和 *scan*（见第 3.1 节）。

我们考虑程序 ***field1*** ： $\Delta \Rightarrow \Gamma_{\mathrm{out}}$ with $\Delta = \{\mathrm{up} :_{\mathrm{p}} 1,\ \mathrm{toggle} :_{\mathrm{p}} 1\}, \Gamma_{\mathrm{out}} = x :$ 自然数，且

$$\begin{aligned} \mathit{field1} = \mathit{toggleSig}\ t\ s_1\ s_2\ 0 \quad \text{where} \quad & t = \mathrm{box}\,\mathrm{wait}_{\mathrm{toggle}} \\ & s_1 = \mathrm{box}\,(\mathit{count}\ \mathit{sigAwait}_{\mathrm{up}}) \\ & s_2 = \mathrm{box}\ \mathit{const} \end{aligned}$$

也就是说，这个程序描述了文本字段的行为：它最初处于焦点状态，因此会对“up”按钮作出响应。

为了更清晰地展示机器的转换步骤，我们将机器的存储仅写成其堆位置列表，并在下方分别写出各位置的内容及其时钟。机器的第一步执行初始化，为输出信号提供初始值：

$$\langle \mathit{field1}; \emptyset\rangle \overset{x\mapsto 0}{\Longrightarrow} \langle x \mapsto l_1; l_1, l_2, l_3, l_4; \emptyset\rangle$$

$$\begin{aligned} \text{where} \quad & l_1 \mapsto \mathrm{case\ select}\ l_3\ l_4\ \mathrm{of}\ \ldots && \mathrm{cl}\,(l_1) = \{\mathrm{toggle}, \mathrm{up}\} \\ & l_2 \mapsto \mathrm{adv}\,\mathrm{wait}_{\mathrm{up}} :: \mathrm{adv}\ \mathit{sigAwait}' && \mathrm{cl}\,(l_2) = \{\mathrm{up}\} \\ & l_3 \mapsto \mathit{scan}\,(\mathrm{box}\,\lambda n.\lambda m.m+1)\,0\,(\mathrm{adv}\,l_2) && \mathrm{cl}\,(l_3) = \{\mathrm{up}\} \\ & l_4 \mapsto \mathrm{adv}\,\mathrm{wait}_{\mathrm{toggle}}; \mathrm{adv}\ \mathit{toggleSig}'\ t\ s_2\ s_1 && \mathrm{cl}\,(l_4) = \{\mathrm{toggle}\} \end{aligned}$$

我们可以看到，输出通道 x 的下一个值由位置 l_1处的延迟计算提供，并且由于 $\mathrm{cl}\,(l_1) = \{\mathrm{toggle}$，已切换$\}$，我们知道 x会在用户点击两个按钮中的任意一个后立即产生新值。如果用户点击“向上”按钮，我们会看到以下内容：

$$\langle x \mapsto l_1; l_1, l_2, l_3, l_4; \emptyset\rangle \overset{\mathrm{up}\mapsto\langle\rangle}{\Longrightarrow} \langle x \mapsto l_1; l_1, l_2, l_3\ \langle \mathrm{up} \mapsto \langle\rangle\rangle\ l_4; \emptyset\rangle \overset{x\mapsto 1}{\Longrightarrow} \langle x \mapsto l_5; l_4, l_5, l_6, l_7; \emptyset\rangle$$

$$\begin{aligned} \text{where} \quad & l_5 \mapsto \mathrm{case\ select}\ l_7\ l_4\ \mathrm{of}\ \ldots && \mathrm{cl}\,(l_5) = \{\mathrm{toggle}, \mathrm{up}\} \\ & l_6 \mapsto \mathrm{adv}\,\mathrm{wait}_{\mathrm{up}} :: \mathrm{adv}\ \mathit{sigAwait}' && \mathrm{cl}\,(l_6) = \{\mathrm{up}\} \\ & l_7 \mapsto \mathrm{adv}\ \mathit{scan}'\,(\mathrm{box}\,\lambda n.\lambda m.m+1)\,0\,(\mathrm{adv}\,l_6) && \mathrm{cl}\,(l_7) = \{\mathrm{up}\} \end{aligned}$$

堆位置 l_1, l_2, l_3 会被垃圾回收，只有 l_4 得以保留，因为只有 l_4 的时钟不包含 up。如果用户现在点击“切换”按钮，我们会看到以下内容：

$$\langle x \mapsto l_5; l_4, l_5, l_6, l_7; \emptyset\rangle \overset{\text{toggle}\mapsto\langle\rangle}{\Longrightarrow} \langle x \mapsto l_5; l_4, l_5\,\langle \text{toggle} \mapsto \langle\rangle\rangle\, l_6, l_7; \emptyset\rangle \overset{x\mapsto 1}{\Longrightarrow} \langle x \mapsto l_8; l_6, l_7, l_8, l_9; \emptyset\rangle$$

$$\text{where} \quad \mathsf{cl}\,(l_0) = \emptyset \quad l_8 \mapsto \mathsf{case}\,\mathsf{select}\,l_0\,l_9\,\mathsf{of}\ldots \qquad \mathsf{cl}\,(l_8) = \{\text{toggle}\}$$

$$l_9 \mapsto \mathsf{adv}\,\mathsf{wait}_{\text{toggle}} :: \mathsf{adv}\,\mathit{sigAwait}'\,t\,s_1\,s_2 \quad \mathsf{cl}\,(l_9) = \{\text{toggle}\}$$

堆位置 l_0 由 never 分配，因此不会出现在堆上。现在，输出通道 x 仅取决于输入通道 toggle。如果用户此时反复点击“向上”按钮，则不会产生输出：

$$\langle x \mapsto l_8; l_6, l_7, l_8, l_9; \emptyset\rangle \overset{\text{up}\mapsto\langle\rangle}{\Longrightarrow} \langle x \mapsto l_8; l_6, l_7\,\langle \text{up} \mapsto \langle\rangle\rangle\, l_8, l_9; \emptyset\rangle \overset{\cdot}{\Longrightarrow} \langle x \mapsto l_8; l_8, l_9; \emptyset\rangle$$

$$\overset{\text{up}\mapsto\langle\rangle}{\Longrightarrow} \langle x \mapsto l_8; \langle \text{up} \mapsto \langle\rangle\rangle\, l_8, l_9; \emptyset\rangle \overset{\cdot}{\Longrightarrow} \langle x \mapsto l_8; l_8, l_9; \emptyset\rangle$$

最后请注意，由于输入上下文 Δ 不包含缓冲输入通道，因此在程序的整个运行过程中，输入缓冲区始终为空。

4.4 主要结果

上述操作语义使我们能够准确陈述 Async RaTT 提供的操作性保证，即生产性、没有隐式空间泄漏、因果性和信号独立性。我们将逐一讨论这些保证。

4.4.1 生产性。响应式程序 $t\colon\ \Delta \Rightarrow \Gamma_{\text{out}}$ 如果我们为 t提供一个良类型的初始输入缓冲区，以及其输入通道上的无限序列良类型输入，那么它将通过输出通道产生无限序列类型良好的输出。要正式陈述生产性性质，我们首先需要准确说明良类型的含义：

- 输入缓冲区 ι称为良类型的，记作 $\vdash \iota\colon \Delta$，如果对于每个满足 $\kappa :_{\mathsf{b}} A \in \Delta$或 $\kappa :_{\mathsf{bp}} A \in \Delta$ 的 κ，都有 $\vdash \iota(\kappa)\colon A$。
- 输入值 $\kappa \mapsto v$称为良类型的，记作 $\vdash \kappa \mapsto v\colon \Delta$，如果 $\kappa :_{c} A \in \Delta$ 且 $\vdash v\colon A$。
- 输出值集合 O称为良类型的，记作 $\vdash O\colon \Gamma_{\text{输出}}$，如果对于所有 $x \mapsto v \in O$，都有$x : A \in \Gamma_{\text{输出}}$ 且 $\vdash v\colon A$。

现在我们可以正式表述生产性性质如下：

定理 4.1（生产性）。给定一个响应式程序 $t\colon\ \Delta \Rightarrow \Gamma_{\text{out}}$、对所有 $i \in \mathbb{N}$均为良类型的输入值$\vdash \kappa_i \mapsto v_i\colon\ \Delta$，以及一个良类型初始输入缓冲区 $\vdash \iota_0\colon\ \Delta$，存在一个无限迁移序列

$$\langle t; \iota_0\rangle \overset{O_0}{\Longrightarrow} \langle N_0; \eta_0; \iota_0\rangle \overset{\kappa_0\mapsto v_0}{\Longrightarrow} \langle N_0; \sigma_0; \iota_1\rangle \overset{O_1}{\Longrightarrow} \langle N_1; \eta_1; \iota_1\rangle \overset{\kappa_1\mapsto v_1}{\Longrightarrow} \ldots$$

对于所有 $i \in \mathbb{N}$，有 $\vdash O_i\colon\ \Gamma_{\text{out}}$。

对于每个输入值，响应式程序总会生成一组输出值 O_{i+1}，$\kappa_i \mapsto v_i$但这组值可能为空。当 N_i中的堆位置都不依赖于输入 κ_i时，也就是说，当 $\kappa_i \notin \mathsf{cl}(l)$对所有 $x \mapsto l \in N_i$都成立时，就会发生这种情况。正如我们将在命题4.4中看到的，对于 $\kappa\colon\ _{\mathsf{b}}\, A \in \Delta$ 仅缓冲的输入，这种情况必然成立。请注意，所有输出通道都在初始化转换中完成初始化。因此，输出值集合为空仅意味着无需更新任何输出通道。

4.4.2 隐式空间泄漏。不存在隐式空间泄漏，是生产性性质（定理4.1的直接结果。更准确地说，Async RaTT 的操作语义经过设计，使得每一对输入/输出转换之后

$$\langle N;\eta;\iota\rangle \overset{\kappa\mapsto v}{\Longrightarrow} \langle N;\eta_N\,\langle\kappa\mapsto v\rangle\,\eta_L;\iota'\rangle \overset{O}{\Longrightarrow} \langle N';\eta_L;\iota'\rangle$$

所有堆位置 l 中 η 依赖于 κ的那些位置，即具有 $\kappa\in$ 闭包(l)的那些位置，都会被垃圾回收，因此不会出现在 η_L中。也就是说，位于位置 l 的延迟计算只会保留在内存中，直到其时钟闭包(l) 时钟滴答。根据定理4.1，这种积极的垃圾回收策略是安全的：机器在尝试解引用已被垃圾回收的堆位置时永远不会陷入停滞。

4.4.3 因果性。下文中，我们将定理 t4.1所得到的响应式程序 t的转换序列简称为 t的良类型转换序列。

若响应式程序 t 的任意良类型转换序列都满足以下条件，则称该程序具有因果性：

$$\langle t;\iota_0\rangle \overset{O_0}{\Longrightarrow} \langle N_0;\eta_0;\iota_0\rangle \overset{\kappa_0\mapsto v_0}{\Longrightarrow} \langle N_0;\sigma_0;\iota_1\rangle \overset{O_1}{\Longrightarrow} \langle N_1;\eta_1;\iota_1\rangle \overset{\kappa_1\mapsto v_1}{\Longrightarrow} \dots \tag{1}$$

每组输出值 O_n 仅依赖于初始输入缓冲区 ι_0 以及此前接收到的输入值 $\kappa_i\mapsto v_i$，其中 $i<n$。为说明这一点始终成立，我们首先注意到，操作语义具有如下意义上的确定性：

引理 4.2（确定性语义）。

(i) $\langle t;\sigma\rangle \Downarrow^{\iota} \langle v_1;\sigma_1\rangle$ *and* $\langle t;\sigma\rangle \Downarrow^{\iota} \langle v_2;\sigma_2\rangle$ *implies that* $v_1=v_2$ *and that* $\sigma_1=\sigma_2$.

(ii) $c \overset{\kappa\mapsto v}{\Longrightarrow} c_1$ *and* $c \overset{\kappa\mapsto v}{\Longrightarrow} c_2$ *implies* $c_1=c_2$.

(iii) $c \overset{O_1}{\Longrightarrow} c_1$ *and* $c \overset{O_2}{\Longrightarrow} c_2$ *implies* $O_1=O_2$ *and* $c_1=c_2$.

因果性现在由定理 4.1 和引理 4.2得出。

推论 4.3（因果性）。假设（*1*）以及以下内容是类型正确的转换序列 *s*

$$\langle t;\iota_0\rangle \overset{O_0'}{\Longrightarrow} \left\langle N_0';\eta_0';\iota_0'\right\rangle \overset{\kappa_0'\mapsto v_0'}{\Longrightarrow} \left\langle N_0';\sigma_0';\iota_1'\right\rangle \overset{O_1'}{\Longrightarrow} \left\langle N_1';\eta_1';\iota_1'\right\rangle \overset{\kappa_1'\mapsto v_1'}{\Longrightarrow} \dots$$

令 $n\in\mathbb{N}$，并假设对于所有 $i<n$，$\kappa_i'=\kappa_i$和 $v_i'=v_i$。则 $O_n'=O_n$。

4.4.4 信号独立性。从反应式语义的定义可以看出，机器仅在输出通道 x：$A\in\Gamma_{\text{out}}$ 依赖于刚刚到达的输入值 $\kappa\mapsto v$时才会更新该通道，也就是说，仅当机器处于状态 $\mathrm{h}N;\eta_N\mathrm{h}\kappa\mapsto v\mathrm{i}\ \eta_L;\ \mathrm{i}$ 且满足 $\kappa\in\mathrm{cl}\,(N(x))$ 时才会更新。然而，类型系统允许我们给出两个有用的静态判据，用于判断何时能够保证 $\kappa\notin\mathrm{cl}\,(N(x))$，因此输出信号 x 无需（事实上也不能）更新。

As allud如前所述，在仅缓冲通道上接收的值永远不会产生输出 t:

命题 4.4（缓冲信号独立性）。假设 t：$\Delta\Rightarrow\Gamma_{\text{out}}$ 是一个响应式程序，并且 *d*

$$\langle t;\iota_0\rangle \overset{O_0}{\Longrightarrow} \langle N_0;\eta_0;\iota_0\rangle \overset{\kappa_0\mapsto v_0}{\Longrightarrow} \langle N_0;\sigma_0;\iota_1\rangle \overset{O_1}{\Longrightarrow} \langle N_1;\eta_1;\iota_1\rangle \overset{\kappa_1\mapsto v_1}{\Longrightarrow} \dots$$

是 t 的一个类型正确的转换序列。那么，当 κ_i：$_{\mathrm{b}}\,A\in\Delta$ 对某个 A 成立时，O_{i+1} 为空。

其次，给定输出信号实现的输入上下文 Δ 为将触发更新的推送通道提供了一个上界：

定理 4.5（推送信号独立性）。假设（t，s）：$\Delta\Rightarrow\Gamma_{\text{out}}$ 是一个响应式程序，并且$\Gamma_{\text{out}}=\Gamma$，$z$：$C$，使得对于某个 $\Delta'\subset\Delta$，s：$\Delta'\Rightarrow$（z：C）也是一个响应式程序，并且

$$\langle(t,s)\,;\iota_0\rangle \overset{O_0}{\Longrightarrow} \langle N_0;\eta_0;\iota_0\rangle \overset{\kappa_0\mapsto v_0}{\Longrightarrow} \langle N_0;\sigma_0;\iota_1\rangle \overset{O_1}{\Longrightarrow} \langle N_1;\eta_1;\iota_1\rangle \overset{\kappa_1\mapsto v_1}{\Longrightarrow} \dots$$

是 $(t,\ s)$ 的一个类型正确的转换序列。则 $z\mapsto v\in O_{i+1}$ 蕴含 $\kappa_i\in\mathrm{dom}\,(\Delta')$。换言之，只有当 Δ' 中的输入得到更新时，输出通道 z 才会更新。

5 元理论

本节概述第 4.4 节中所述操作性质的证明，即定理4.1、命题 4.4 以及定理 4.5。这三者均可由一个更一般的语义可靠性性质推出。为证明该性质，我们首先以 Kripke 逻辑关系的形式构造 Async RaTT 演算的一个语义模型。也就是说，该模型由一族 $[\![A]\!](w)$ 闭项集合组成，这些集合满足我们所关注的可靠性性质。这族集合由一个世界索引，并通过对类型 A 和世界 w的结构进行归纳法定义。因此，可靠性证明被归约为证明 $\vdash_\Delta t : A$ 蕴含 $t \in [\![A]\!](w)$，这也被称为逻辑关系的基本性质。

5.1 Kripke 逻辑关系

逻辑关系所用的世界 w由两个部分组成：一个自然数 n和一个存储 σ。数字 n 使我们能够通过步数索引 [1]来建模受保护递归类型。具体做法是，对于某个合适的 σ'，将 $[\![\triangleright A]\!](n+1,\ \sigma)$ 定义为 $[\![A]\!](n,\ \sigma')$。由于递归类型 $\mathsf{Fix}\ \alpha.A$ 会展开为 $A[\triangleright(\mathsf{Fix}\,\alpha.A)/\alpha]$，我们可以将 $[\![\mathsf{Fix}\ \alpha.A]\!](n+1,\ \sigma)$ 定义为 $[\![A]\!](n+1,\ \sigma)$ 和 $[\![\mathsf{Fix}\,\alpha.A]\!](n,\ \sigma')$。这是良基的，因为前者引用了更小的类型 A，而后者引用了更小的步数索引 n。

Async RaTT 操作语义的一个关键方面是，它会将延迟计算存储在存储中 σ。因此，为了刻画项 t的语义，我们必须考虑到 t 可能包含指向某个合适存储 σ的堆位置。直观地说，集合 $[\![A]\!](n,\ \sigma)$ 包含这样一些项：从存储 σ出发，它们可以安全地执行并产生类型为 A的值。最终，索引 σ使我们能够证明，反应式语义所执行的垃圾回收确实是健全的。

使 $[\![A]\!](n,\ \sigma)$ 成为 Kripke 逻辑关系的原因在于，我们在世界上定义了一个预序 $\lesssim$，使得 $(n,\ \sigma) \lesssim (n',\ \sigma')$ 蕴含 $[\![A]\!](n,\ \sigma) \subseteq [\![A]\!](n',\ \sigma')$。我们可以将 $(n',\ \sigma')$ 看作从 $(n,\ \sigma)$ 可达的未来世界；也就是说，它描述了机器执行计算时周围上下文如何变化。变化有四种不同类型，下面我们将依次讨论：

首先，时间可能会流逝，这意味着剩余的时间步更少了，即 $n > n'$。其次，机器会对存储 σ 执行垃圾回收。下面的垃圾回收函数对此进行了描述：

$$\mathrm{gc}(\eta_L) = \eta_L \qquad\qquad \mathrm{gc}(\eta_N \langle \kappa \mapsto v \rangle\, \eta_L) = \eta_L$$

第三，机器可能会将延迟计算存储在 σ 中，我们通过堆和存储上的序关系 $\sqsubseteq$ 来对此进行处理：

$$\frac{\eta(l) = \eta'(l) \text{ for all } l \in \mathrm{dom}\,(\eta)}{\eta \sqsubseteq \eta'} \qquad\qquad \frac{\eta_N \sqsubseteq \eta'_N \qquad \eta_L \sqsubseteq \eta'_L}{\eta_N \langle \kappa \mapsto v \rangle\, \eta_L \sqsubseteq \eta'_N \langle \kappa \mapsto v \rangle\, \eta'_L}$$

也就是说，当且仅当 σ' 是通过存储额外的项从 σ 获得时，$\sigma \sqsubseteq \sigma'$。

最后，机器可能会收到一个输入值 $\kappa \mapsto v$，它由存储上的以下序关系$\sqsubseteq^\Delta_\checkmark$ 捕获：

$$\frac{\sigma \sqsubseteq \sigma'}{\sigma \sqsubseteq^\Delta_\checkmark \sigma'} \qquad\qquad \frac{\eta_L \sqsubseteq \eta'_L \qquad \kappa :_c A \in \Delta \qquad \vdash v : A}{\eta_L \sqsubseteq^\Delta_\checkmark \eta_N \langle \kappa \mapsto v \rangle\, \eta'_L}$$

也就是说，除了由 $\sqsubseteq$ 捕获的分配之外，该序关系还可能引入一个输入值$\kappa \mapsto v$。

综上，我们可以如下定义世界上的 Kripke预序 $\lesssim$：

$$(n, \sigma) \lesssim (n', \sigma') \quad \text{iff} \quad n \geq n' \text{ and } \sigma \sqsubseteq^\Delta_\checkmark \sigma'$$

这不包括垃圾回收，因为它仅限于特定情况。事实上，机器只会在执行的某些时刻进行垃圾回收，即在输出转换结束时。

最后，在给出 Kripke 逻辑关系的定义之前，我们需要从语义上刻画输入独立性的概念。无论是select的操作语义，还是信号独立性性质（命题 4.4 和定理 4.5），都需要这一概念。本质上，我们需要确保堆位置 l在世界 $(n+1,\sigma)$ 中仍然存在于未来世界 (n,σ')中，而我们正是在该未来世界里收到了通道 $\kappa \notin l$上的输入。为此，我们令逻辑关系$[\![\Diamond A]\!](n,\sigma)$ 满足以下时钟独立性性质：

$$\text{If } l \in [\![\Diamond A]\!](n,\sigma), \text{ then } l \in [\![\Diamond A]\!](n,[\sigma]_{\mathrm{cl}(l)})$$

其中， $[\sigma]_\Theta$ 将 σ限制为时钟是 Θ的子时钟的堆位置：

$$[\eta]_\Theta(l) = \eta(l) \qquad \text{if } \mathrm{cl}(l) \subseteq \Theta$$

$$[\eta_N\langle\kappa \mapsto v\rangle\,\eta_L]_\Theta = \begin{cases} [\eta_N]_\Theta\langle\kappa \mapsto v\rangle\,[\eta_L]_\Theta & \text{if } \kappa \in \Theta \\ [\eta_L]_\Theta & \text{if } \kappa \notin \Theta \end{cases}$$

Kripke 逻辑关系的完整定义见图 7。除了上述讨论的方面外，该定义还以 Δ 为参数，并区分值关系$\mathcal{V}_\Delta[\![A]\!](w)$ 和项关系 $\mathcal{T}_\Delta[\![A]\!](w)$。这两个关系通过良基递归定义，递归依据是元组$(n,|A|,e)$上的字典序，其中 $|A|$ 是下文定义的 A 的大小；项关系取 $e=1$，值关系取 $e=0$。

$$|\alpha| = |\Diamond A| = |\bigcirc A| = |1| = |\mathsf{Nat}| = 1$$
$$|A \times B| = |A + B| = |A \to B| = 1 + |A| + |B|$$
$$|\Box A| = |\mathsf{Fix}\,\alpha.A| = 1 + |A|$$

注意，在图 7 中 对 $\mathrm{V}_\Delta[\![\Diamond A]\!]$ 的定义中，我们使用简写 $\sigma(l)$ 表示 $\eta_L(l)$，其中η_L 是 σ的后继堆。

我们的目标是通过对类型推导进行归纳，证明基本性质，即 $\vdash_\Delta t : A$ 蕴含 $t \in \mathcal{T}_\Delta[\![A]\!](n,\sigma)$。因此，我们也需要将基本性质推广到开放项。这意味着我们还需要一个相应的上下文逻辑关系，它给在图 7 底部。上下文中 $\checkmark_\theta$ 的解释在技术上相当复杂，但由于要求它是 $\Diamond$ 的左伴随 [7]，其本质上由 $\Diamond$ 的解释决定。

这三个逻辑关系确实满足 Kripke预序下的保持性。

引理 5.1. 设 $n \geq n'$ 和 $\sigma \sqsubseteq^{\Delta}_{\checkmark} \sigma'$。

(i) $\mathrm{V}_\Delta[\![A]\!](n,\sigma) \subseteq \mathrm{V}_\Delta[\![A]\!](n',\sigma')$。

(ii) $\mathrm{T}_\Delta[\![A]\!](n,\sigma) \subseteq \mathrm{T}_\Delta[\![A]\!](n',\sigma')$。

(iii) $\mathrm{C}_\Delta[\![\Gamma]\!](n,\sigma) \subseteq \mathrm{C}_\Delta[\![\Gamma]\!](n',\sigma')$。

保留 然而，在垃圾回收下的保持性仅适用于值和无时钟标记上下文 ：

引理 5.2。（垃圾回收）

(i) $\mathcal{V}_\Delta[\![A]\!](n,\sigma) \subseteq \mathcal{V}_\Delta[\![A]\!](n,\mathrm{gc}(\sigma))$。 *(ii)* $C_\Delta[\![\Gamma]\!]$

$(n,\sigma) \subseteq C_\Delta[\![\Gamma]\!](n,\mathrm{gc}(\sigma))$，前提是 Γ 无时钟标记。

此外，时钟独立性性质对值关系和上下文关系都成立：

引理 5.3。

(i) If $v \in \mathcal{V}_\Delta[\![\Diamond A]\!](n,\sigma)$, *then* $v \in \mathcal{V}_\Delta[\![\Diamond A]\!](n,\sigma')$, *for any* σ' *with* $[\sigma]_{\mathrm{cl}(v)} = [\sigma']_{\mathrm{cl}(v)}$.

$$
\begin{aligned}
\mathrm{V}_\Delta[\![1]\!](w) &= \{\mathsf{hi}\},\\
\mathrm{V}_\Delta[\![\text{自然数}]\!](w) &= \{\text{后继}^n\ 0 \mid n \in \mathbb{N}\},\\
\mathrm{V}_\Delta[\![A \times B]\!](w) &= \{(v_1, v_2) \mid v_1 \in \mathrm{V}_\Delta[\![A]\!](w) \ \wedge\ v_2 \in \mathrm{V}_\Delta[\![B]\!](w)\ \},\\
\mathrm{V}_\Delta[\![A + B]\!](w) &= \{\mathsf{in}_1\, v \mid v \in \mathrm{V}_\Delta[\![A]\!](w)\ \} \cup \{\mathsf{in}_2\, v \mid v \in \mathrm{V}_\Delta[\![B]\!](w)\ \}\\
\mathrm{V}_\Delta[\![A \to B]\!](n,\sigma) &= \mathrm{n}\ \lambda x.t \;\middle|\; \forall \sigma'\, \sqsupseteq^{\Delta}_{\checkmark} \mathsf{gc}(\sigma)\ n' \le n, v \in \mathrm{V}_\Delta[\![A]\!](n'\,\sigma').t[v/x] \in \mathrm{T}_\Delta[\![B]\!](n'\,\sigma')\ \mathrm{o}\\
\mathrm{V}_\Delta[\![\Box A]\!](n,\ \sigma) &= \{\text{盒}\ t \mid t \in \mathrm{T}_\Delta[\![A]\!](n,\ \emptyset)\ \}\\
\mathrm{V}_\Delta[\![\text{Ⓨ}A]\!](0,\ \sigma) &= \{\mathsf{dfix}\ x.t \mid \mathsf{dfix}\ x.t \text{一个闭项}\}\\
\mathrm{V}_\Delta[\![\text{Ⓨ}A]\!](n+1,\ \sigma) &= \{\mathsf{dfix}\ x.t \mid t[\mathsf{dfix}\ x.t/x] \in \mathrm{T}_\Delta[\![A]\!](n,\ \emptyset)\ \}\\
\mathrm{V}_\Delta[\![\text{Ⓖ}A]\!](0,\ \sigma) &= \mathrm{Loc}_\Delta \cup \{\text{等待}_\kappa \mid \kappa :_c A \in \Delta,\ c \in \{\mathsf{p},\ \mathsf{bp}\}\ \}\\
\mathrm{V}_\Delta[\![\text{Ⓖ}A]\!](n+1,\ \sigma) &= \mathrm{n}\ l \in \mathrm{Loc}_\Delta \;\middle|\; \forall \kappa \in \text{闭包}\ (l)\ \vdash v\colon \Delta(\kappa).\sigma(l) \in \mathrm{T}_\Delta[\![A]\!](n,\ \left[\mathsf{gc}(\sigma)\right]_{\kappa \in} \mathsf{h}\kappa \mapsto v \mathsf{i}[\mathsf{gc}(\sigma)]_{\kappa \notin}]_{\text{闭包}(l)})\ \mathrm{o}\\
&\qquad \cup \{\mathsf{wait}_\kappa \mid \kappa :_c A \in \Delta, c \in \{\mathsf{p}, \mathsf{bp}\}\ \}\\
\mathrm{V}_\Delta[\![\text{不动点}\ \alpha.A]\!](w) &= \{\text{进入}\ v \mid v \in \mathrm{V}_\Delta[\![A[\text{Ⓖ}(\text{不动点}\,\alpha.A)/\alpha]]\!](w)\ \}
\end{aligned}
$$

$$
\mathcal{T}_\Delta[\![A]\!](n,\sigma) = \left\{ t \;\middle|\; t \text{ closed}, \forall \iota : \Delta. \forall \sigma' \sqsupseteq^{\Delta}_{\checkmark} \sigma. \exists \sigma'', v.\ \langle t; \sigma' \rangle \Downarrow^{\iota} \langle v; \sigma'' \rangle \wedge v \in \mathcal{V}_\Delta[\![A]\!](n,\sigma'') \right\}
$$

$$
\begin{aligned}
C_\Delta[\![\cdot]\!](w) &= \{*\}\\
C_\Delta[\![\Gamma, x : A]\!](w) &= \left\{ \gamma[x \mapsto v] \;\middle|\; \gamma \in C_\Delta[\![\Gamma]\!](w), v \in \mathcal{V}_\Delta[\![A]\!](w) \right\}\\
C_\Delta[\![\Gamma, \checkmark_\theta]\!](n, \eta_N \langle \kappa \mapsto v \rangle\ \eta_L) &= \left\{ \gamma \in C_\Delta[\![\Gamma]\!](n+1, (\eta'_N, \left[\eta'_L\right]_{\kappa \notin})) \;\middle|\; \begin{array}{l} [\eta_L]_\Theta = \left[\eta'_L\right]_\Theta, [\eta_N]_\Theta = \left[\eta'_N\right]_\Theta,\\ \eta'_N \in \mathsf{Heap}^\kappa, \vdash v : \Delta(\kappa), \kappa \in \Theta, \text{ and}\\ \Theta \subseteq \mathrm{dom_p}\,(\Delta), \text{ where } \Theta = |\theta\gamma| \end{array} \right\}
\end{aligned}
$$

Using

$$
\mathrm{dom_p}\,(\Delta) = \{\kappa \mid \exists A, c \in \{\mathsf{p}, \mathsf{bp}\}. x :_c A \in \Delta\} \qquad \mathrm{Loc}_\Delta = \left\{ l \in \mathrm{Loc} \;\middle|\; \mathrm{cl}\,(l) \subseteq \mathrm{dom_p}\,(\Delta) \right\}
$$

图 7. 逻辑关系。

(ii) If $\gamma \in C_\Delta[\![\Gamma, \checkmark_\theta]\!](n,\sigma)$, *then* $\gamma \in C_\Delta[\![\Gamma, \checkmark_\theta]\!](n, [\sigma]_{|\theta\gamma|})$

最后，我们通过逻辑关系的以下基本性质获得该语言的可靠性：

定理 5.4。 给定 $\Gamma \vdash_\Delta, \Gamma \vdash_\Delta t\colon A$，以及 $\gamma \in \mathrm{C}_\Delta[\![\Gamma]\!](n,\ \sigma)$，则 $t\gamma \in \mathrm{T}_\Delta[\![A]\!](n,\ \sigma)$。

证明是对类型关系 $\Gamma \vdash_\Delta t :\ A$ 进行标准归纳法，利用了上述逻辑关系的闭包性质，证明收录于附录 A。

5.2 操作性质

我们通过展示如何利用基本性质证明第 4.4 节中给出的操作性质，来结束本节。为此，我们将概述定理 4.1、命题 4.4 和定理 4.5 的证明。

5.2.1 生产性。下面我们假设有一个固定的反应式接口 $\Delta \Rightarrow \Gamma_{\text{out}}$，并为反应式语义定义以下机器状态集合 I_n 和 S_n：

$$I_n = \left\{ \langle t; \iota \rangle \,\middle|\, \iota : \Delta \wedge t \in \mathcal{T}_\Delta[\![\text{Prod}\,(\Gamma_{\text{out}})]\!](n, \emptyset) \right\}$$

$$S_n = \left\{ \langle N; \eta; \iota \rangle \,\middle|\, \iota : \Delta \wedge \forall x \mapsto l \in N. \exists x : A \in \Gamma_{\text{out}}. l \in \mathcal{V}_\Delta[\![\textcircled{\exists}(\text{Sig}\,A)]\!](n, \eta) \right\}$$

以下引理证明，机器始终处于上述定义的状态集合之中，并且只会产生类型良好的输出。对于后者，我们利用这样一个事实：$v \in \mathrm{V}_\Delta[\![A]\!](n,\ \sigma)$ 当且仅当 $\vdash v :\ A$ 对于每个值类型 A。

引理 5.5（生产性）。

(i) 若 $t :\ \Delta \Rightarrow \Gamma_{\text{out}}$ 且 $\iota :\ \Delta$，则对所有 $n \in \mathbb{N}$，$\mathit{ht; \iota i} \in I_n$。*(ii)* 若 $\mathit{ht; \iota i} \in I_n$，则存在一个迁移 $\mathit{ht; \iota i} \overset{O}{\Longrightarrow} \mathit{hN; \eta; \iota i}$，使得 $\mathit{hN; \eta; \iota i} \in S_n$ 且 $\vdash O :\ \Gamma_{\text{out}}$。*(iii)* 若 $\mathit{hN; \eta; \iota i} \in S_{n+1}$ 且 $\vdash \kappa \mapsto v :\ \Delta$，则存在两个迁移组成的序列

$$\langle N; \eta; \iota \rangle \overset{\kappa \mapsto v}{\Longrightarrow} \langle N; \sigma; \iota' \rangle \overset{O}{\Longrightarrow} \langle N'; \eta'; \iota' \rangle$$

使得 $\langle N'; \eta'; \iota' \rangle \in S_n$ 且 $\vdash O :\ \Gamma_{\text{out}}$。

证明。

(i) 我们需要证明 $t \in \mathrm{T}_\Delta[\![\text{Prod}\,(\Gamma_{\text{out}})]\!](n, \emptyset)$。由于 $t :\ \Delta \Rightarrow \Gamma_{\text{out}}$，我们知道 $\vdash_\Delta t : \text{Prod}\,(\Gamma_{\text{out}})$。因此，根据定理 5.4，我们有 $t \in \mathrm{T}_\Delta[\![\text{Prod}\,(\Gamma_{\text{out}})]\!](n, \emptyset)$。(ii) 设 $\mathrm{h}t; \iota\mathrm{i} \in I_n$。由此我们有 $t \in \mathrm{T}_\Delta[\![\text{Prod}\,(\Gamma_{\text{out}})]\!](n, \emptyset)$。因此，根据定义，我们有 $\mathrm{h}t; \emptyset\mathrm{i} \Downarrow' \mathrm{h}\mathrm{h}v_1 :: w_1, \ldots, v_k :: w_k\mathrm{i}; \eta\mathrm{i}$，其中 $v_i \in \mathrm{V}_\Delta[\![A_i]\!](n, \eta)$，且 $w_i \in \mathrm{V}_\Delta[\![\textcircled{\exists}(\text{信号 } A_i)]\!](n, \eta)$。由于信号 A_i 不是值类型，我们知道每个 w_i 都必须是堆位置，因此可以将 l_i 写作 w_i。因此，根据定义，$\mathrm{h}t; \iota\mathrm{i} \overset{x_1 \mapsto v_1, \ldots, x_k \mapsto v_k}{\Longrightarrow} \mathrm{h}x_1 \mapsto l_1, \ldots, x_k \mapsto l_k; \eta; \iota\mathrm{i}$ 和 $\mathrm{h}x_1 \mapsto l_1, \ldots, x_k \mapsto l_k; \eta; \iota\mathrm{i} \in S_n$。由于每个 A_i 都是值类型，我们知道 $v_i \in \mathrm{V}_\Delta[\![A_i]\!](n, \eta)$ 意味着 $\vdash v_i :\ A_i$，因而 $\vdash x_1 \mapsto v_1, \ldots, x_k \mapsto v_k :\ \Gamma_{\text{out}}$。

(iii) 根据定义，$\mathrm{h}N; \eta; \iota\mathrm{i} \overset{\kappa \mapsto v}{\Longrightarrow} \mathrm{h}N; \sigma; \iota'\mathrm{i}$，其中 $\sigma = [\eta]_{\kappa \in} \mathrm{h}\kappa \mapsto v\mathrm{i}\ [\eta]_{\kappa \notin}$。我们断言以下三个陈述成立：

$$\text{If } y : B \in \Gamma_{\text{out}}, N(y) = l, \kappa \in \text{cl}\,(l)\text{, then adv}\, l \in \mathcal{T}_\Delta[\![\text{Sig}\,B]\!](n, \sigma) \tag{2}$$

$$\text{If } y : B \in \Gamma_{\text{out}}, N(y) = l, \kappa \notin \text{cl}\,(l)\text{, then } l \in \mathcal{V}_\Delta[\![\textcircled{\exists}(\text{Sig}\,B)]\!](n, \sigma) \tag{3}$$

$$\text{If } \sigma' \sqsupseteq \sigma, N_1 \subseteq N, \text{ then } \langle N_1; \sigma'; \iota' \rangle \overset{O}{\Longrightarrow} \langle N_2; \eta'; \iota' \rangle, \langle N_2; \eta'; \iota' \rangle \in S_n, \text{gc}(\sigma') \sqsubseteq \eta', \vdash O : \Gamma_{\text{out}} \tag{4}$$

由（4），可得 $\mathrm{h}N; \sigma; \iota'\mathrm{i} \overset{O}{\Longrightarrow} \mathrm{h}N'; \eta'; \iota'\mathrm{i}$，其中 $\mathrm{h}N'; \eta'; \iota'\mathrm{i} \in S_n$ 且 $\vdash O :\ \Gamma_{\text{out}}$。我们通过证明上述三个断言来完成证明：

- （2）的证明。由于 $\langle N; \eta; \iota \rangle \in S_{n+1}$，我们知道 $l \in \mathcal{V}_\Delta[\![\textcircled{\exists}(\text{信号 } B)]\!](n+1, \eta)$，因此 $\text{adv}\, l \in \mathcal{T}_\Delta[\![\text{信号 } B]\!](n, \sigma)$。
- （3）的证明。由于 $\mathrm{h}N; \eta; \iota\mathrm{i} \in S_{n+1}$，我们知道 $l \in \mathrm{V}_\Delta[\![\textcircled{\exists}(\text{信号 } B)]\!](n+1, \eta)$。由于 $\kappa \notin$ 闭包 (l)，我们可以得出 $[\eta]$闭包$(\iota) = [[\eta]_{\kappa \notin}]$闭包$(\iota)$。进而，根据引理 5.3，这意味着 $l \in \mathrm{V}_\Delta[\![\textcircled{\exists}(\text{信号 } B)]\!](n+1, [\eta]_{\kappa \notin})$，而根据引理 5.1，这又意味着 $l \in \mathrm{V}_\Delta[\![\textcircled{\exists}(\text{信号 } B)]\!](n, \sigma)$。
- （4）的证明。我们对 N_1 进行归纳。情形 $N_1 = \cdot$ 是显然的。对于情形 $N_1 = y \mapsto w, N'1$，我们区分 $\kappa \in \text{cl}\,(w)$ 和 $\kappa \notin \text{cl}\,(w)$ 两种情形。然后我们可以分别应用（2）或（3），并对 $N'1$ 和某个 $\sigma'' \sqsupseteq \sigma'$ 使用归纳假设。若 $\kappa \notin \text{cl}\,(w)$，则 σ'' 就是 σ'。若 $\kappa \in \text{cl}\,(w)$，则 σ'' 源自我们通过（2）得到的求值 $\mathrm{h}\text{adv}\, w; \sigma'\mathrm{i} \Downarrow'' \mathrm{h}v :: w'; \sigma''\mathrm{i}$。无论哪种情形，我们都利用引理 5.1 和引理 5.2 得出 $\mathrm{h}N_2; \eta'; \iota'\mathrm{i} \in S_n$。□

Th生产性性质现在是上述引理的直接推论：

定理 4.1 （生产性）的证明。 对于每个 $n \in \mathbb{N}$ 我们都可以构造如下有限转换序列 s_n 利用引理 5.5：

$$\langle t; \iota_0 \rangle \overset{O_0}{\Longrightarrow} \langle N_0; \eta_0; \iota_0 \rangle \overset{\kappa_0 \mapsto v_0}{\Longrightarrow} \langle N_0; \sigma_0; \iota_1 \rangle \overset{O_1}{\Longrightarrow} \langle N_1; \eta_1; \iota_1 \rangle \overset{\kappa_1 \mapsto v_1}{\Longrightarrow} \dots \overset{O_n}{\Longrightarrow} \langle N_n; \eta_n; \iota_n \rangle$$

满足 $\vdash O_i : \Gamma_{\text{out}}$ 对所有 $0 \le i \le n$。由引理 4.2， s_n 是 s_m 对所有 $m > n$的前缀。因此，我们可将所有 s_n的极限作为所需的无限转换序列。 □

5.2.2 信号独立性。 命题 4.4 是引理 5.5的直接推论：

命题 4.4（缓冲信号独立性）的证明。由引理 5.5，对于从 $\mathrm{h}t; \iota_i\mathrm{i}$ 开始的序列中的任意一对迁移 $\mathrm{h}N_i; \eta_i; \iota'\mathrm{i} \overset{\kappa_i \mapsto v_i}{\Longrightarrow} \mathrm{h}N_i; \sigma_i; \iota_{i+1}\mathrm{i} \overset{O_{i+1}}{\Longrightarrow} \mathrm{h}N_{i+1}; \eta_{i+1}; \iota_{i+1}\mathrm{i}$，我们有 $\mathrm{h}N_i; \eta_i; \iota'\mathrm{i} \in S_1$。特别地，这意味着，对于任意满足 $x\colon A \in \Gamma_{\text{out}}$ 的 $x \mapsto l \in N_i$， $l \in \mathrm{V}_\Delta[\![\Im(\text{信号 } A)]\!](1, \eta_i)$，进而意味着 $l \in \mathrm{Loc}_\Delta$。因此， $\kappa_i \notin$ 闭包 (l)，所以 O_{i+1} 为空。 □

为了证明定理 4.5，我们首先在偏映射 Δ' 将变量 x 映射到输入上下文 Δ'_x的情形下，定义以下机器状态集合：

$$T_n^{\Delta'} = \left\{ \langle N; \eta; \iota \rangle \,\middle|\, \iota : \Delta \wedge \forall x \mapsto w \in N.\Delta'_x \subseteq \Delta \wedge \exists x : A \in \Gamma_{\text{out}}.w \in \mathcal{V}_{\Delta'_x}[\![\Im(\mathrm{Sig}\, A)]\!](n, \eta) \right\}$$

在$T_n^{\Delta'}$ 中的机器状态会在机器执行过程中保持不变：

引理 5.6。 若 $\mathrm{h}N; \eta; \iota\mathrm{i} \in T_{n+1}^{\Delta'}$ 且 $\mathrm{h}N; \eta; \iota\mathrm{i} \overset{\kappa \mapsto v}{\Longrightarrow} \mathrm{h}N; \sigma; \iota'\mathrm{i} \overset{O}{\Longrightarrow} \mathrm{h}N'; \eta'; \iota'\mathrm{i}$，则 $\mathrm{h}N'; \eta'; \iota'\mathrm{i} \in T_n^{\Delta'}$。

证明。注意， $\sigma = [\eta]_{\kappa \in} \mathrm{h}\kappa \mapsto v\mathrm{i}\ [\eta]_{\kappa \notin}$ ，且 $[\eta]_{\kappa \notin} \subseteq \eta'$。假设$(x \mapsto w) \in N$，并注意到 w必定是一个位置，因为 $\mathrm{Sig}\, A$ 不是值类型。我们将 l记作 w 以强调这一点。于是存在一个 l'， 使得$(x \mapsto l') \in N'$， 我们需要利用假设 $l \in \mathrm{V}_{\Delta'_x}[\![\Im(\mathrm{Sig}\ A)]\!](n+1, \eta)$证明 $l' \in \mathrm{V}_{\Delta'_x}[\![\Im(\mathrm{Sig}\ A)]\!](n, \eta')$。先假设 $\kappa \notin \mathrm{cl}\,(l)$。那么 $l' = l$， 因此

$$l' \in \mathcal{V}_{\Delta'_x}[\![\Im(\mathrm{Sig}\, A)]\!](n, [\eta]_{\mathrm{cl}(l)}) \subseteq \mathcal{V}_{\Delta'_x}[\![\Im(\mathrm{Sig}\, A)]\!](n, [\eta]_{\kappa \notin}) \subseteq \mathcal{V}_{\Delta'_x}[\![\Im(\mathrm{Sig}\, A)]\!](n, \eta')$$

现在假设 $\kappa \in \mathrm{cl}\,(l)$。在这种情况下，通过求值 $\mathrm{h}\sigma'(l); \sigma'\mathrm{i} \Downarrow \mathrm{h}w :: l'; \sigma''\mathrm{i}$，必定发生了 l'，其中 σ'， σ'' 满足 $\sigma \subseteq \sigma'$，且 $\mathrm{gc}(\sigma'') \subseteq \eta'$。该假设告诉我们

$$\eta(l) \in \mathcal{T}_{\Delta'_x}[\![\Im(\mathrm{Sig}\, A)]\!](n, [\eta]_{\kappa \in} \langle \kappa \mapsto v \rangle\, [\eta]_{\kappa \notin})$$

由于 $[\eta]_{\kappa \in} \mathrm{h}\kappa \mapsto v\mathrm{i}\ [\eta]_{\kappa \notin} \subseteq \sigma'$，这意味着 $\mathrm{h}\eta(l); \sigma'\mathrm{i} \Downarrow \mathrm{h}w' :: l''; \sigma'''\mathrm{i}$。由于 $\eta(l) = [\eta]_{\kappa \in}(l) = \sigma'(l)$，根据操作语义的确定性（引理 4.2）， $\sigma''' = \sigma''$ 且 $l'' = l'$。由此我们得出

$$l' \in \mathcal{V}_{\Delta'_x}[\![\Im(\mathrm{Sig}\, A)]\!](n, \sigma'') = \mathcal{V}_{\Delta'_x}[\![\Im(\mathrm{Sig}\, A)]\!](n, \mathrm{gc}(\sigma'')) \subseteq \mathcal{V}_{\Delta'_x}[\![\Im(\mathrm{Sig}\, A)]\!](n, \eta') \qquad \square$$

定理的证明 4.5 （推送信号独立性）。令 $\Gamma_{\text{out}} = x_1 : B_1, \dots, x_m : B_m, z : C$。初始化转换

$$\langle (t, s); \iota_0 \rangle \overset{x_1 \mapsto v_1, \dots, x_m \mapsto v_m, z \mapsto w}{\Longrightarrow} \langle x_1 \mapsto l_1, \dots, x_m \mapsto l_m, z \mapsto l; \eta_0; \iota_0 \rangle = \langle N_0; \eta_0; \iota_0 \rangle$$

由以下形式的求值导致：$\mathrm{h}t; \emptyset\mathrm{i} \Downarrow \mathrm{h}\mathrm{h}v_1 :: l_1, \dots, v_m :: l_m\mathrm{i}; \eta\mathrm{i}$，以及 $\mathrm{h}s; \eta\mathrm{i} \Downarrow \mathrm{h}w :: l; \eta_0\mathrm{i}$。根据假设， $\vdash_{\Delta'} s$: 信号 C，因此根据定理 5.4，对所有 n，都有 $s \in \mathrm{T}_{\Delta'}[\![\text{ 信号 } C]\!](n, \emptyset)$，进而推出对所有 n，都有 $l \in \mathrm{V}_{\Delta'}[\![\Im$ 信号$C]\!](n, \eta_0)$。同样，对所有 n，都有 $l_j \in \mathrm{V}_\Delta[\![\Im$信号 $B_j]\!](n, \eta_0)$，且 $j = 1, \dots, m$。因此，对所有 n，都有 $\mathrm{h}x_1 \mapsto l_1, \dots, x_m \mapsto l_m, z \mapsto l; \eta_0; \iota_0\mathrm{i} \in T_n^{\Delta''}$，其中 $\Delta''_z = \Delta'$ 且 $\Delta''_{x_j} = \Delta$，对于 $j = 1, \dots, m$。由引理 5.6 的 n 次应用，我们由此得到 $\mathrm{h}N_n; \eta_n; \iota_n\mathrm{i} \in T_1^{\Delta''}$。特别地，$N_n(z) = l'$，其中某个 $l' \in \mathrm{V}_{\Delta'}[\![\Im($ 信号 $C)]\!](1, \eta_n)$。因此，$\mathrm{cl}\,(l') \subseteq \mathrm{dom}\,(\Delta')$，于是 $z \mapsto v \in O_{i+1}$ 推出 $\kappa_i \in \mathrm{cl}\,(l')$，进而推出 $\kappa_i \in \mathrm{dom}\,(\Delta')$。 □

6 相关工作

函数式响应式编程起源于 Elliott 和 Hudak [12]。Krishnaswami 和 Benton 首先提出将模态类型用于函数式响应式编程 [20],，而线性时序逻辑与函数式响应式编程之间的联系则由 Jeffrey [16] 和 Jeltsch [18]独立发现。尽管其中一些演算系统已经实现，但由于缺乏防止空间泄漏的能力，它们并未提供本文所证明的那类操作性保证。Krishnaswami 等人 [21] 最先给出了此类操作性保证，他们描述了一种模态函数式响应式编程语言，利用线性类型和分配资源，在静态层面限制响应式程序所使用的内存。另一种更简单但不那么精确的思路，是采用激进的垃圾回收技术来避免空间泄漏；这一思路由 Krishnaswami 提出 [19]。Krishnaswami 的演算采用双上下文方法来处理模态类型编程。Bahr 等人 [3] 将这些结果改写为 Fitch 风格模态演算，这是 RaTT 系列中的第一个演算。之后，该演算经过少量修改后在 Haskell 中实现 [2]。

上述所有演算系统都基于全局时间概念，而在几乎所有情况下，时间都是离散的。特别地，这些演算系统中用于时间步的模态算子 $\bigcirc$ 指的是全局时钟上的下一个时间步。当然，也可以将 Async RaTT 的步语义理解为在全局时钟上运行，但在我们的模型中，每一步都与来自某个输入通道的输入相关联，这使我们能够将延迟模态 $\Circled{\exists}$ 定义为对一组输入通道的延迟。从模型的角度来看，$\Circled{\exists} A$ 与类型 $\bigcirc(\diamond A)$ 有一些相似之处，其中 $\diamond A \cong A + \bigcirc\diamond A$ 是一个受保护递归类型。然而，这种编码存在引言中提到的效率和抽象问题。

据我们所知，唯一的异步模态函数式响应式编程演算是 Graulund 等人定义的 λ_{widget} [14],，它将 $\diamond$ 作为类型构造子原语，并为其赋予了类似于 Async RaTT 中 select 的同步原语。然而，λ_{widget} 中的编程原语与此处使用的原语非常不同。例如，λ_{widget} 允许将 $\diamond A$ 的一个元素分解为一个时间以及该时间上的一个 A 元素，并且许多对 $\diamond$ 的编程都使用这种分解。它也没有延迟类型构造子 $\bigcirc$，因此无法表达 $\Circled{\exists}$：不同于 $\Circled{\exists} A$，$\diamond A$ 的一个元素现在就可能给出一个类型为 A 的值。Graulund 等人为 λ_{widget} 提供了指称语义，但没有提供操作语义，也没有提供如本文所证明的操作性保证。

另一种避免空间泄漏和非因果响应式程序的方法，是精心设计一个用于操作信号的接口，例如 Yampa [23] 或 FRPNow! [24]。Rhine [5] 是对 Yampa 的近期改进，它为信号函数标注类型级时钟，从而能够构建复杂的数据流图，将以不同速度运行的子系统组合起来。类型规则会在编译时静态确定每个子系统的时钟，因为 Rhine 的目标是在子系统之间实现高效重采样。相比之下，Async RaTT 的类型级时钟是存在量化的，因此 Async RaTT 程序可以动态地改变信号的时钟，例如使用第 *3.2* 节中的组合子 switch。

Elliott [13] 提出了一种 FRP 的推拉式实现，其中信号（按照经典函数式响应式编程的传统，[12] 称为行为）会在离散时间步更新（推），但也可以在这些更新之间的任意时刻采样（拉）。我们可以在 Async RaTT 中使用类型 $\mathsf{Sig}(\mathsf{Time} \to A)$ 表示此类推拉式信号；也就是说，在时钟的每次滴答时，我们都会获得一个新的函数 $\mathsf{Time} \to A$，它描述了信号在随后的时钟滴答到来之前随时间变化的值。

期值最初在 MuliLisp [15] 中实现，如今常见于许多编程语言中，只是名称各异（如承诺、异步/等待、延迟等），它们提供了一种强大的抽象，便于并发计算之间进行通信。类型为 Future A 的值，表示承诺在未来某个时间交付类型为 A 的值。例如，一个读取文件内容的函数可以立即返回类型为 Future缓冲区 的值，而不是

阻塞调用方，直到文件被读取到缓冲区中。Async RaTT 可以通过类型模态 ◁提供类似的接口，既可以直接实现，也可以将 Future 定义为受保护递归类型 Future $A \cong A + ◁(\text{Future } A)$，从而赋予 Future 单子接口。由于 Async RaTT 不要求只推送通道的集合是有限的，我们可以将一个接收文件名 f 并返回 FutureBuffer 类型结果的函数，简单地实现为一组通道：readFile$f$:ℙ 缓冲区。机器会监视包含这些通道的时钟所对应的延迟计算，并行启动对相应文件的读取，并在文件读取过程完成后，通过通道提供类型为 缓冲区 的值。

如前所述，Krishnaswami 等人 [21] 采用线性类型系统来获得静态内存界。除了这类内存界，同步（数据流）语言（例如 Esterel [6], Lustre [8], 和 Lucid Synchrone [26] 甚至还能提供运行时间界限。尽管有这些强保证，Lucid Synchrone 仍支持高阶、模块化的编程风格，并支持高阶函数。然而，要实现这些静态保证，同步数据流语言必然要严格限制动态行为，禁止任意类型的值随时间变化（例如，我们不能拥有由流构成的流），也禁止动态切换（即，不具备与 *switch* 组合子等效的功能）。Lustre 和 Lucid Synchrone 都有时钟的概念，它本质上是一个布尔值流，用于指示在全局时钟的每个时钟滴答时，本地时钟是否也发生滴答。

7 结论与未来工作

本文介绍了 Async RaTT，这是首个具有操作性保证的异步 FRP 模态语言。我们展示了如何使用新的异步延迟模态类型 ◁，为运行时系统标注从输出通道到输入通道的依赖关系，从而确保仅在必要时重新计算输出。积分和导数的示例还展示了程序员如何主动影响输出通道的更新频率。

选择 Fitch 风格模态纯属个人偏好，我们相信这些结果也可以在对偶上下文语言中复现。尽管 Fitch 风格对上下文使用了非标准操作，RaTT 系列中的其他语言已作为 Haskell [2] 库实现。因此，我们也相信 Async RaTT 可以在 Haskell 或其他函数式编程语言中实现，让程序员能够使用 RaTT 与宿主编程语言相结合的特性。

Async RaTT 尚缺少的一个方面是输出通道过滤。例如，无法编写一个仅在满足某个输入条件时才产生输出的过滤函数。实现这一点的最佳方式是使用类型为 Maybe(A) 的输出通道，并由运行时系统负责仅向该输出通道的使用者推送类型为 A 的值。这样一来，过滤就由编程语言之外的机制完成。我们认为，无法对 Async RaTT 的运行时模型进行有意义的扩展以将其纳入语言内部。

致谢

Møgelberg w 获得丹麦独立研究基金资助，项目编号为 2032-00134B .

参考文献

[1] Andrew W. Appel 和 David McAllester。2001。用于基础证明携带代码的递归类型索引模型。《*ACM*编程语言系统汇刊》23, 5（2001 年 9 月），657–683。https://doi.org/10.1145/504709.504712 00283。[2] Patrick Bahr。2022。人人皆可用的模态函数式响应式编程：在 Haskell 中实现无空间泄漏的函数式响应式编程。《函数式编程杂志》32（2022），e15。[3] Patrick Bahr、Christian Uldal Graulund 和 Rasmus Ejlers Møgelberg。2019。Simply RaTT：一种用于无空间泄漏响应式编程的 Fitch 风格模态演算。《*ACM*编程语言会议录》3, ICFP（2019），1–27。https://doi.org/10.1145/3341713

[4] Patrick Bahr、Christian Uldal Graulund 和 Rasmus Ejlers Møgelberg。2021。钻石并非永恒：受保护递归响应式编程中的活性。《ACM编程语言会议录》5, POPL（2021），1–28。[5] Manuel Bärenz 和 Ivan Perez。2018。Rhine：带有类型级时钟的函数式响应式编程。载于第 11 届 ACM SIGPLAN Haskell 国际研讨会论文集（Haskell 2018）。计算机协会，美国纽约州纽约市，145– 157 10 1145 3242744 3242757 。https://doi.org/ . / . Gérard Berry 和 Laurent Cosserat。 . ESTEREL 同步[6]编程语言及其数学语义。载于《并发1985研讨会》，Stephen D. Brookes、Andrew William Roscoe 和 Glynn Winskel（编）。Springer Berlin Heidelberg，德国海德堡柏林，389–448。[7] Lars Birkedal、Ranald Clouston、Bassel Mannaa、Rasmus Ejlers Møgelberg、Andrew M Pitts 和 Bas Spitters。2020。模态依赖类型理论与依赖右伴随。《数学结构与计算机科学》30, 2（2020），118–138。[8] Paul Caspi、Daniel Pilaud、Nicolas Halbwachs 和 John A. Plaice。1987。LUSTRE：一种用于实时编程的声明式语言。载于第 14 届 ACM SIGACT-SIGPLAN 编程语言原理研讨会论文集（POPL ’ *87* 178 188 10 1145 41625 41641 ）。ACM，美国纽约州纽约市，– 。[9]https://doi.org/ . / . Andrew Cave、Francisco Ferreira、Prakash Panangaden 和 Brigitte Pientka。 . 公平响应式编程。载于第 41 届 ACM SIGPLAN-SIGACT 编程语言原理研讨会论文集（POPL ’14）。ACM，美国加利福尼亚州圣迭戈， 361 372 10 1145 2535838 2535881 – 。https://doi.org/ . / . Ranald Clouston。[10] . Fitch 风格模态 λ 演算。载于《软件科学与计算结构基础》，Christel Baier 和 Ugo Dal Lago（编），第 10803 卷。Springer，Springer International Publishing，瑞士 Cham，258–275。https://doi.org/ 10 1007 978 3 319 89366 2_14 . / - - - - Rowan Davies 和 Frank Pfenning。 . 分阶段计算的模态分析。《[11]ACM杂志》（JACM）48, 3（2001），555–604。[12] Conal Elliott 和 Paul Hudak。1997。函数式响应式动画。载于第二届 ACM SIGPLAN 函数式编程国际会议论文集（ICFP ’97）。ACM，美国纽约州纽约市，263–273。https://doi.org/ 10 1145 258948 258973 . / . Conal M. Elliott。 . 推拉式函数式响应式编程。载于第 2 届 ACM SIGPLAN [13]Haskell 研讨会论文集（Haskell ’ *09* 25 36 10 1145 1596638 1596643 ）。ACM，美国纽约州纽约市，– 。[14]https://doi.org/ . / . Christian Uldal Graulund、Dmitrij Szamozvancev 和 Neel Krishnaswami。 . 伴随响应式 GUI 编程。。载于 FoSSaCS。289–309。[15] Robert H. Halstead。1985。Multilisp：一种用于并发符号计算的语言。《ACM编程语言系统汇刊》 7, 4 1985 501 538 10 1145 4472 4478 （10 月），– 。https://doi.org/ . / . Alan Jeffrey。[16] . LTL 类型 FRP：将线性时间时序逻辑命题作为类型、证明作为函数式响应式程序。载于第六届“编程语言与程序验证”研讨会论文集（PLPV 2012），美国宾夕法尼亚州费城，2012 年 1 月 24 日，Koen Claessen 和 Nikhil Swamy（编）。ACM，美国宾夕法尼亚州费城，49–60。https://doi.org/ 10 1145 2103776 2103783 . / . Alan Jeffrey。 . 函数式响应式类型。载于第二十三届 EACSL 计算机科学逻辑年会（CSL）与第二十九届 ACM/IEEE 逻辑与计算机科学研讨会（LICS）联合会议论文集（CSL-LICS ’ *14* 54, 9 10 1145 2603088 2603106 ）。ACM，[17]美国纽约州纽约市，文章页码。[18]https://doi.org/ . / . Wolfgang Jeltsch。 . 迈向线性时间时序逻辑与函数式响应式编程的统一范畴语义。《理论计算机科学电子札记》286（2012），229–242。https://doi.org/ 10 1016 2012 08 015 . /j.entcs. . . Neelakantan R. Krishnaswami。 . 无时空泄漏的高阶函数式响应式编程。载于第 18 届 ACM SIGPLAN 函数式编程国际会议论文集（ICFP ’13）。ACM，美国马萨诸塞州波士顿，[19] 221 232 10 1145 2500365 2500588 – 。https://doi.org/ . / . Neelakantan R. Krishnaswami 和 Nick Benton。 . 响应式程序的超度量语义。载于 2011 年 IEEE 第 26 届逻辑与计算机科学年会。IEEE计算机学会，美国华盛顿州华盛顿，257–266。https://doi.org/ 10 1109 2011 38 . /LICS. . Neelakantan R. Krishnaswami、Nick Benton 和 Jan Hoffmann。2012。[20]有界空间中的高阶函数式响应式编程。载于第 39 届 ACM SIGPLAN-SIGACT 编程语言原理研讨会论文集（POPL 2012），美国宾夕法尼亚州费城，2012 年 1 月 22–28 日，John Field 和 Michael Hicks（编）。ACM，美国宾夕法尼亚州费城， 45 58 10 1145 2103656 2103665 – 。https://doi.org/ . / . Hiroshi Nakano。 . 一种递归模态。载于第十五届 IEEE 逻辑与计算机科学年会论文集（目录号 99CB36332）。IEEE计算机学会，美国华盛顿州华盛顿，[22]255–266。https://doi.org/ 10 1109 2000 855774 . /LICS. . Henrik Nilsson、Antony Courtney 和 John Peterson。2002。函数式响应式编程续篇。载于 Haskell ’02：2002 年 ACM SIGPLAN Haskell 研讨会论文集。ACM，美国纽约州纽约市，51–64。[23]

https://doi.org/10.1145/581690.581695

[24] Atze van der Ploeg 和 Koen Claessen。2015。实用的原则性函数式响应式编程：忘记过去，改变未来，FRPNow!。载于 第 *20* 届 *ACM SIGPLAN* 函数式编程国际会议论文集（*ICFP 2015*）。计算机协会，温哥华，不列颠哥伦比亚省，加拿大，第 302–314 页。 https://doi.org/10.1145/2784731.2784752 00019。 [25] Amir Pnueli。1977。《程序的时序逻辑》。载于 第 *18* 届计算机科学基础年度研讨会论文集（*SFCS ’* *77* 46 57 10 1109 1977 32 ）。IEEE计算机学会，美国，第 – 页。 [26] 2006 https://doi.org/./SFCS..Marc Pouzet。 . Lucid 3 Synchrone，版本 . 教程与参考手册。巴黎南大学，*LRI* 1（2006），25。

基本性质的证明

给定一个堆 η，我们使用以下记法来构造一个良构存储，其中 $\mathrm{h}\kappa \mapsto v\mathrm{i}$ 如下：

$$\mathrm{tick}_{\kappa\mapsto v}(\eta) = [\eta]_{\kappa\in} \langle \kappa \mapsto v \rangle [\eta]_{\kappa\notin}$$

引理 A.1（机器单调性）。若 $\mathrm{h}t;\ \sigma\mathrm{i} \Downarrow' \mathrm{h}v;\sigma'\mathrm{i}$，则 $\sigma \sqsubseteq \sigma'$。

证明。对 $\mathrm{h}t;\ \sigma\mathrm{i} \Downarrow' \mathrm{h}v;\sigma'\mathrm{i}$ 直接使用归纳法即可。 □

引理 A.2。

(i) 若 $\sigma \sqsubseteq^{\Delta}_{\surd} \sigma'$，则 $\mathrm{gc}(\sigma) \sqsubseteq \mathrm{gc}(\sigma')$。*(ii)* $\mathrm{gc}(\sigma) \sqsubseteq^{\Delta}_{\surd} \sigma$。*(iii)* 若 $\eta \sqsubseteq \eta'$，则 时钟滴答$_{\kappa\mapsto v}(\eta) \sqsubseteq$ 时钟滴答$_{\kappa\mapsto v}(\eta')$。

证明。通过直接的情形分析。 □

引理 A.3。

(i) $[\text{时钟滴答}_{\kappa\mapsto v}(\eta)]_{\Theta} = \text{时钟滴答}_{\kappa\mapsto v}([\eta]_{\Theta})$。 *(ii)* $[\mathrm{gc}(\sigma)]_{\Theta} = \mathrm{gc}([\sigma]_{\Theta})$。

证明。通过直接的情形分析。 □

引理5.1。设 $n \geq n'$，以及 $\sigma \sqsubseteq^{\Delta}_{\surd} \sigma'$。

(i) $\mathrm{V}_{\Delta}[\![A]\!](n,\ \sigma) \subseteq \mathrm{V}_{\Delta}[\![A]\!](n',\ \sigma')$。

(ii) $\mathrm{T}_{\Delta}[\![A]\!](n,\ \sigma) \subseteq \mathrm{T}_{\Delta}[\![A]\!](n',\ \sigma')$。

(iii) $\mathrm{C}_{\Delta}[\![\Gamma]\!](n,\ \sigma) \subseteq \mathrm{C}_{\Delta}[\![\Gamma]\!](n',\ \sigma')$。

引理的证明 5.1。 (i) 和 (ii) 通过良基归纳法证明，所用的良基序与我们用于论证两种逻辑关系定义良好的良基序相同。 (iii) 通过对 Γ 的长度进行归纳，并利用 (i) 得证。 □

引理 5.2。

(i) $\mathcal{V}_{\Delta}[\![A]\!](n, \sigma) \subseteq \mathcal{V}_{\Delta}[\![A]\!](n, \mathrm{gc}(\sigma))$。*(ii)* $\mathcal{C}_{\Delta}[\![\Gamma]\!](n, \sigma) \subseteq \mathcal{C}_{\Delta}[\![\Gamma]\!](n, \mathrm{gc}(\sigma))$，如果 Γ 是无*tick*的。

引理 5.2 的证明。 两项分别通过对大小 $|A|$ 和长度 Γ 进行归纳证明。 □

引理 5.3。

(i) 若 $v \in \mathcal{V}_{\Delta}[\![\Box A]\!](n, \sigma)$，则对于任意满足 $[\sigma]_{\mathrm{cl}(v)} = [\sigma']_{\mathrm{cl}(v)}$ 的 σ'，有 $v \in \mathcal{V}_{\Delta}[\![\Box A]\!](n, \sigma')$。*(ii)* 若 $\gamma \in \mathcal{C}_{\Delta}[\![\Gamma, \surd_{\theta}]\!](n, \sigma)$ 且 $\Theta = |\theta\gamma|$，则 $\gamma \in \mathcal{C}_{\Delta}[\![\Gamma, \surd_{\theta}]\!](n, [\sigma]_{\Theta})$

引理 5.3的证明。 两项均可通过检查 $\mathcal{V}_{\Delta}[\![\Box A]\!](n,\sigma)$和 $\mathcal{C}_{\Delta}[\![\Gamma, \surd_{\theta}]\!](n,\sigma)$的定义分别得证。 □

事实是，(i) 对 $\mathrm{V}_{\Delta}[\![\Box A]\!](n, \sigma)$ 成立，但对 $\mathrm{T}_{\Delta}[\![\Box A]\!](n, \sigma)$ 不成立；这正是我们需要限制演算、规定 adv 和 select 只能应用于值的原因。

引理 A.4。$\mathcal{V}_{\Delta}[\![A]\!](n, \sigma) = \mathcal{V}_{\Delta}[\![A]\!](n, \emptyset)$ 对于任意稳定类型 A。

证明。对 A的大小进行直接归纳即可。 □

引理 A.5。设 $\gamma \in \mathcal{C}_{\Delta}[\![\Gamma, \Gamma']\!](n, \sigma)$满足 Γ' 无*tick*。那么 $\gamma|_{\Gamma} \in \mathcal{C}_{\Delta}[\![\Gamma]\!](n, \sigma)$。

证明。对 Γ' 的长度进行直接归纳法。 □

引理 A.6。如果 $\gamma \in C_\Delta[\![\Gamma]\!](n,\ \sigma)$，那么 $\gamma|_{\Gamma^\square} \in C_\Delta[\![\Gamma^\square]\!](n,\ \emptyset)$。

证明。 对 Γ 的长度作直接归纳，并使用引理 A.4。 □

引理 A.7。 若 $t \in \mathcal{V}_\Delta[\![A]\!](w)$，则 t 是一个值。

证明。通过检查定义可知。 □

引理 A.8。 $\mathcal{V}_\Delta[\![A]\!](w) \subseteq \mathcal{T}_\Delta[\![A]\!](w)$。

证明。设 $t \in \mathrm{V}_\Delta[\![A]\!](n,\sigma)$，且 $\sigma' \sqsupseteq^\Delta_{\checkmark} \sigma$ 和 $\iota :\ \Delta$。由引理 A.7，t 是一个值，因此 $\mathrm{h}t;\sigma'\mathrm{i} \Downarrow \mathrm{h}t;\sigma'\mathrm{i}$。由引理 5.1 可得 $t \in \mathrm{V}_\Delta[\![A]\!](n,\sigma')$，进而由引理 5.1 推出 $t \in \mathrm{V}_\Delta[\![A]\!](n,\sigma')$。 □

引理 A.9. 如果 v 是一个值，且 $v \in \mathcal{T}_\Delta[\![A]\!](w)$，那么 $v \in \mathcal{V}_\Delta[\![A]\!](w)$。

证明。令 $v \in \mathrm{T}_\Delta[\![A]\!](n,\sigma)$，并任取一个 $\iota :\ \Delta$。由于 $\mathrm{h}v;\sigma\mathrm{i} \Downarrow \mathrm{h}v;\sigma\mathrm{i}$，根据定义可得 $v \in \mathrm{V}_\Delta[\![A]\!](n,\sigma)$。 □

引理 A.10. 如果 $\Gamma \vdash_\Delta \theta : \text{时钟}$ 且 $\gamma \in C_\Delta[\![\Gamma]\!](w)$，那么 $\theta\gamma$ 是一个闭合的时钟表达式，且 $|\theta\gamma| \subseteq \mathrm{dom}_\mathrm{p}(\Delta)$。

证明。我们对 $\Gamma \vdash_\Delta \theta : \mathsf{Clock}$.$\Gamma \vdash_\Delta \theta : \mathsf{Clock}$ $\Gamma \vdash_\Delta \theta' : \mathsf{Clock}$ 进行归纳。

- $\overline{\Gamma \vdash_\Delta \theta \sqcup \theta\prime : \mathsf{Clock}}$ 根据归纳法，$\theta\gamma$ 和 $\theta\prime\gamma$ 都是闭合的，且 $|\theta\gamma| \subseteq \mathrm{dom}(\Delta)$。因此，$(\theta \sqcup \theta\prime)\gamma = \theta\gamma \sqcup \theta\prime\gamma$ 是闭合的，且 $|(\theta \sqcup \theta\prime)\gamma| = |\theta\gamma| \cup |\theta\prime\gamma| \subseteq \mathrm{dom}(\Delta)$。$\Gamma \vdash_\Delta v : \exists A$

- $\overline{\Gamma \vdash_\Delta \mathrm{cl}(v) : \text{时钟}}\Gamma \vdash_\Delta v :\ \exists A$ 意味着 $v\gamma \in \mathrm{V}_\Delta[\![\exists A]\!](w)$（因为 v 要么是一个变量，要么是某个时钟的 $v = \text{等待}_\kappa\ \kappa$）。因此，$v\gamma \in \mathrm{Loc}_\Delta \cup \{\text{等待}_\kappa \mid \kappa \in \mathrm{dom}_\mathrm{p}(\Delta)\}$，所以 $\mathrm{cl}(v)\ \gamma$ 是闭合的时钟表达式，并且 $|\mathrm{cl}(v)\ \gamma| \subseteq \mathrm{dom}_\mathrm{p}(\Delta)$。

□

引理 A.11. 设 A 是一个值类型。那么 $v \in \mathcal{V}_\Delta[\![A]\!](n,\sigma)$ 当且仅当 $\vdash_\Delta v :\ A$。

证明。对 A 作直接归纳。 □

引理 *A.12.* 设 $\eta_N \in \text{堆}^\kappa, v \in V_\Delta[\![\exists A]\!](n+1, (\eta_N, [\eta_L]_{\kappa\notin}))$，$\iota :\ \Delta, \vdash w :\ \Delta(\kappa)$ 和 $\kappa \in |cl\ (v)|$。那么，对于任意 $\sigma \sqsupseteq \eta_N\langle\kappa \mapsto w\rangle\ \eta_L$，存在一些 σ' 和 $v' \in V_\Delta[\![A]\!](n,\sigma')$，使得 $hadv\ v;\sigma i \Downarrow hv';\ \sigma' i$。

证明。根据定义，v 要么是某个 $l \in \mathrm{Loc}$，要么具有 $\mathrm{wait}_{\kappa'}$ 的形式。

- 在前一种情形下，根据值关系的定义和引理A.3 ，

$$(\eta_N, [\eta_L]_{\kappa\notin})(l) \in \mathcal{T}_\Delta[\![A]\!](n, \left[\eta_N\,\langle\kappa \mapsto w\rangle\,[\eta_L]_{\kappa\notin}\right]_{\mathrm{cl}(l)}).$$

进而，由引理5.1 可得

$$(\eta_N, [\eta_L]_{\kappa\notin})(l) \in \mathcal{T}_\Delta[\![A]\!](n, \eta_N\,\langle\kappa \mapsto w\rangle\,\eta_L).$$

此外，由于 $\kappa \in$闭包 (l)，我们知道 $(\eta_N,\ [\eta_L]_{\kappa\notin})(l) = \eta_N(l)$。因此，存在一个归约 $\mathrm{h}\eta_N(l);\ \sigma\mathrm{i} \Downarrow \mathrm{h}v';\ \sigma'\mathrm{i}$，其中 $v' \in \mathrm{V}_\Delta[\![A]\!](n,\ \sigma')$；按定义，这意味着 $\mathrm{h}\mathrm{adv}\,v;\ \sigma\mathrm{i} \Downarrow \mathrm{h}v';\ \sigma'\mathrm{i}$。

• 在后一种情形下，我们知道 $\kappa' = \kappa$，因为 $\kappa \in |$闭包 (等待$_{\kappa'}$)$| = \{\kappa'\}$。此外，我们有 $\kappa :_c A \in \Delta$，适用于 $c \in \{\mathrm{p}, \mathrm{bp}\}$，因此 $\vdash w : A$。根据定义，$\eta_N \mathrm{h}\kappa \mapsto w\mathrm{i}\ \eta_L \sqsubseteq \sigma$意味着 σ的形式为 $\eta'_N \mathrm{h}\kappa \mapsto w\mathrm{i}\ \eta'_L$。因此，根据定义，hadv 等待$_\kappa$; $\sigma\mathrm{i} \Downarrow^{\iota} \mathrm{h}w; \sigma\mathrm{i}$。此外，根据引理 A.11 $w \in \mathrm{V}_\Delta[\![A]\!](n, \sigma)$。

□

定理 5.4 (基本性质)。给定 $\Gamma \vdash_\Delta$，$\Gamma \vdash_\Delta t : A$，以及 $\gamma \in C_\Delta[\![\Gamma]\!](n, \sigma)$，则 $t\gamma \in \mathcal{T}_\Delta[\![A]\!](n, \sigma)$。

定理 5.4 的证明。我们对类型推导 $\Gamma \vdash_\Delta t : A$ 进行结构归纳法。如果 $t\gamma$ 是一个值，根据引理 A.8，只需证明 $t\gamma \in \mathrm{V}_\Delta[\![A]\!](n, \sigma)$。在所有其他情形下，为了证明 $t\gamma \in \mathrm{T}_\Delta[\![A]\!](n, \sigma)$，我们假设存在某个输入缓冲区 $\iota : \Delta$ 和存储 $\sigma' \sqsupseteq^{\Delta}_{\checkmark} \sigma$，并证明存在 σ'' 和 v，使得 $\mathrm{h}t\gamma; \sigma\mathrm{i} \Downarrow^{\iota} \mathrm{h}v; \sigma''\mathrm{i}$ 且 $v \in \mathrm{V}_\Delta[\![A]\!](n, \sigma'')$。根据引理 5.1，我们可以假设 $\gamma \in \mathrm{C}_\Delta[\![\Gamma]\!](n, \sigma')$。

- $$\frac{\Gamma' \text{ tick-free or } A \text{ stable} \qquad \Gamma, x : A, \Gamma' \vdash_\Delta}{\Gamma, x : A, \Gamma' \vdash_\Delta x : A}$$

  我们证明 $x\gamma \in \mathcal{V}_\Delta[\![A]\!](n, \sigma)$。若 Γ' 是无tick的，则根据 $x\gamma \in \mathcal{V}_\Delta[\![A]\!](n, \sigma)$ 引理 A.5。若 Γ' 不是无tick的，则它具有以下形式：Γ_1，$\checkmark_\theta$，Γ_2 且 A 是稳定的。根据引理 A.5，
  $$\gamma|_{\Gamma, x:A} \in C_\Delta[\![\Gamma, x : A]\!](n + 1, \sigma')$$
  对于某个 σ'。因此，$x\gamma \in \mathcal{V}_\Delta[\![A]\!](n + 1, \sigma')$ 并且根据引理 5.1 和引理 A.4 $x\gamma \in \mathcal{V}_\Delta[\![A]\!](n, \sigma)$。

- $$\frac{}{\Gamma \vdash_\Delta \mathrm{hi} : 1}$$

  根据定义立即可得。

- $$\frac{\Gamma \vdash_\Delta s : A \qquad \Gamma, x : A \vdash_\Delta t : B}{\Gamma \vdash_\Delta \text{令 } x = s \text{取值为 } t : B}$$

  根据归纳法，我们有 $s\gamma \in \mathrm{T}_\Delta[\![A]\!](n, \sigma)$，这意味着 $\mathrm{h}s\gamma; \sigma'\mathrm{i} \Downarrow^{\iota} \mathrm{h}v; \sigma''\mathrm{i}$，其中某个$v \in \mathrm{V}_\Delta[\![A]\!](n, \sigma'')$成立。根据引理 5.1 和引理 A.1，$\gamma \in \mathrm{C}_\Delta[\![\Gamma]\!](n, \sigma'')$，因此
  $$\gamma[x \mapsto v] \in C_\Delta[\![\Gamma, x : A]\!](n, \sigma'').$$
  因此，我们可以应用归纳假设，得到 $t\gamma[x \mapsto v] \in \mathrm{T}_\Delta[\![B]\!](n, \sigma'')$。由于 γ 的值域中的所有元素都是闭项，因此 $t\gamma[x \mapsto v] = (t\gamma)[v/x]$，从而 $(t\gamma)[v/x] \in \mathrm{T}_\Delta[\![B]\!](n, \sigma'')$。于是，$\mathrm{h}(t\gamma)[v/x]; \sigma''\mathrm{i} \Downarrow^{\iota} \mathrm{h}w; \sigma'''\mathrm{i}$，其中 $w \in \mathrm{V}_\Delta[\![B]\!](n, \sigma''')$。根据定义，我们因此有 $\mathrm{h}(\mathrm{let}\ x = s \mathrm{in}\ t)\gamma; \sigma'\mathrm{i} \Downarrow^{\iota} \mathrm{h}w; \sigma'''\mathrm{i}$，其中 $w \in \mathrm{V}_\Delta[\![B]\!](n, \sigma''')$。

- $$\frac{x : A \vdash_\Delta t : B \qquad \Gamma \text{ tick-free}}{\Gamma \vdash_\Delta \lambda x.t : A \to B}$$
  我们证明 $\lambda x.t\gamma \in \mathrm{V}_\Delta[\![A \to B]\!](n, \sigma)$。为此，我们假设 $\sigma' \sqsupseteq^{\Delta}_{\checkmark} \mathrm{gc}(\sigma)$、$n' \le n$ 和 $v \in \mathrm{V}_\Delta[\![A]\!](n', \sigma')$，目标是证明 $(t\gamma)[v/x] \in \mathrm{T}_\Delta[\![B]\!](n', \sigma')$。根据引理 5.2 和引理 5.1，$\gamma \in \mathrm{C}_\Delta[\![\Gamma]\!](n', \sigma')$，因此根据定义，$\gamma[x \mapsto v] \in \mathrm{C}_\Delta[\![\Gamma, x : A]\!](n', \sigma')$。于是根据归纳法，我们有 $t\gamma[x \mapsto v] \in \mathrm{T}_\Delta[\![B]\!](n', \sigma')$。由于 γ 的值域中的所有元素都是闭项，$t\gamma[x \mapsto v] = (t\gamma)[v/x]$，因此 $(t\gamma)[v/x] \in \mathrm{T}_\Delta[\![B]\!](n', \sigma')$。

- $$\frac{\Gamma \vdash_\Delta s : A \to B \qquad \Gamma \vdash_\Delta t : A}{\Gamma \vdash_\Delta s\, t : B}$$

根据归纳法，我们有 $s\gamma \in \mathrm{T}_\Delta[\![A \to B]\!](n,\sigma)$，这意味着对于某个 $\lambda x.s' \in \mathrm{V}_\Delta[\![A \to B]\!](n,\sigma'')$，$\mathrm{h}s\gamma;\sigma\mathrm{i} \Downarrow^\iota \mathrm{h}\lambda x.s';\sigma''\mathrm{i}$。根据归纳法，我们还得到 $t\gamma \in \mathrm{T}_\Delta[\![A]\!](n,\sigma)$。由于根据引理 A.1，$\sigma'' \sqsupseteq^\Delta_\checkmark \sigma$，这意味着对于某个 $v \in \mathrm{V}_\Delta[\![A]\!](n,\sigma''')$，$\mathrm{h}t\gamma;\sigma''\mathrm{i} \Downarrow^\iota \mathrm{h}v;\sigma'''\mathrm{i}$。因此根据定义，$s'[v/x] \in \mathrm{T}_\Delta[\![B]\!](n,\sigma''')$，因为根据引理 A.2 和引理 A.1，$\sigma''' \sqsupseteq^{\Delta'}_\checkmark \mathrm{gc}(\sigma'')$。这意味着对于某个 $w \in \mathrm{V}_\Delta[\![B]\!](n,\sigma'''')$，我们有 $\mathrm{h}s[v/x];\sigma'''\mathrm{i} \Downarrow^\iota \mathrm{h}w;\sigma''''\mathrm{i}$。因此根据机器定义，$\mathrm{h}(s\gamma)(t\gamma);\sigma\mathrm{i} \Downarrow^\iota \mathrm{h}w;\sigma''''\mathrm{i}$。

- $$\frac{\Gamma \vdash_\Delta t : A \qquad \Gamma \vdash_\Delta t' : B}{\Gamma \vdash_\Delta (t,\ t') :\ A \times B}$$

  由归纳法可得 $s\gamma \in \mathrm{T}_\Delta[\![A]\!](n,\sigma)$，这意味着对于某个 $v \in \mathrm{V}_\Delta[\![A]\!](n,\sigma'')$，有 $\mathrm{h}s\gamma;\sigma\mathrm{i} \Downarrow^\iota \mathrm{h}v;\sigma''\mathrm{i}$。由归纳法也可得 $t\gamma \in \mathrm{T}_\Delta[\![A]\!](n,\sigma)$，根据引理 A.1，这意味着对于某个 $v' \in \mathrm{V}_\Delta[\![B]\!](n,\sigma''')$，有 $\mathrm{h}t\gamma;\sigma''\mathrm{i} \Downarrow^\iota \mathrm{h}v';\sigma'''\mathrm{i}$。因此，$\mathrm{h}(t,t');\sigma\mathrm{i} \Downarrow^\iota \mathrm{h}(v,v');\sigma'''\mathrm{i}$，并且根据引理 5.1，$(v,v') \in \mathrm{V}_\Delta[\![A \times B]\!](n,\sigma''')$。

- $$\frac{\Gamma \vdash_\Delta t :\ A_1 \times A_2 \qquad i \in \{1,\ 2\}}{\Gamma \vdash_\Delta \pi_i\, t :\ A_i}$$

  根据归纳法，我们有 $t\gamma \in \mathrm{T}_\Delta[\![A_1 \times A_2]\!](n,\ \sigma)$，这意味着 $\mathrm{h}t\gamma;\ \sigma\mathrm{i} \Downarrow^\iota \mathrm{h}(v_1,\ v_2);\ \sigma''\mathrm{i}$，其中 $v_i \in \mathrm{V}_\Delta[\![A_i]\!](n,\ \sigma'')$。此外，根据定义，$\mathrm{h}\pi_i\, t\gamma;\ \sigma\mathrm{i} \Downarrow^\iota \mathrm{h}v_i;\ \sigma''\mathrm{i}$。

- $$\frac{\Gamma \vdash_\Delta t :\ A_i \qquad i \in \{1,\ 2\}}{\Gamma \vdash_\Delta \mathrm{in}_i\, t :\ A_1 + A_2}$$

  根据归纳法，我们有 $t\gamma \in \mathrm{T}_\Delta[\![A_i]\!](n,\ \sigma)$，这意味着 $\mathrm{h}t\gamma;\ \sigma\mathrm{i} \Downarrow^\iota \mathrm{h}v;\ \sigma''\mathrm{i}$，其中 $v \in \mathrm{V}_\Delta[\![A_i]\!](n,\ \sigma'')$。因此，根据定义，$\mathrm{h}\mathrm{in}_i\, t\gamma;\ \sigma\mathrm{i} \Downarrow^\iota \mathrm{h}\mathrm{in}_i\, v;\ \sigma''\mathrm{i}$，且 $\mathrm{in}_i\, v \in \mathrm{V}_\Delta[\![A_1 + A_2]\!](n,\ \sigma'')$。

- $$\frac{\Gamma, x\colon A_i \vdash_\Delta t_i\colon B \qquad \Gamma \vdash_\Delta t\colon A_1 + A_2 \qquad i \in \{1,2\}}{\Gamma \vdash_\Delta \text{情形 } t \text{ 下，} \mathrm{in}_1\, x.t_1;\ \mathrm{in}_2\, x.t_2\colon\ B}$$

  根据归纳法，我们有 $t\gamma \in \mathrm{T}_\Delta[\![A_1 + A_2]\!](n,\sigma)$，这意味着对于某个 $i \in \{1,2\}$，有 $\mathrm{h}t\gamma;\sigma\mathrm{i} \Downarrow^\iota \mathrm{h}\mathrm{in}_i\, v;\sigma''\mathrm{i}$，且满足 $v \in \mathrm{V}_\Delta[\![A_i]\!](n,\sigma'')$。由引理 5.1 和引理 A.1，$\gamma \in \mathrm{C}_\Delta[\![\Gamma]\!](n,\sigma'')$，因此 $\gamma[x \mapsto v] \in \mathrm{C}_\Delta[\![\Gamma, x\colon A_i]\!](n,\sigma'')$。于是，我们可以应用归纳假设，得到 $t_i\gamma[x \mapsto v] \in \mathrm{T}_\Delta[\![B]\!](n,\sigma'')$。由于 γ 的值域中的所有元素都是闭项，$t_i\gamma[x \mapsto v] = (t_i\gamma)\,[v/x]$，因此 $(t_i\gamma)\,[v/x] \in \mathrm{T}_\Delta[\![B]\!](n,\sigma'')$。因此，$\mathrm{h}(t_i\gamma)\,[v/x];\ \sigma''\mathrm{i} \Downarrow^\iota \mathrm{h}w;\ \sigma'''\mathrm{i}$，且满足 $w \in \mathrm{V}_\Delta[\![B]\!](n,\sigma''')$。根据定义，我们也有 $\mathrm{h}(\mathsf{case}\ t\ \mathsf{of}\ \mathrm{in}_1\, x.t_1;\ \mathrm{in}_2\, x.t_2)\gamma;\ \sigma\mathrm{i} \Downarrow^\iota \mathrm{h}w;\ \sigma'''\mathrm{i}$。

- $$\frac{\Gamma, \checkmark_\theta \vdash_\Delta t :A \qquad \Gamma \vdash_\Delta \theta : \mathrm{Clock}}{\Gamma \vdash_\Delta \mathrm{delay}_\theta\, t :\ ②A}$$

  根据机器定义，我们有 $\mathrm{D}\ \mathrm{delay}^{\theta\gamma}\, t\gamma;\ \sigma'\mathrm{E} \Downarrow^\iota \mathrm{h}l;\ \sigma''\mathrm{i}$，其中 $\sigma'' = \sigma', l \mapsto t\gamma$ 且 $\mathrm{cl}\,(l) = |\theta\gamma|_\circ$。根据引理 A.10，$|\theta\gamma| \subseteq \mathrm{dom}_\mathrm{P}\,(\Delta)$。接下来只需证明 $l \in \mathrm{V}_\Delta[\![\circledcirc A]\!](n,\sigma'')$。对于 $n = 0$ 的情形，这可直接由以下事实得出：$|\theta\gamma| \subseteq \mathrm{dom}_\mathrm{P}\,(\Delta)$。

假设 $n = n'+1$，$\kappa \in \Theta$，且 $\vdash v\colon\ \Delta(\kappa)$，其中 $\Theta = |\theta\gamma|_\circ$。根据引理 5.1 和引理 5.2，我们有 $\gamma \in C_\Delta[\![\Gamma]\!](n'+1,\ \mathrm{gc}(\sigma''))$。因此，根据定义，我们有

$$\gamma \in C_\Delta[\![\Gamma, \checkmark_\theta]\!](n', [\mathrm{gc}(\sigma'')]_{\kappa\in}\, \langle \kappa \mapsto v\rangle\, [\mathrm{gc}(\sigma'')]_{\kappa\notin}) = C_\Delta[\![\Gamma, \checkmark_\theta]\!](n', \mathrm{tick}_{\kappa\mapsto v}\,(\mathrm{gc}(\sigma''))),$$

因此 $\gamma \in C_\Delta[\![\Gamma, \checkmark_\theta]\!](n', [\text{时钟滴答}_{\kappa\mapsto v}\,(\mathrm{gc}(\sigma''))]_\Theta)$ 根据引理 5.3。因此，我们可以应用归纳假设得出

$$t\gamma \in \mathcal{T}_\Delta[\![A]\!](n', [\mathrm{tick}_{\kappa\mapsto v}\,(\mathrm{gc}(\sigma''))]_\Theta).$$

由于 $\sigma''(l) = t\gamma$，因此我们有 $l \in \mathrm{V}_\Delta[\![\Diamond A]\!](n, \sigma'')$。

- $\dfrac{}{\Gamma \vdash_\Delta \text{永不} : \Diamond A}$
  根据机器定义，我们有 $\mathsf{hnever}; \sigma'\mathsf{i} \Downarrow^{\iota} \mathsf{h}l; \sigma'\mathsf{i}$，其中 $l = \mathrm{alloc}^{\emptyset}(\sigma)$。
  由于 $\mathrm{cl}(l) = \emptyset$，我们知道 $l \in \mathrm{V}_\Delta[\![\Diamond A]\!](n, \sigma')$。
- $\dfrac{\kappa :_c A \in \Delta, c \in \{\mathrm{p}, \mathrm{bp}\}}{\Gamma \vdash_\Delta \text{等待}\kappa : \Diamond A}$
  $\text{等待}_\kappa \gamma = \text{等待}_\kappa \in \mathrm{V}_\Delta[\![A]\!](n, \sigma)$ 可由定义和前提直接得出。
- $\dfrac{\kappa :_c A \in \Delta, c \in \{\mathrm{b}, \mathrm{bp}\}}{\Gamma \vdash_\Delta \text{读取}\kappa : A}$
  由于 $\iota : \Delta$，我们知道 $\vdash \iota(\kappa) : A$。因此，根据机器定义，$\mathsf{hread}\kappa; \sigma'\mathsf{i} \Downarrow^{\iota} \langle \iota(\kappa); \sigma' \rangle$。此外，根据引理 A.11 $\iota(\kappa) \in \mathcal{V}_\Delta[\![A]\!](n, \sigma')$。
- $\dfrac{\Gamma \vdash_\Delta v : \Diamond A \qquad \Gamma, \checkmark_{\mathrm{cl}(v)}, \Gamma' \vdash_\Delta}{\Gamma, \checkmark_{\mathrm{cl}(v)} \Gamma' \vdash_\Delta \mathsf{adv}\, v : A}$
  由引理 5.1 可知 $\gamma \in \mathrm{C}_\Delta[\![\Gamma, \checkmark_{\mathrm{cl}(v)}, \Gamma']\!](n, \sigma')$，由引理 A.5 可知 $\gamma|_\Gamma \in \mathrm{C}_\Delta[\![\Gamma, \checkmark_{\mathrm{cl}(v)}]\!](n, \sigma')$。令 $\Theta = |\mathrm{cl}(v)\ \gamma|_\Gamma|$。根据上下文关系的定义，$\Theta$ 定义良好，且是 $\mathrm{dom}^{\mathrm{p}}(\Delta)$ 的子集。根据上下文关系的定义，我们还可找到 $\kappa \in \Theta, \eta_N, \eta_L, \eta'_N, \eta'_L$，使得 $\sigma' = \eta_N \mathsf{h}\kappa \mapsto w\mathsf{i}\ \eta_L, \vdash w : \Delta(\kappa)$，$[\eta_N]_\Theta = [\eta'_N]_\Theta$，$[\eta_L]_\Theta = [\eta'_L]_\Theta$，且 $\gamma|_\Gamma \in \mathrm{C}_\Delta[\![\Gamma]\!](n+1, (\eta'_N, [\eta'_L]_{\kappa \notin}))$。由归纳法可得 $v\gamma \in \mathrm{T}_\Delta[\![\Diamond A]\!](n+1, (\eta'_N, [\eta'_L]_{\kappa \notin}))$。由于 $v\gamma$ 是一个值，根据引理 A.9，我们有 $v\gamma \in \mathrm{V}_\Delta[\![\Diamond A]\!](n+1, (\eta'_N, [\eta'_L]_{\kappa \notin}))$。于是，由引理 5.3 可得
  $$v\gamma \in \mathcal{V}_\Delta[\![\Diamond A]\!](n+1, (\eta_N, [\eta_L]_{\kappa \notin})).$$
  根据引理 A.12，我们进而得到一个归约 $\langle \mathsf{adv}\, v\gamma;\ \sigma' \rangle \Downarrow^{\iota} \langle v';\ \sigma'' \rangle$ 满足 $v' \in \mathcal{V}_\Delta[\![A]\!](n,\ \sigma'')$。
- $\dfrac{\Gamma \vdash_\Delta v_1 : \Diamond A_1 \qquad \Gamma \vdash_\Delta v_2 : \Diamond A_2 \qquad \vdash \theta = \text{闭包}\ (v_1) \sqcup \text{闭包}\ (v_2)\ \Gamma,\ \checkmark_\theta,\ \Gamma' \vdash_\Delta}{\Gamma, \checkmark_\theta, \Gamma' \vdash_\Delta \mathsf{select}\, v_1\, v_2 : ((A_1 \times \Diamond A_2) + (\Diamond A_1 \times A_2)) + (A_1 \times A_2)}$
  根据引理 5.1 我们有 $\gamma \in C_\Delta[\![\Gamma,\ \checkmark_\theta,\ \Gamma']\!](n,\ \sigma')$ 并且根据引理 A.5，$\gamma|_\Gamma \in C_\Delta[\![\Gamma,\ \checkmark_\theta]\!](n,\ \sigma')$。令 $\Theta_1 = |\mathrm{cl}(v_1)\ \gamma|_\Gamma|, \Theta_2 = |\mathrm{cl}(v_2)\ \gamma|_\Gamma|$，且 $\Theta = \Theta_1 \cup \Theta_2$。根据上下文关系的定义，$\Theta$ 定义良好，且是 $\mathrm{dom}(\Delta)$ 的子集。根据上下文关系的定义，我们还可找到 $\kappa \in \Theta, \eta_N, \eta_L, \eta'_N, \eta'_L$，使得 $\sigma' = \eta_N \mathsf{h}\kappa \mapsto w\mathsf{i}\ \eta_L, \vdash w : \Delta(\kappa), [\eta_N]_\Theta = [\eta'_N]_\Theta, [\eta_L]_\Theta = [\eta'_L]_\Theta$，且 $\gamma|_\Gamma \in \mathrm{C}_\Delta[\![\Gamma]\!](n+1, (\eta'_N, [\eta'_L]_{\kappa \notin}))$。由归纳假设可知，对于所有 $i \in \{1, 2\}$，均有 $v_i\gamma \in \mathrm{T}_\Delta[\![\Diamond A_i]\!](n+1, (\eta'_N, [\eta'_L]_{\kappa \notin}))$。由于 $v_i\gamma$ 是值，根据引理 A.9，我们还有 $v_i\gamma \in \mathrm{V}_\Delta[\![\Diamond A_i]\!](n+1, (\eta'_N, [\eta'_L]_{\kappa \notin}))$。因此，根据引理 5.3，对于所有 $i \in \{1, 2\}$，均有 $v_i\gamma \in \mathrm{V}_\Delta[\![\Diamond A_i]\!](n+1, (\eta_N, [\eta_L]_{\kappa \notin}))$。需要考虑两种情形：
  - 设 $i \in \{1, 2\}$ 且 $j = 3 - i$ 满足 $\kappa \in \Theta_i \setminus \Theta_j$：根据引理 A.12，存在一个归约 $\mathsf{hadv}\ v_i\gamma;\ \sigma'\mathsf{i} \Downarrow^{\iota} \mathsf{h}u_i;\ \sigma''\mathsf{i}$，且满足 $u_i \in \mathrm{V}_\Delta[\![A_i]\!](n, \sigma'')$。根据定义，这意味着 $\mathsf{hselect}\ v_1\gamma\, v_2\gamma;\ \sigma'\mathsf{i} \Downarrow^{\iota} \mathsf{hin}_1(\mathrm{in}_i(u_1, u_2));\ \sigma''\mathsf{i}$，且 $u_j = v_j\gamma$。因此，剩下要证明的是 $v_j\gamma \in \mathrm{V}_\Delta[\![\Diamond A_j]\!](n, \sigma'')$。需要考虑两种情形。
    - 令 $v_j\gamma = l$，其中 $l \in \mathrm{Loc}_\Delta$。由 $l \in \mathrm{V}_\Delta[\![\Diamond A_j]\!](n+1,\ (\eta_N,\ [\eta_L]_{\kappa \notin}))$ 以及事实 $\kappa \notin \Theta_j$，应用引理 5.3，可得 $l \in \mathrm{V}_\Delta[\![\Diamond A_j]\!](n+1,\ [\eta_L]_{\kappa \notin})$。特别地，我们用到事实 $[\eta_N]_{\Theta_j} = \emptyset$，因为 $\eta_N \in \mathrm{Heap}^\kappa$ 且 $\kappa \notin \Theta_j$。由于 $[\eta_L]_{\kappa \notin} \sqsubseteq^\Delta_\checkmark \sigma'$，并且根据引理 A.1，$\sigma' \sqsubseteq^\Delta_\checkmark \sigma''$，因此可以应用引理 5.1 得出 $l \in \mathrm{V}_\Delta[\![\Diamond A_j]\!](n,\ \sigma'')$。

∗令 $v_j\gamma = \mathsf{wait}_{\kappa'}$，其中时钟为 κ'。但此时 $\Gamma \vdash_\Delta v_j : \textcircled{\exists} A_j$ 是由 $\kappa' :_{\mathsf{p}} A_j \in \Delta$ 或 $\kappa' :_{\mathsf{bp}} A_j \in \Delta$ 导致的，因此根据值关系的定义，可立即得到 $v_j\gamma \in \mathrm{V}_\Delta[\![\textcircled{\exists} A_j]\!](n,\ \sigma'')$。

– $\kappa \in \Theta_1 \cap \Theta_2$：根据引理 A.12，我们得到一个归约 $\mathsf{hadv}\ v_1\gamma);\ \sigma'\mathrm{i} \Downarrow^\iota \langle v_1';\ \sigma''\rangle$，其满足 $v_1' \in \mathrm{V}_\Delta[\![A_1]\!](n, \sigma'')$；以及一个归约 $\mathsf{hadv}\, v_2\gamma;\ \sigma''\mathrm{i} \Downarrow^\iota \langle v_2';\ \sigma'''\rangle$，其满足 $v_2' \in \mathrm{V}_\Delta[\![A_2]\!](n, \sigma'')$。根据定义，我们由此得到一个归约 $\mathsf{hselect}\ (v_1\gamma)\ (v_2\gamma);\ \sigma'\mathrm{i} \Downarrow^\iota \langle \mathsf{in}_2(\ (v_1', v_2'));\ \sigma'''\rangle$。此外，应用引理 A.1 和引理 5.1，我们得到 $v_1' \in \mathrm{V}_\Delta[\![A_1]\!](n, \sigma'')$，这意味着我们有 $\mathsf{in}_2(\ (v_1', v_2')) \in \mathrm{V}_\Delta[\![A_1 \times A_2]\!](n, \sigma''')$。

- $\dfrac{}{\Gamma \vdash_\Delta 0 : \text{自然数}}$ $0\gamma \in \mathrm{V}_\Delta[\![\text{自然数}]\!](n,\ \sigma)$ 根据定义立即成立。

- $\dfrac{\Gamma \vdash_\Delta t : \text{自然数}}{\Gamma \vdash_\Delta \text{后继}\, t : \text{自然数}}$
根据归纳假设 $t\gamma \in \mathrm{T}_\Delta[\![\text{自然数}]\!](n,\ \sigma)$，这意味着 $\mathrm{h} t\gamma; \sigma'\mathrm{i} \Downarrow^\iota \mathrm{h}\mathsf{suc}^m\, 0;\ \sigma''\mathrm{i}$，其中 $m \in \mathbb{N}$。因此，根据定义，$\mathrm{h}\mathsf{suc}\, t\gamma;\ \sigma'\mathrm{i} \Downarrow^\iota \langle \mathsf{suc}^{m+1}\, 0;\ \sigma''\rangle$，且后继$^{m+1}$ $0 \in \mathrm{V}_\Delta[\![\text{自然数}]\!](n,\ \sigma'')$。

- $\dfrac{\Gamma \vdash_\Delta s : A \qquad \Gamma, x : \mathsf{Nat}, y : A \vdash_\Delta t : A \qquad \Gamma \vdash_\Delta u : \mathsf{Nat}}{\Gamma \vdash_\Delta \text{递归}_{\mathsf{Nat}}(s,\ x\, y.t,\ u) :\ A}$
我们断言以下结论成立：

$$\mathsf{rec}_{\mathsf{Nat}}(s\gamma, x\, y.t\gamma, \mathsf{suc}^k\, 0) \in \mathcal{T}_\Delta[\![A]\!](n, \sigma) \text{ for all } k \in \mathbb{N} \tag{5}$$

为了证明 $\mathsf{rec}_{\mathsf{Nat}}(s\gamma, x\, y.t\gamma, u\gamma) \in \mathrm{T}_\Delta[\![A]\!](n, \sigma)$，假设有某个 $\sigma' \sqsupseteq^\Delta_\checkmark \sigma$ 和 $\iota : \Delta$。根据归纳假设 $u\gamma \in \mathrm{T}_\Delta[\![\text{自然数}]\!](n, \sigma)$，这意味着 $\mathrm{h} u\gamma; \sigma'\mathrm{i} \Downarrow^\iota \mathrm{h}\mathsf{suc}^m\, 0;\ \sigma''\mathrm{i}$。由 (5) 和引理 A.1 可得 $\mathrm{h}\mathsf{rec}_{\mathsf{Nat}}(s\gamma, x\, y.t\gamma, \mathsf{suc}^m\, 0);\ \sigma''\mathrm{i} \Downarrow^\iota \mathrm{h} v;\ \sigma'''\mathrm{i}$，且 $v \in \mathrm{T}_\Delta[\![A]\!](n, \sigma''')$。根据机器定义，我们还可得 $\mathrm{h}\mathsf{rec}_{\mathsf{Nat}}(s\gamma, x\, y.t\gamma, u\gamma);\ \sigma'\mathrm{i} \Downarrow^\iota \mathrm{h} v;\ \sigma'''\mathrm{i}$。

我们通过对5进行归纳来证明 (k。

– 情形 $k = 0$: 令 $\sigma' \sqsupseteq^\Delta_\checkmark \sigma$ 和 $\iota : \Delta$。根据定义，$\mathrm{h}0; \sigma'\mathrm{i} \Downarrow^\iota \mathrm{h}0; \sigma'\mathrm{i}$。根据归纳假设 $s\gamma \in \mathrm{T}_\Delta[\![A]\!](n,\ \sigma)$，这意味着对于某个 $v \in \mathrm{V}_\Delta[\![A]\!](n,\ \sigma')$，有 $\mathrm{h} s\gamma; \sigma'\mathrm{i} \Downarrow^\iota \mathrm{h} v; \sigma''\mathrm{i}$。根据定义，可得 $\mathrm{h}\mathsf{rec}_{\mathsf{Nat}}(s\gamma,\ x\, y.t\gamma,\ 0);\ \sigma'\mathrm{i} \Downarrow^\iota \mathrm{h} v;\ \sigma''\mathrm{i}$。

– 情形 $k = l + 1$。设 $\sigma' \sqsupseteq^\Delta_\checkmark \sigma$ 和 $\iota :\ \Delta$。根据定义，$\langle \text{后继}^k\, 0;\ \sigma'\rangle \Downarrow^\iota \langle \text{后继}(\text{后继}^l\, 0);\ \sigma'\rangle$。根据归纳法（对 k），我们有 $\langle \mathsf{rec}_{\mathsf{Nat}}(s\gamma, x\, y.t\gamma, \text{后继}^l\, 0);\ \sigma'\rangle \Downarrow^\iota \mathrm{h} v;\ \sigma''\mathrm{i}$，其中 $v \in \mathrm{V}_\Delta[\![A]\!](n, \sigma'')$。根据引理 A.1 和引理 5.1，我们有 $\gamma \in \mathrm{C}_\Delta[\![\Gamma]\!](n, \sigma'')$，因此

$$\gamma[x \mapsto \mathsf{suc}^l\, 0, y \mapsto v] \in C_\Delta[\![\Gamma, x : \mathsf{Nat}, y : A]\!](n, \sigma'').$$

因此，根据归纳法可得

$$(t\gamma)[\mathsf{suc}^l\, 0/x, v/y] = t\gamma[x \mapsto \mathsf{suc}^l\, 0, y \mapsto v] \in \mathcal{T}_\Delta[\![A]\!](n, \sigma''),$$

这意味着存在一个归约 $\langle (t\gamma)[\text{后继}^l\, 0/x,\ v/y];\ \sigma''\rangle \Downarrow^\iota \mathrm{h} w;\ \sigma'''\mathrm{i}$，其中 $w \in \mathrm{V}_\Delta[\![A]\!](n,\ \sigma''')$。根据机器定义，我们因此有

$$\left\langle \mathsf{rec}_{\mathsf{Nat}}(s\gamma, x\, y.t\gamma, \mathsf{suc}^k\, 0); \sigma' \right\rangle \Downarrow^\iota \langle w; \sigma'''\rangle\,.$$

- $\dfrac{\Gamma^\square,\ x :\ \textcircled{\vee} A \vdash_\Delta t :\ A}{\Gamma \vdash_\Delta \text{不动点}\ x.t :\ A}$
我们将证明

$$\mathrm{dfix}\, x.t\gamma \in \mathcal{V}_\Delta[\![\textcircled{\vee} A]\!](m, \emptyset) \text{ for all } m \le n. \tag{6}$$

利用 (6)、引理 A.6 和引理 5.1，我们进而得到 $(\gamma|_{\Gamma^\Box})\ [x \mapsto \mathrm{dfix}\ x.t\gamma] \in \mathsf{C}_\Delta[\![\Gamma^\Box, x{:}\circledcirc A]\!](n,\sigma)$。因此，根据归纳法，$t[\mathrm{dfix}\ x.t/x]\gamma = t(\gamma|_{\Gamma^\Box})\ [x \mapsto \mathrm{dfix}\ x.t\gamma] \in \mathsf{T}_\Delta[\![A]\!](n,\sigma)$，这意味着我们找到了 $\mathsf{h}_t[\mathrm{dfix}\ x.t/x]\gamma;\ \sigma' \mathsf{i}\ \Downarrow^\iota \mathsf{h}v;\ \sigma''\mathsf{i}$，且满足 $v \in \mathsf{V}_\Delta[\![A]\!](n,\sigma'')$。因此，根据机器的定义，我们得到所需的 h不动点 $x.t\gamma;\ \sigma\mathsf{i}\ \Downarrow^\iota \mathsf{h}v;\ \sigma''\mathsf{i}$。

We prove (6) by 对 m进行归纳。

若 $m = 0$，则 6) 可由以下事实立即推出：$\mathrm{dfix}\ x.t\gamma$是一个闭项 .

设 $m = m' + 1$。由引理 5.1 和引理 A.6，$\gamma|_{\Gamma^\Box} \in C_\Delta[\![\Gamma^\Box]\!](m',\emptyset)$。根据归纳假设（对 (6)），我们有 $\mathrm{dfix}\ x.t\gamma \in \mathcal{V}_\Delta[\![\circledcirc A]\!](m',\emptyset)$，因此

$$(\gamma|_{\Gamma^\Box})[x \mapsto \mathrm{dfix}\, x.t\gamma] \in C_\Delta[\![\Gamma^\Box, x : \circledcirc A]\!](m',\emptyset).$$

因此，由归纳法，$t[\mathrm{dfix}\ x.t/x]\gamma = t(\gamma|_{\Gamma^\Box})\ [x \mapsto \mathrm{dfix}\ x.t\gamma] \in \mathsf{T}_\Delta[\![A]\!](m',\ \emptyset)$，从而可以得出结论：$\mathrm{dfix}\ x.t\gamma \in \mathsf{V}_\Delta[\![\circledcirc A]\!](m,\ \emptyset)$。

- $\dfrac{\Gamma \vdash_\Delta x :\ \circledcirc A \qquad \Gamma, \checkmark_\theta, \Gamma' \vdash_\Delta}{\Gamma,\ \checkmark_\theta,\ \Gamma, \vdash_\Delta \mathrm{adv}\, x :\ A}$

由引理 5.1 可知，$\gamma \in \mathsf{C}_\Delta[\![\Gamma, \checkmark_\theta, \Gamma']\!](n,\sigma')$；由引理 A.5 可知，$\gamma|_\Gamma \in \mathsf{C}_\Delta[\![\Gamma, \checkmark_\theta]\!](n,\sigma')$。令 $\Theta = |\theta\gamma|_\Gamma|_\circ$。根据上下文关系的定义，$\Theta$定义良好，且我们可以找到 $\kappa \in \Theta, \vdash w :\ \Delta(\kappa)$，$\eta_N, \eta_L, \eta'_N, \eta'_L$，使得 $\sigma' = \eta_N \mathsf{h}\kappa \mapsto w\mathsf{i}\ \eta_L,\ [\eta_N]_\Theta = \left[\eta'_N\right]_\Theta, [\eta_L]_\Theta = \left[\eta'_L\right]_\Theta$，并且 $\gamma|_\Gamma \in \mathsf{C}_{\Delta'}[\![\Gamma]\!](n+1, (\eta'_N, \left[\eta'_L\right]_{\kappa\notin}))$。由引理 A.5，我们因此有$\gamma(x) = \mathrm{dfix}\, y.t$，其中 $\mathrm{dfix}\, y.t \in \mathsf{V}_\Delta[\![\circledcirc A]\!](n+1, (\eta'_N, \left[\eta'_L\right]_{\kappa\notin}))$。根据定义，这意味着$t[\mathrm{dfix}\, y.t/y] \in \mathsf{T}_\Delta[\![A]\!](n,\emptyset)$。也就是说，我们可以找到一个归约 $\mathsf{h}t[\mathrm{dfix}\, y.t/y];\ \sigma'\mathsf{i}\ \Downarrow^\iota \mathsf{h}v;\ \sigma''\mathsf{i}$，其中$v \in \mathsf{V}_\Delta[\![A]\!](n,\sigma'')$；根据定义，这意味着我们也有一个归约 $\mathsf{h}\mathrm{adv}\ x\gamma;\sigma'\mathsf{i}\ \Downarrow^\iota \mathsf{h}v;\ \sigma''\mathsf{i}$。

- $\dfrac{\Gamma^\Box \vdash_\Delta t : A\Gamma \vdash_\Delta \text{箱}\, t :\ \Box A}{\Gamma \vdash_\Delta t : \Box A}$我们证明箱 $t\gamma \in \mathsf{V}_\Delta[\![\Box A]\!](n,\sigma)$。由引理 A.6，$\gamma|_{\Gamma^\Box} \in \mathsf{C}_\Delta[\![\Gamma^\Box]\!](n,\emptyset)$。因此，根据归纳法，$t\gamma = t\gamma|_{\Gamma^\Box} \in \mathsf{T}_\Delta[\![A]\!](n,\emptyset)$，从而箱 $t\gamma \in \mathsf{V}_\Delta[\![\Box A]\!](n,\sigma)$。

- $\dfrac{\Gamma \vdash_\Delta \text{拆箱}\, t :\ A}{\Gamma \vdash_\Delta t : \Box A}$根据归纳假设，我们有 $t\gamma \in \mathsf{T}_\Delta[\![\Box A]\!](n,\sigma)$。也就是说，对于某个 $s \in \mathsf{T}_\Delta[\![A]\!](n,\emptyset)$，$\mathsf{h}t\gamma;\ \sigma'\mathsf{i}\ \Downarrow^\iota \mathsf{h}\text{箱}\, s;\ \sigma''\mathsf{i}$。因此，$\mathsf{h}s;\ \sigma''\mathsf{i}\ \Downarrow^\iota \mathsf{h}v;\sigma'''\mathsf{i}$，使得 $v \in \mathsf{V}_\Delta[\![A]\!](n,\sigma''')$，由引理 5.1 可知，这意味着 $v \in \mathsf{V}_\Delta[\![A]\!](n,\sigma''')$。此外，根据机器定义，我们有 $\mathsf{h}\mathrm{unbox}\ t;\sigma'\mathsf{i}\ \Downarrow^\iota \mathsf{h}v;\sigma'''\mathsf{i}$。$\Gamma \vdash_\Delta t :$ 不动点 $\alpha.A$

- $\dfrac{\Gamma \vdash_\Delta \mathrm{out}\, t :\ A[\circledcirc(\mathrm{Fix}\ \alpha.A)/\alpha]}{\Gamma \vdash_\Delta t :\ \circledcirc(\mathrm{Fix}\ \alpha.A)}$根据归纳假设 $t\gamma \in \mathsf{T}_\Delta[\![\mathrm{Fix}\, \alpha.A]\!](n,\sigma)$，这意味着 $\mathsf{h}t\gamma;\ \sigma\prime\mathsf{i}\ \Downarrow^\iota \mathsf{h}\text{into}\ v;\ \sigma\prime\prime\mathsf{i}$，其中某个 $v \in \mathsf{V}_\Delta[\![A[\circledcirc(\mathrm{Fix}\, \alpha.A)/\alpha]]\!](n,\sigma\prime\prime)$ 满足此条件。此外，根据机器定义，我们因此有 $\mathsf{h}\mathrm{out}\, t\gamma;\ \sigma\prime\mathsf{i}\ \Downarrow^\iota \mathsf{h}v;\ \sigma\prime\prime\mathsf{i}.\Gamma \vdash_\Delta t :\ A[\circledcirc(\mathrm{Fix}\, \alpha.A)/\alpha]$

- $\dfrac{\Gamma \vdash_\Delta \mathrm{into}\, t\colon \mathrm{Fix}\ \alpha.A}{\Gamma \vdash_\Delta \text{拆箱}\, t :\ A}$根据归纳假设 $t\gamma \in \mathsf{T}_\Delta[\![A[\circledcirc(\mathrm{Fix}\, \alpha.A)/\alpha]]\!](n,\sigma)$，这意味着 $\mathsf{h}t\gamma;\ \sigma\prime\mathsf{i}\ \Downarrow^\iota \mathsf{h}v;\ \sigma\prime\prime\mathsf{i}$，其中 $v \in \mathsf{V}_\Delta[\![A[\circledcirc(\mathrm{Fix}\ \alpha.A)/\alpha]]\!](n,\sigma\prime\prime)$。因此，根据定义，$\mathrm{into}\, t\gamma;\ \sigma\prime\mathsf{i}\ \Downarrow^\iota \mathsf{h}\text{into}\ v;\ \sigma''\rangle$

并进入$v \in \mathsf{V}_\Delta[\![$的不动点 $\alpha.A]\!](n,\sigma'')$。

□