arXiv:2302.07782v1 [cs.PL] 15 Feb 2023

多重分级轻量级 Java

Riccardo Bianchini DIBRIS
热那亚大学，意大利
FrancescoDagnino ！ DIBRIS 热那亚大学，意大利
Paola Giannini
DiSSTE，皮埃蒙特东方大学，意大利
ElenaZucca ！
DIBRIS，热那亚大学，意大利

摘要

资源感知的类型系统不仅会静态近似程序预期的结果类型，还会近似外部资源的使用方式，例如，某个变量的值会被需要多少次。我们将轻量级 Java 的类型系统扩展为资源感知型，并以任意一种用于刻画特定资源使用情况的等级代数模型为参数。我们证明，该类型系统对于资源感知版本的归约是可靠的；也就是说，良类型程序存在一个归约序列，不会因资源消耗而陷入停滞。此外，我们还表明，可用的等级可以是异构的，即通过从一种种类到另一种种类的一组最小同态组合不同种类的等级。最后，我们展示如何将等级代数和同态指定为 Java 类，从而可以直接在该语言中编写类型中的分级注解。

2012 年 ACM 学科分类计算理论

关键词与短语分级模态类型，Java

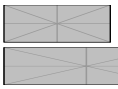
数字对象标识符10.4230/LIPIcs...

致谢 本研究部分由 MUR 项目“T-LADIES”（PRIN 2020TL3X8X）资助，并获得皮埃蒙特东方大学的财政支持。我们感谢匿名审稿人提出的宝贵建议。

1 引言

近年来，大量研究 [24, 6, 2, 13, 14, 22, 7, 10] 致力于允许对资源使用进行推理的类型系统。在（类型与）共效应系统中，类型判断呈现为 $x_1 :^{r_1} T1$，...，$x_n :^{r_n} T_n \vdash e : T$，其中共效应（等级）$r_i$用于建模变量 x_i 在 e 中的使用方式。例如，形如 $r ::= 0 \mid 1 \mid \omega$ 的共效应分别表示变量未被使用、至多使用一次，或以不受限制的方式使用。通过这种方式，在简单类型 λ演算中具有相同类型的函数，例如 $\lambda x{:}int.5$、$\lambda x{:}int.x$ 和 $\lambda x{:}int.x + x$，可以通过添加共效应注释加以区分：$\lambda x{:}int[0].5$、$\lambda x{:}int[1].x$ 和 $\lambda x{:}int[\omega].x + x$。其他例子包括精确使用（共效应为自然数）和隐私级别。分级模态类型更进一步，通过为类型本身添加等级，来规定表达式的结果应如何被使用。现有文献中的不同方案所采用的等级具有相似的代数结构，基本上是一个半环，用于规定加法 +、乘法 ·、0 和 1 常量，以及某种序关系。本文将假设这一概念的一个变体，称为等级代数。

资源感知型类型检查已在一门完整的程序设计语言 Granule [22], 中得到应用。Granule 是一种配备分级模态类型的函数式语言，因此程序员可以编写类似上述内容的函数声明。在 Granule 中，可以同时使用不同种类的副效应，包括用于精确使用的自然数、隐私


©Riccardo Bianchini、Francesco Dagnino、Paola Giannini 和 Eelena Zucca；依据知识共享许可协议 CC-BY 4.0 授权。Leibniz国际信息学论文集，Schloss Dagstuhl – Leibniz信息学中心，uhl出版社，德国

级别、区间、无穷以及副作用产品；但是，可用的等级在语言中是固定的。本文所述工作的最初目标，是通过在 Featherweight Java（FJ） [18], 的一个变体中引入带有等级修饰的类型，研究对类 Java 语言提供类似支集的可能性。此外，我们希望这些等级能够参数化地取自任意等级代数；更进一步，我们不希望像 Granule 那样固定等级代数，而是希望程序员能够借助面向对象语言的继承机制，使用用户定义的等级对其进行扩展。在追求这些目标的过程中，我们提出了若干据我们所知相对于资源感知类型系统相关文献而言具有新颖性的想法，具体内容详见下文给出的贡献概要。

资源感知的参数化 FJ 归约。给定一个资源感知类型系统，我们希望证明，类型判断对资源使用量的估算是过度近似的。然而，标准操作语义并未对资源使用量建模；因此， [7] 提出了一种插桩操作语义[1] ，并证明了一个可靠性定理，表明资源使用量得到了正确核算。受这项工作的启发，我们针对 FJ 定义了一种资源感知语义，它以任意等级代数为参数，跟踪每一步消耗的各项可用资源数量；如果所需资源不足，则计算会陷入停滞。与此不同的是，[7], 语义的定义独立于类型系统，这也是演算系统中的标准做法。换言之，我们的目标还在于提供一种简单、纯粹的语义模型，将资源使用纳入考量。资源感知归约相对于标准归约是可靠的，但显然并不完备，因为标准语义允许的规约步可能会因资源消耗而无法执行。

分级 FJ。 定义资源感知型演算后，我们定义资源感知类型系统。也就是说，类型会附加等级，使程序员能够指定变量、字段或方法结果的使用方式，例如使用次数。与分级模态类型相关文献中通常采用的方法相比，我们的方法是新颖的。值得注意的是，在这类研究中，类型的构造规则为 $T ::= \ldots \mid T_r$，也就是说，等级标注可以任意嵌套。相应地，语法中包含显式的 box构造，它通过一条 提升规则将类型为 T的项转换为类型为 T_r的项，该规则会将上下文乘以 r，并配有相应的拆箱机制。这里，我们倾向于采用一种轻量得多的方法，这种方法可能更适合类 Java 语言，因为它不会影响项的语法。类型的构造规则为 $T ::= C_r$，也就是说，所有类型（此处仅有类名）都只附加一次等级；在上下文中，类型不带等级，等级则用作副效应，从而得到形如 $x_1 :^{r_1} C_1, \ldots, x_n :^{r_n} C_n \vdash e : C_r$的判断。最后，由于不存在装箱/拆箱，也就没有显式的提升规则；不过，假设采用不同的共效上下文，可以为一个表达式赋予不同的等级。我们证明了一个可靠性定理，指出分级类型系统对资源使用进行了过近似，因此能够保证可靠性；由此，对于良类型程序，该系统相对于标准归约也是完备的。

组合等级。 接下来要解决的问题是如何让语言支持多等级，也就是说，程序员可以使用不同种类的等级，例如自然数和隐私级别。这样一来，就需要定义当类型系统组合不同种类的等级时，结果应是什么。Granule 语言中已考虑过这个问题[22], ，但其中可用的等级种类是固定的，因此可以通过特设方式进行组合。我们希望能有更大的灵活性，也就是允许程序员

[1] 随后， [7] 的模型于 [20], 被用于追踪唯一性的引用计数。

定义新的等级，并将其添加到已有等级中，这与 Java 程序员可以定义自己的 Exception 类非常相似。为此，我们定义了一种构造：给定一个等级代数族和一个同态族，可得出一个异构等级的唯一等级代数。这使得我们能够采用模块化方法，也就是说，已建立的元理论（包括结果的证明）同样适用于这种情况。

将等级表示为 Java 表达式。最后，我们探讨如何提供语言支持，以便指定所需的等级代数和同态。当然，可以通过使用专门的配置语言来实现这一点。不过，我们认为，一个有趣的解决方案是直接用 Java 编写分级注解，这与异常的处理方式类似。我们将说明如何编写对应于等级代数和同态的 Java 类，并给出一些示例。

本文所述成果的一个前期步骤是 [3], ，该工作提出了一个初步版本的类型系统，其中只有共效应（类型不分级），并给出了上述构造的一个简化版本：组合不同种类的共效应会得到平凡共效应。

第 2 节形式化定义等级代数及相关概念。第 3 节定义了针对 FJ 的参数化资源感知归约，第4节定义了参数化资源感知类型系统，并证明其可靠性。第 5 节定义异构等级的等级代数构造，第6节说明如何在 Java 中表达等级代数和同态。最后，第 7 节综述相关工作，第 8 节总结本文的贡献并展望未来工作。第 5 节的证明见附录。

2 代数预备知识

本节将介绍本文中将使用的代数结构。我们的工作核心是等级，即代码中的注释，用于表达程序如何使用资源，或使用了多少资源。正如我们将看到的，为了在资源感知语义和类型规则中正确地组合等级，我们需要一些运算，因此我们假设等级构成一种称为等级代数的代数结构，其定义如下。

▶ 定义 *1*（等级代数）。等级代数是一个元组 $R = \langle |R|, \preceq, +, \cdot, 0, 1 \rangle$，满足：$\langle |R|, \preceq \rangle$ 是一个偏序集；$\langle |R|, +, 0 \rangle$ 是一个交换幺半群；$\langle |R|, \cdot, 1 \rangle$ 是一个幺半群；

并且满足以下公理：

- $r \cdot (s + t) = r \cdot s + r \cdot t$ 且 $(s + t) \cdot r = s \cdot r + t \cdot r$，对所有 $r, s, t \in |R|$；$r \cdot \mathbf{0} = \mathbf{0}$ 且 $\mathbf{0} \cdot r = \mathbf{0}$，对所有 $r \in |R|$；若 $r \preceq r'$ 且 $s \preceq s'$，则 $r + s \preceq r' + s'$ 且 $r \cdot s \preceq r' \cdot s'$，对所有 $r, r's, s' \in |R|$；$\mathbf{0} \preceq r$，对所有 $r \in |R|$。

本质上，等级代数是一个有序半环，也就是说，它是在载体上配有偏序关系的半环，并且加法和乘法相对于该偏序关系都是单调的。我们还要求半环的零是该偏序中的最小元。我们的定义是对文献中其他定义的轻微变体[6, 14, 21, 2, 13, 1, 22, 7, 26]。具体而言，该偏序对资源使用进行过近似，并允许灵活处理，例如，if-then-else 构造的不同分支可以有不同的使用方式。零是最小元这一事实意味着，尤其是，过近似可以加入未使用的变量，从而使演算仿射。

XX：4 多重分级 FeatherweightJava

▶示例 2. 1. 半环自然数 Nat $=\mathrm{h}\mathbb{N}, \leq, +, \cdot, 0, 1\mathrm{i}$，配备自然序和通常的算术运算，是一个等级代数。2. 亲和性等级代数 $\mathrm{h}\{0, 1, \infty\} \leq, +, \cdot, 0, 1\mathrm{i}$由前一个代数将所有大于 1 的自然数视为相同而得到。3. 平凡半环 Triv，其载体是单元素集合 $|\mathrm{Triv}| = \{\infty\}$，偏序为等号关系，加法和乘法按平凡方式定义$\mathbf{0}_{\mathsf{Triv}} = \mathbf{1}_{\mathsf{Triv}} = \infty$，是一个等级代数。4. 半环 $\mathrm{R}_{\geq 0}^{\infty} = \mathrm{h}[0, \infty], \leq, +, \cdot, 0, 1\mathrm{i}$，其载体为扩展非负实数，并配备通常的序和运算，且以预期的方式扩展到 ∞，是一个等级代数。5. 分配格 $\mathrm{L} = \mathrm{h}|\mathrm{L}|, \leq, \vee, \wedge, \bot, \top\mathrm{i}$，其中 $\vee$和 $\wedge$分别表示并运算和交运算，而 $\bot$ 和 $\top$ 分别表示底元和顶元素，是一个等级代数。6. 布尔等级代数是 $\mathrm{B} = \mathrm{h}\{0, 1\} \leq, \vee, \wedge, 0, 1\mathrm{i}$，其中 $0 \leq 1$。它是一个等级代数，因为它是分配格。7. 给定等级代数 $\mathrm{R} = \mathrm{h}|\mathrm{R}|, \preceq_R, +_R, \cdot_R, \mathbf{0}_R, \mathbf{1}_R\mathrm{i}$ 和 $\mathrm{S} = \mathrm{h}|\mathrm{S}|, \preceq_S, +_S, \cdot_S, \mathbf{0}_S, \mathbf{1}_S\mathrm{i}$，积 $\mathrm{R} \times \mathrm{S} = \mathrm{h}\{\mathrm{h}r, s\mathrm{i} \mid r \in |\mathrm{R}| \wedge s \in |\mathrm{S}|\} \preceq, +, \cdot, \mathrm{h}\mathbf{0}_R, \mathbf{0}_S\mathrm{i}, \mathrm{h}\mathbf{1}_R, \mathbf{1}_S\mathrm{ii}$ 中的运算是分别对 R 和 S 的运算逐对应用，因此该积是一个等级代数。8. 给定等级代数 $\mathrm{R} = \mathrm{h}|\mathrm{R}|, \preceq_R, +_R, \cdot_R, \mathbf{0}_R, \mathbf{1}_R\mathrm{i}$，如 [22] 中所述，我们定义 $\mathrm{Ext\ R} = \mathrm{h}|\mathrm{R}| + \{\infty\} \preceq, +, \cdot\ \mathbf{0}_R, \mathbf{1}_R\mathrm{i}$，其中 $\preceq$通过添加 $r \preceq \infty$扩展 $\preceq_R$，适用于所有 $r \in |\mathrm{Ext\ R}|$和 $+$，并

$\cdot$对所有 $r \in |\mathrm{Ext}\ R|$，通过 $r + \infty = \infty + r = \infty$扩展 $+_R$和 $\cdot_R$，并且对所有带有 $r \neq \mathbf{0}_R$ 的 $r \in |\mathrm{Ext}R|$，满足 $r \cdot \infty = \infty \cdot r = \infty$，以及 $\mathbf{0}_R \cdot \infty = \infty \cdot \mathbf{0}_R = \mathbf{0}_R$。于是，Ext R是一个等级代数。

等级代数同态 $f\colon R \to S$是底层偏序之间的单调函数 $f\colon \langle |R|, \preceq_R \rangle \to \langle |S|, \preceq_S \rangle$，它保持半环结构，即满足以下方程：

- $f(\mathbf{0}_R) = \mathbf{0}_S$和 $f(r +_R s) = f(r) +_S f(s)$，对所有 $r, s \in |R|$;
- $f(\mathbf{1}_R) = \mathbf{1}_S$和 $f(r \cdot_R s) = f(r) \cdot_S f(s)$，对所有 $r, s \in |R|$。

等级代数及其同态构成一个范畴，记作 GrAlg.

考虑一个等级代数 R。于是，我们可以定义函数 $\zeta_R\colon |R| \to |\mathrm{Triv}|$ 和$\iota_R\colon |\text{自然数}| \to |R|$ 如下：

$$\zeta_R(r) = \infty \qquad \iota_R(m) = \begin{cases} \mathbf{0}_R & \text{if } m = 0 \\ \iota_R(n) +_R \mathbf{1}_R & \text{if } m = n + 1 \end{cases}$$

粗略地说，ζ_R 将R 中的每个元素映射到 ∞，而 ι_R 将自然数 n 映射到R 中 n 个 $\mathbf{1}_R$ 相加的和。我们很容易验证，这两个函数都导出同态 $\zeta_R\colon R \to \mathrm{Triv}$ 和 $\iota_R\colon$自然数 $\to R$。对于 ζ_R，这一事实显然成立；而对于 ι_R，则可由算术归纳法得出。接下来，利用这些同态，我们可以证明以下结果。

▶**命题 3**。 以下事实成立：1. 自然数 是 GrAlg;2. Triv 是 GrAlg.中的始对象中的终对象

我们还将处理另一种对象：为变量赋予等级的映射。这些映射继承了底层等级代数所具有的优良代数结构。

设有等级代数 $\mathrm{R} = \mathrm{h}|\mathrm{R}|, \preceq, +, \cdot, 0, 1\mathrm{i}$ 以及集合 X。从X 到 $|\mathrm{R}|$ 的函数集合，带有一种偏序交换幺半群结构，该结构由 R 的加法结构逐点扩张而得。也就是说，给定 $\gamma, \gamma'\colon X \to |\mathrm{R}|$，我们定义 $\gamma \preceq \gamma'$当且仅当，对于所有 $x \in X$, γ，$(x) \preceq \gamma'(x)$，并且 $(\gamma + \gamma')(x) = \gamma(x) + \gamma'(x)$，且 $\hat{}0(x) = \mathbf{0}$，对于所有 $x \in X$。此外，我们可以定义数乘，将 $|\mathrm{R}|$ 中的元素与一个函数

$\gamma\colon X \to |R|$；事实上，我们令 $(r \cdot \gamma)(x) = r \cdot \gamma(x)$，对所有 $r \in |R|$ 和 $x \in X$ 均成立。容易看出，这一运算将从 X 到$|R|$ 的函数所成的偏序交换幺半群变成一个偏序 R-模。

函数 $\gamma\colon X \to |R|$ 的支集是集合 $\mathsf{S}(\gamma) = \{x \in X \mid \gamma(x) \neq 0\}$。将$R^X$ 记为具有有限支集的函数 $\gamma\colon X \to |R|$ 的集合。上文定义的偏序和运算可以安全地限制到 R^X，注意到 $\mathsf{S}(0) = \emptyset$、$\mathsf{S}(\gamma + \gamma') \subseteq \mathsf{S}(\gamma) \cup \mathsf{S}(\gamma')$ 以及 $\mathsf{S}(r \cdot \gamma) \subseteq \mathsf{S}(\gamma)$。因此，$R^X$ 也具有偏序R-模结构。

正如我们将在第4节中看到的，共效应上下文是 R^X 中函数的（表示），其 X 为变量集合。共效应上下文构成模这一事实最初在 [21, 26]，中被注意到，并在 [4]，中得到完整形式化；后者还展示了一个非结构化示例。也就是说，需要一个不同于上文所述R^X 的模；本文以及大部分相关文献都使用这种模，其中对共效应上下文的运算不是逐点进行的，也就是说，共效应无法按变量分别计算。

3 资源感知语义

标准操作模型并未说明计算所使用的资源。为了解决这一问题，我们采用与 [7], 中类似的方法，也就是说，我们定义一种带跟踪的 语义，用以跟踪资源使用；因此，特别是当所需资源不足时，计算会卡住。然而，与 [7], 不同，我们的资源感知语义定义虽然以等级代数为参数，却是独立于分级类型系统给出的，这也是演算系统中的标准做法；在下一节中，我们将展示分级类型系统如何实际地对资源使用进行过近似，从而保证可靠性。如下文将详述，资源感知语义是非确定性的，因为需要资源时，可以通过不同方式消耗资源；因此，这里的可靠性是可能健全性，意即存在一种归约，不会因标准类型错误或资源消耗而卡住。

3.1 参考演算

该演算是 FJ [18] 的一个变体。语法见图1上方部分。我们用 es 作为元变量，表示 $e_1, \ldots, e_n$ $n \geq 0$；其他序列也采用类似表示。我们假设 变量 $x, y, z \ldots$、类名 C、D、字段名 f以及方法名 m。类型与类名相互区分，这意味着类型还可以扩展为包括其他类型，例如基本类型。除标准的 FJ 构造外，我们还引入了块表达式，它由一个局部变量声明以及一个可以使用该变量的方法体组成。

语义的定义不同于原始定义；也就是说，归约定义在配置 $e|\rho$上，其中 ρ 是一个环境，即从变量到值的有限映射。这样，变量出现会逐个替换为环境中的对应值，而不是一次性全部替换。可以很容易证明，该定义与原始定义等价；而且这种定义便于实现我们的目标，因为在这种表述中，表达式中的自由变量可以自然地视为资源，它们会在执行过程中每次变量出现被使用（替换为其值）时消耗。换言之，这种语义可以通过添加表示资源消耗“成本”的等级来自然地插装，正如我们将在图 2 中所做的那样。除此之外，这些规则都很直接；只需注意，在规则(INVK)和(BLOCK)中，参数（包括 `this`）和局部变量会被重命名为新鲜变量，以避免名称冲突。这里给出单条上下文规则，而非定义求值上下文，以便与插装版本保持一致；对于后者，这种表述更为方便。

XX：6 多重分级 FeatherweightJava

$$
\begin{array}{lll}
e & ::= & x \mid e_{\circ}\ f \mid \texttt{new}\ C(es) \mid e_{\circ}\ m(es) \mid \{T\ x\ = e; e'\} & \text{表达式} \\
T & ::= & C & \text{类型（类名）} \\
v & ::= & \texttt{new}\ C(\text{对比}) & \text{值}
\end{array}
$$

(VAR) $$\frac{}{x|\rho \to v|\rho}\ \rho(x) = v$$

(FIELD-ACCESS) $$\frac{}{\texttt{new}\ C(v_1, \ldots, v_n).f_i|\rho \to v_i|\rho}\ \begin{array}{l}\mathsf{fields}(C) = T_1\ f_1; \ldots T_n\ f_n; \\ i \in 1..n\end{array}$$

(INVK) $$\frac{}{v_0.m(v_1, \ldots, v_n)|\rho \to e[y_0/\texttt{this}][y_1/x_1 \ldots y_n/x_n]|\rho'}\ \begin{array}{l}v_0 = \texttt{new}\ C(_) \\ \mathsf{mbody}(C, m) = \langle x_1 \ldots x_n, e\rangle \\ y_0, \ldots, y_n \notin \mathsf{dom}(\rho) \\ \rho' = \rho, y_0 \mapsto v_0, \ldots, y_n \mapsto v_n\end{array}$$

(BLOCK) $$\frac{}{\{C\ x = v;\ e\}|\rho \to e[y/x]|\rho, y \mapsto v}\ y \notin \mathsf{dom}(\rho)$$

(FIELD-ACCESS-CTX) $$\frac{e|\rho \to e'|\rho'}{e.f|\rho \to e'.f|\rho'}$$

(NEW-CTX) $$\frac{e_i|\rho \to e_i'|\rho'}{\texttt{new}\ C(v_1, \ldots, v_{i-1}, e_i, \ldots, e_n)|\rho \to \texttt{new}\ C(v_1, \ldots, v_{i-1}, e_i', \ldots, e_n)|\rho'}$$

(INVK-RCV-CTX) $$\frac{e_0|\rho \to e_0'|\rho'}{e_0.m(e_1, \ldots, e_n)|\rho \to e_0'.m(e_1, \ldots, e_n)|\rho'}$$

(INVK-ARG-CTX) $$\frac{e_i|\rho \to e_i'|\rho'}{v_0.m(v_1, \ldots, v_{i-1}, e_i, \ldots, e_n)|\rho \to v_0.m(v_1, \ldots, v_{i-1}, e_i', \ldots, e_n)|\rho'}$$

(BLOCK-CTX) $$\frac{e_1|\rho \to e_1'|\rho'}{\{C\ x = e_1;\ e_2\}|\rho \to \{C\ x = e_1';\ e_2\}|\rho'}$$

图1 语法与标准归约

为简洁起见，类表抽象地建模如下，省略其（标准）语法：

- fields(C) 为每个类 C 给出其字段序列 $T_1 f_1; \ldots T_n f_n;$，假设字段名各不相同，并附有各自的类型；mbody(C, m) 为类 C 的每个方法 m 给出其参数和方法体。

3.2 插桩归约

此归约使用等级 *grades*，以 r,s,t 表示，并假设它们构成一个等级代数，指定一个偏序 *partial order* $\preceq$、一个求和 *sum* $+$、一个乘法 *multiplication* $\cdot$ 以及常量 **0** 和 **1**，满足若干公理，详见第 2 节的定义 1。

为了跟踪资源的使用情况（相对于给定的等级代数参数化），我们按如下方式对归约进行插桩。

- 环境会为每个资源（变量）关联一个值，以及一个用于表示其允许使用次数的等级。此外，归约关系是分级的，也就是说，它以等级r为索引，表示其目标是产生一个最多可使用r 次的值；或者，更一般地说（在非定量的意义上），产生一个最多只能以等级r使用的值。每次使用变量时，环境中该变量的等级都会减少归约等级[2]所指定的量。当然，只有当变量当前的等级可以减少这么多时，才能进行这样的操作；否则归约就会卡住。

在给出形式化定义之前，我们先看一些简单的归约示例。这里考虑示例 2(1) 中自然数的等级代数，并追踪资源被使用的次数。

▶ **例 4**。 假设有以下类：

```
class A {}
class Pair {A first; A second}
```

我们用$v_{\texttt{Pair}}$ 作为 `new Pair(new A(),new A())`的缩写。

$\{\texttt{A a} = [\texttt{new A()}]_4; \{\texttt{Pair p} = [\texttt{new Pair(a,a)}]_2; \texttt{new Pair(p.first,p.second)}\}\}|\emptyset \to_1 \{\texttt{Pair p} = [\texttt{new Pair(a,a)}]_2; \texttt{new Pair(p.first,p.second)}\}|\texttt{a} \mapsto \langle \texttt{new A()}, 4i \to_1 \{\texttt{Pair p} = [\texttt{new Pair(new A(),a)}]_2; \texttt{new Pair(p.first,p.second)}\}|\texttt{a} \mapsto \langle \texttt{new A()}, 2i \to_1 \{\texttt{Pair p} = [\texttt{new Pair(new A(),new A())}]_2; \texttt{new Pair(p.first,p.second)}\}|\texttt{a} \mapsto \langle \texttt{new A()}, 0i \to_1 \texttt{new Pair(p.first,p.second)}|\texttt{a} \mapsto \langle \texttt{new A()}, 0i\ \texttt{p} \mapsto \langle v_{\texttt{Pair}}, 2i \to_1 \texttt{new Pair}(v_{\texttt{Pair}}\texttt{.first}, \texttt{p.second})|\texttt{a} \mapsto \langle \texttt{new A()}, 0i\ \texttt{p} \mapsto \langle v_{\texttt{Pair}}, 1i \to_1 \texttt{new Pair(new A(),p.second)}|\texttt{a} \mapsto \langle \texttt{new A()}, 0i\ \texttt{p} \mapsto \langle v_{\texttt{Pair}}, 1i \to_1 \texttt{new Pair(new A(},v_{\texttt{Pair}}\texttt{.second)}|\texttt{a} \mapsto \langle \texttt{new A()}, 0i\ \texttt{p} \mapsto \langle v_{\texttt{Pair}}, 0i \to_1 v_{\texttt{Pair}}|\texttt{a} \mapsto \langle \texttt{new A()}, 0i\ \texttt{p} \mapsto \langle v_{\texttt{Pair}}, 0i$

在这个例子中，顶层归约的等级为 1，这意味着会产生一个值。子项标注了其归约的等级。例如，在外层代码块中，初始化表达式标注为 4，这意味着其结果最多可以使用 4 次。为简化记法，本例省略索引 1。在代码块中引入的局部变量会作为另一项可用资源添加[3] 到环境中，其值和等级与其初始化表达式相同；例如，外层局部变量

[2] 更准确地说，归约等级为该数量提供了一个下界，参见规则 (VAR) 的注释。[3] 为避免冲突而进行的重命名未在示例中展示，以求简洁。

添加时的等级为 4。求值内部块的初始化表达式时，该表达式以等级 2 进行归约，每次使用变量 a 时，环境中该变量的等级都会递减 2。

需要注意的是，子项中的注释并非类型注释。除构造函数调用的参数中的注释（下文将对此进行说明）外，注释的唯一作用是确保子项在每一步都以相同的等级进行归约，详见下文的形式化定义。我们计划在未来的工作中研究一种大步语义表述，以免使用这种变通方法。在上面的例子中，我们为子项的归约选择了能够执行顶层归约的最低等级。我们本可以选择任何更高的等级；相反，如果选择严格更低的等级，归约就会陷入停滞。

如前所述，在构造函数调用 $\texttt{new}\, C([e_1]_{r_1}, \ldots\ldots, [e_n]_{r_n})$ 中，注释 r_i起着特殊作用：直观而言，它指定待构造的对象应包含该字段的r_i 个副本。形式上，这体现在子项 e_i 的归约等级必须恰好为 $r \cdot r_i$，其中r 是对象的归约等级，用于指定归约正在构造该对象的多少个副本。相应地，对该字段的访问最多可以使用$r \cdot r_i$ 次。下面通过前一个例子的一个变体来说明这一点。

▶ **示例 5.** 考虑项

$$\{\texttt{A a} = [\texttt{new A()}]_4; \{\texttt{Pair p} = [\texttt{new Pair(a, a)}]_2; \texttt{new Pair(}\,[\texttt{p.first}]_2\,\texttt{, p.second)}\}\}$$

如灰色部分所示，构造函数调用的第一个参数（即内部块的方法体）现在标注为 2，这意味着生成的对象应有该字段的“两份副本”。因此，表达式 p.first 应按等级 2 进行归约，如下所示，其中 $v_{\texttt{Pair}} = \texttt{new Pair(new A(), new A())}$ 的前四个归约步骤与例 4 中相同，并且为清楚起见，我们明确写出了一些注释 1

$$\{\texttt{A a} = [\texttt{new A()}]_4; \{\texttt{Pair p} = [\texttt{new Pair(}[\texttt{a}]_1\texttt{, a)}]_2; \texttt{new Pair(}\,[[\texttt{p}]_1\texttt{.first}]_2\,\texttt{, p.second)}\}\}|\emptyset \to_1^*$$

$$\texttt{new Pair(}\,[[\texttt{p}]_1\texttt{.first}]_2\,\texttt{, p.second)}|\texttt{a} \mapsto \langle \texttt{new A()}, 0\rangle, \texttt{p} \mapsto \langle v_{\texttt{Pair}}, 2\rangle \to_1$$

$$\texttt{new Pair(}\,[[v_{\texttt{Pair}}]_1\texttt{.first}]_2\,\texttt{, p.second)}|\texttt{a} \mapsto \langle \texttt{new A()}, 0\rangle, \texttt{p} \mapsto \langle v_{\texttt{Pair}}, 1\rangle \quad \text{STUCK}$$

对灰色子项进行归约，目的是构造一个可使用两次的值（new A()），但归约会陷入停滞，因为我们无法从对象 $v_{\texttt{Pair}}$ 的字段 first 中获得 new A() 的两份副本。相反，如果我们选择归约要使用两次的 p，则会得到以下归约，其中同样省略了与之前相同的步骤：

$$\{\texttt{A a} = [\texttt{new A()}]_4; \{\texttt{Pair p} = [\texttt{new Pair(}[\texttt{a}]_1\texttt{, a)}]_2; \texttt{new Pair(}\,[[\texttt{p}]_2\texttt{.first}]_2\,\texttt{, p.second)}\}\}|\emptyset \to_1^*$$

$$\texttt{new Pair(}\,[[\texttt{p}]_2\texttt{.first}]_2\,\texttt{, p.second)}|\texttt{a} \mapsto \langle \texttt{new A()}, 0\rangle, \texttt{p} \mapsto \langle v_{\texttt{Pair}}, 2\rangle \to_1$$

$$\texttt{new Pair(}\,[[v_{\texttt{Pair}}]_2\texttt{.first}]_2\,\texttt{, p.second)}|\texttt{a} \mapsto \langle \texttt{new A()}, 0\rangle, \texttt{p} \mapsto \langle v_{\texttt{Pair}}, 0\rangle \to_1$$

$$\texttt{new Pair(}[\texttt{new A()}]_2\texttt{, p.second)}|\texttt{a} \mapsto \langle \texttt{new A()}, 0\rangle, \texttt{p} \mapsto \langle v_{\texttt{Pair}}, 0\rangle \quad \text{STUCK}$$

在这种情况下，归约会陷入停滞，因为我们已经用尽所有可用的 p 副本来生成字段 first 的两份副本，因此现在无法归约 p.second。若要避免归约陷入停滞，我们应选择归约索引为 3 的 p 的初始化表达式，因此也应选择索引为 6 的 a 的初始化表达式。为了完成 Pair 的构造，也就是得到一个不会陷入停滞的归约，我们应有 p 的 3 份副本，因此需要 a 的 6 份副本。

插桩语义的形式化定义见图 2。为了简化记号，我们沿用图1中标准语义的元变量。如上所述，归约定义在带注释项上。值得注意的是，在每个构造中，上下文规则所归约的子项都带有注释，因此其归约总是在固定等级下进行。

$$
\begin{array}{lll}
e & ::= & x \mid [e]_r.f \mid \texttt{new}\ C([e_1]_{r_1}, \ldots, [e_n]_{r_n}) \mid \\
 & & [e_0]_{r_0} \circ m([e_1]_{r_1}, \ldots, [e_n]_{r_n})es \mid \{T\ x = [e]_r; e'\} \qquad \text{（带注释的）表达式} \\
v & ::= & \texttt{new}\ C([v_1]_{r_1}, \ldots, [v_n]_{r_n}) \qquad \text{（带注释的）值}
\end{array}
$$

（变量） $\dfrac{}{x|\rho, x \mapsto hv,\ si \to_r v|\rho,\ x \mapsto hv,\ s'i}$ $\begin{array}{l} r \preceq r' \neq \mathbf{0} \\ s' + r' \preceq s \end{array}$

（字段访问） $\dfrac{}{[\texttt{new}\ C([v_1]_{r_1}, \ldots, [v_n]_{r_n})]_r.f_i|\rho \to_s v_i|\rho}$ $\begin{array}{l} \text{字段}(C) = T_1 f_1; \ldots T_n f_n; \\ i \in 1..n \\ s \preceq r \cdot r_i \end{array}$

(invk) $\dfrac{}{[v_0]_{r_0}.m([v_1]_{r_1}, \ldots, [v_n]_{r_n})|\rho \to_r e[y_0/\texttt{this}][y_1/x_1 \ldots y_n/x_n]|\rho'}$ $\begin{array}{l} v_0 = \texttt{new}\ C(_) \\ \text{mbody}(C, m) = \langle x_1 \ldots x_n, e \rangle \\ y_0, \ldots, y_n \notin \text{dom}(\rho) \\ \rho' = \rho, y_0 \mapsto hv_0, r_0 i \ldots, y_n \mapsto hv_n, r_n i \end{array}$

(块) $\dfrac{}{\{C\ x = [v]_r; e\}|\rho \to_s e[y/x]|\rho,\ y \mapsto hv,\ ri}$ $y \notin \text{dom}(\rho)$

(字段访问-上下文) $\dfrac{e|\rho \to_r e'|\rho'}{[e]_r.f|\rho \to_s [e']_r.f|\rho'}$

（新上下文） $\dfrac{e_i|\rho \to_{r \cdot r_i} e_i'|\rho'}{\begin{array}{l}\texttt{new}\ C([v_1]_{r_1}, \ldots, [v_{i-1}]_{r_{i-1}}, [e_i]_{r_i}, \ldots, [e_n]_{r_n})|\rho \to_r \\ \qquad \texttt{new}\ C([v_1]_{r_1}, \ldots, [v_{i-1}]_{r_{i-1}}, [e_i']_{r_i}, \ldots, [e_n]_{r_n})|\rho'\end{array}}$

(invk-rcv-ctx) $\dfrac{e_0|\rho \to_{r_0} e_0'|\rho'}{[e_0]_{r_0} \circ m([e_1]_{r_1}, \ldots, [e_n]_{r_n})|\rho \to_r [e_0']_{r_0} \circ m([e_1]_{r_1}, \ldots, [e_n]_{r_n})|\rho'}$

(invk-arg-ctx) $\dfrac{e_i|\rho \to_{r_i} e_i'|\rho'}{\begin{array}{l}[e_0]_{r_0} \circ m([v_1]_{r_1}, \ldots, [v_{i-1}]_{r_{i-1}}, [e_i]_{r_i}, \ldots, [e_n]_{r_n})|\rho \to_r \\ \qquad [e_0]_{r_0} \cdot m([v_1]_{r_1}, \ldots, [v_{i-1}]_{r_{i-1}}, [e_i']_{r_i}, \ldots, [e_n]_{r_n})|\rho'\end{array}}$

(块上下文) $\dfrac{e_1|\rho \to_s e_1'|\rho'}{\{C\ x = [e_1]_s; e_2\}|\rho \to_r \{C\ x = [e_1']_s; e_2\}|\rho'}$

图 2 插桩归约

XX:10 多重分级 FeatherweightJava

在规则 (VAR) 中，资源被消耗的关键规则是：将变量出现替换为环境中关联的值，其等级 s 降低至 s', ，消耗非零数量 r' 的资源，且该数量必须至少等于归约等级。侧条件 $r' + s' \preceq s$ 确保变量的初始等级足以同时涵盖消耗等级和剩余等级。为说明资源消耗量为何必须非零，请考虑以下例 4 的变体：

$$\{\texttt{A a} = [\texttt{new A()}]_4; \{\texttt{Pair p} = [\texttt{new Pair(a, a)}]_0; \texttt{new Pair(a, a)}\}\}|\emptyset$$

代码块的方法体中从未使用局部变量 p，因此将其初始化表达式按等级 0 进行归约是合理的，因为执行无需复制结果。然而，表达式 需要进行归约，而为了产生这个无用的结果，会消耗两个 a 副本；从某种意义上说，它们被浪费了。不过，这种资源使用会被追踪；若递减量为 0，则会丢失这一信息。取消非零要求会导致一种资源感知规约变体，在这种变体中，不会追踪用于构造最终结果时毫无用处的资源使用。

在规则 （字段访问）中，归约等级应为接收者的等级与字段（构造器参数）的等级之积（的上界）。实际上，前者说明我们拥有该对象的多少个副本，后者说明每个此类对象拥有多少个字段副本；因此，它们的积为结果值的等级提供了一个上界。请注意，这样一来，某些归约可能会被禁止。例如，取自然数的等级代数，若某字段的值可使用 3 次，而其所属对象以等级 2 归约，那么该字段访问最多可归约 6 次。下面以隐私级别的等级代数为例，给出一个更有意义的例子。

规则 （invk）会将每个方法参数（包括 this）都作为可用资源加入环境，并通过重命名为一个新鲜变量来避免冲突。关联的值和等级分别与对应参数的值和等级相同。规则（block）与此完全类似，区别仅在于它只添加一个变量。

说到上下文规则，子项的归约等级与对应注释的等级相同，因此所有步骤都在固定等级下进行。唯一的例外是规则 (NEW-CTX)。与规则 (FIELD-ACCESS) 对称，子项的归约等级应为对象的归约等级与字段（构造器参数）注释的乘法结果，这体现了这样一种直觉：后者指定了该字段对于对象的单个副本所具有的等级。例如，若采用自然数的等级代数，要构造一个可使用两次、且带有一个可使用 3 次的字段的对象，那么该字段的值应是一个可使用 6 次的对象。

请注意，除了诸如查找缺失的方法或字段之类的标准类型错误外，带归约等级的 r 也可能陷入停滞：规则 (VAR) 可能因侧条件不成立而无法应用；或者规则(FIELD-ACCESS) 可能因侧条件$s \preceq r \cdot r_i$ 不成立而无法应用。通俗地说，这可能是因为某种资源（变量）已耗尽，也就是说，无法再用其值替换；也可能是因为某个对象的某个字段无法提取。还需注意，插桩归约是非确定性的，这是由规则 (VAR)导致的。

在前一个示例所用的等级代数中，等级表示资源的使用次数。不过，等级也可以表示非定量的[4] 知识，也就是说，追踪资源可能的使用模式，换言之，追踪对资源使用方式可能施加的约束。这种情况的一个典型例子是隐私级别，其形式化方式与 [1], 中所述类似。

[4] 这种应用被称为 [1] 中的信息型 应用。

▶ **例6**。 从任意分配半格格 L 出发（如示例 2(5)中所示），定义 $\mathbf{L}_0 = \langle |\mathbf{L}_0|, \leq_0, \vee_0, \wedge_0, 0, \top \rangle$，其中 $|\mathbf{L}_0| = |\mathbf{L}| + \{0\}$ 满足 $0 \leq_0 x$, $x \vee_0 0 = 0 \vee_0 x = x$且 $x \wedge_0 0 = 0 \wedge_0 x = 0$，对所有 $x \in |\mathbf{L}|$均成立；在 $|\mathbf{L}|$ 的元素上，序和运算与 L 中的相同。也就是说，我们假设隐私级别构成一个分配半格，其序表示“隐私递减”，并添加一个表示“未使用”的等级 0。最简单的情况只有两个隐私级别，即 $0 \preceq$私有 $\preceq$公开。加法是并，表示我们得到的隐私级别比两者都宽松：例如，一个变量在某个子项中作为公开使用，在另一个子项中作为私有使用，那么整体上它作为公开使用。乘法是交，表示我们得到的隐私级别比两者都严格：例如，访问某个字段时，若其值是在`public` 模式下获得的，而对象是在 `private` 模式下简化的，那么访问就在 `private` 模式下简化[5]。请注意，完全相同的结构也可用于建模其他情形，例如，不采用隐私级别，而是在命令式环境中使用只读和可变修饰符，分别对应禁止字段赋值和不施加限制。以下示例展示了这种等级代数的用法。我们用 priv 和 pub 作为简写，而类 `A` 和 `Pair` 与前面的示例相同。

1. 令 $e_1 = \{\texttt{A y} = [\texttt{new A()}]_{\mathsf{pub}}; \{\texttt{A x} = [\texttt{y}]_{\mathsf{priv}}; \texttt{x}\}\}$ 和 `p_` 分别为 公开 或 私有，e_1 从空环境开始，按等级归约私有 如下：

$$
\begin{array}{rll}
e_1|\emptyset & \to_{\mathsf{priv}} & \{\texttt{A x} = [\texttt{y}]_{\mathsf{priv}}; \texttt{x}\}|\texttt{y} \mapsto \langle \texttt{new A()}, \mathsf{pub}\rangle \text{ with (BLOCK)} \\
& \to_{\mathsf{priv}} & \{\texttt{A x} = [\texttt{new A()}]_{\mathsf{priv}}; \texttt{x}\}|\texttt{y} \mapsto \langle \texttt{new A()}, \texttt{p_}\rangle \text{ with (BLOCK-CTX) and} \\
& & \texttt{y}|\texttt{y} \mapsto \langle \texttt{new A()}, \mathsf{pub}\rangle \to_{\mathsf{priv}} \texttt{new A()}|\texttt{y} \mapsto \langle \texttt{new A()}, \texttt{p_}\rangle \\
& \to_{\mathsf{priv}} & \texttt{x}|\texttt{y} \mapsto \langle \texttt{new A()}, \texttt{p_}\rangle, \texttt{x} \mapsto \langle \texttt{new A()}, \mathsf{priv}\rangle \text{ with (BLOCK)} \\
& \to_{\mathsf{priv}} & \texttt{new A()}|\texttt{y} \mapsto \langle \texttt{new A()}, \texttt{p_}\rangle, \texttt{x} \mapsto \langle \texttt{new A()}, \mathsf{priv}\rangle \text{ with (VAR)}
\end{array}
$$

相反，按公开等级归约会卡住，因为 $\mathsf{pub} \not\preceq \mathsf{priv}$，因此

$$\texttt{x}|\texttt{y} \mapsto \langle \texttt{new A()}, \texttt{p_}\rangle, \texttt{x} \mapsto \langle \texttt{new A()}, \mathsf{priv}\rangle \not\to_{\mathsf{pub}}$$

此外，按私有等级归约$e_2 = \{\texttt{A y} = [\texttt{new A()}]_{\mathsf{priv}}; \{\texttt{A x} = [\texttt{y}]_{\mathsf{pub}}; \texttt{x}\}\}$ 时

$$
\begin{array}{rll}
e_2|\emptyset & \to_{\mathsf{priv}} & \{\texttt{A x} = [\texttt{y}]_{\mathsf{pub}}; \texttt{x}\}|\texttt{y} \mapsto \langle \texttt{new A()}, \mathsf{priv}\rangle \text{ with (BLOCK)} \\
& \not\to_{\mathsf{priv}} &
\end{array}
$$

会卡住，因为 $\texttt{y}|\texttt{y} \mapsto \langle \texttt{new A()}, \mathsf{priv}\rangle \not\to_{\mathsf{pub}}$。注意，$e_1$ 和e_2 都会根据图1的语义归约为`new A()` 。

2. 令 $e_3 = \{\texttt{A x} = [\texttt{new A()}]_{\mathsf{pub}}; \texttt{new Pair}([\texttt{x}]_{\mathsf{pub}}, [\texttt{x}]_{\mathsf{priv}})\}$，$e_3$ 从空环境开始以等级公开 归约如下：

$$
\begin{array}{rll}
e_3|\emptyset & \to_{\mathsf{pub}} & \texttt{new Pair}([\texttt{x}]_{\mathsf{pub}}, [\texttt{x}]_{\mathsf{priv}})|\texttt{x} \mapsto \langle \texttt{new A()}, \mathsf{pub}\rangle \text{ with (BLOCK)} \\
& \to_{\mathsf{pub}} & \texttt{new Pair}([\texttt{new A()}]_{\mathsf{pub}}, [\texttt{x}]_{\mathsf{priv}})|\texttt{x} \mapsto \langle \texttt{new A()}, \texttt{p_}\rangle \text{ with (NEW-CTX) and} \\
& & \texttt{x}|\texttt{x} \mapsto \langle \texttt{new A()}, \mathsf{pub}\rangle \to_{\mathsf{pub}} \texttt{new A()}|\texttt{x} \mapsto \langle \texttt{new A()}, \texttt{p_}\rangle \\
& \to_{\mathsf{pub}} & \texttt{new Pair}([\texttt{new A()}]_{\mathsf{pub}}, [\texttt{new A()}]_{\mathsf{priv}})|\texttt{x} \mapsto \langle \texttt{new A()}, \texttt{p_}\rangle \text{ with (NEW-CTX) and} \\
& & \texttt{x}|\texttt{x} \mapsto \langle \texttt{new A()}, \texttt{p_}\rangle \to_{\mathsf{priv}} \texttt{new A()}|\texttt{x} \mapsto \langle \texttt{new A()}, \texttt{p_}\rangle
\end{array}
$$

不难看出，$e_3|\emptyset \to^*_{\mathsf{priv}} \texttt{new Pair}([\texttt{new A()}]_{\mathsf{pub}}, [\texttt{new A()}]_{\mathsf{priv}})|\texttt{x} \mapsto \langle \texttt{new A()}, \texttt{p_}\rangle$也成立。因此我们有

$$[e_3]_r.\texttt{f}|\emptyset \quad \to^*_s \quad [\texttt{new Pair}([\texttt{new A()}]_{\mathsf{pub}}, [\texttt{new A()}]_{\mathsf{priv}})]_r.\texttt{f}|\texttt{x} \mapsto \langle \texttt{new A()}, \texttt{p_}\rangle$$

[5] 与视点适配 [12], 一样，字段访问的权限可以根据基础对象的权限加以限制。

其中， f 可以是 first 或 second ，而r 和s 可以是 pub 或 priv。现在，等级为 priv、访问 first 或 second 的归约会产生这些字段的值

$$[\texttt{new Pair}([\texttt{new A()}]_{\mathsf{pub}}, [\texttt{new A()}]_{\mathsf{priv}})]_r.\texttt{f}|_ \rightarrow_{\mathsf{priv}} \texttt{new A()}|_$$

然而，只考察等级为 pub 的归约时，只有

$$[\texttt{new Pair}([\texttt{new A()}]_{\mathsf{pub}}, [\texttt{new A()}]_{\mathsf{priv}})]_{\mathsf{pub}}.\texttt{first}|_ \rightarrow_{\mathsf{pub}} \texttt{new A()}|_$$

不会陷入停滞。也就是说，只有当我们取得一个公开对象的公有字段时，才能生成可用作公开值的值；而任何值都可以用作私有值。

现在我们陈述一些语义的简单性质，并将用它们证明类型可靠性。前者确立了归约不会从环境中移除变量，后者则指出，我们总能降低规约步的等级。

▶ **命题 7**。 若 $e|\rho \rightarrow_r e'|\rho'$ ，则 $dom(\rho) \subseteq dom(\rho')$ ，并且对于所有 $x \in dom(\rho)$， $\rho(x) = hv,ri$ 蕴含 $\rho'(x) = hv, si$，其中 $s \preceq r$。

▶ **命题 8**。 若 $e|\rho \rightarrow_r e'|\rho'$ 且 $s \preceq r$ 则 $e|\rho \rightarrow_s e'|\rho'$。

我们期望插桩归约相对于标准归约是可靠的，也就是说，对插桩归约序列擦除注释后，会得到一个标准归约序列。下面将对此作形式化表述。

对于任意表达式 e，令$de\rceil$ 表示擦除注释后得到的表达式，具体定义方式显而易见。对于环境也作类似定义，同时移除与变量关联的等级。

▶ **命题 9 （带注释语义的可靠性：** 若 $e|\rho \rightarrow_r e'|\rho'$，则 $\lceil e\rceil|\lceil\rho\rceil \rightarrow \lceil e'\rceil|\lceil\rho'\rceil$。

反向结论并不成立，因为配置可能以某种方式添加注释，从而导致其陷入停滞；特别是，某些资源（变量）可能已耗尽，或者对象的某个字段可能无法提取。下一节中的分级类型系统将生成能够确保可靠性的注释，因此相对于标准归约也能确保完备性。

4 分级 Featherweight Java

我们定义参数化资源感知类型系统，展示一些示例，并证明其可靠性。

4.1 分级类型系统

类型（类名）标注有等级，如图 3 所示。

正如第 2 节末尾所预期的，一个形如 $\gamma = \mathrm{x}_1 : r_1, \ldots, \mathrm{x}_n : r_n$ 的共效上下文，其中顺序无关紧要，且对于 $i \neq j$ 有 $\mathrm{x}_i \neq \mathrm{x}j$ ，表示从变量到等级的映射（在此位置使用时称为共效应），其中只有有限个变量具有非零共效应。一个（类型与共效应）上下文，形如 $\Gamma = \mathrm{x}_1 :_{r_1} \mathrm{C1}, \ldots, \mathrm{x}_n :_{r_n} \mathrm{C}_n$，采用类似约定，表示标准类型上下文 $\mathrm{x}_1 : \mathrm{C1} \ldots, \mathrm{x}_n : \mathrm{C}_n$ 与共效上下文 $\mathrm{x}_1 : r_1, \ldots, \mathrm{x}_n : r_n$ 所组成的一对。我们用 dom(Γ) 表示 $\{\mathrm{x1}, \ldots, \mathrm{x}_n\}$。

正如类型与共效应系统中的惯例，在类型规则中，上下文通过某些操作进行组合，而这些操作又根据共效应（等级）上的相应操作来定义。

更准确地说，我们定义：

$$
\begin{array}{lcll}
e & ::= & x \mid e.f \mid \texttt{new}\ C(es) \mid e.m(es) \mid \{T\,x = e;\, e'\} & \text{expression} \\
T & ::= & C^r & \text{(graded) type} \\
v & ::= & \texttt{new}\ C(vs) & \text{value}
\end{array}
$$

图 3 带等级的语法

- 一个偏序 $\preceq$

$$
\begin{array}{ll}
\emptyset \preceq \emptyset & \\
x :_s C, \Gamma \preceq x :_r C, \Delta & \text{if } s \preceq r \text{ and } \Gamma \preceq \Delta \\
\Gamma \preceq x :_r C, \Delta & \text{if } x \notin \mathsf{dom}(\Gamma) \text{ and } \Gamma \preceq \Delta
\end{array}
$$

- 一个和 $+$

$$
\begin{array}{ll}
\emptyset + \Gamma = \Gamma & \\
(x :_s C, \Gamma) + (x :_r C, \Delta) = x :_{s+r} C, (\Gamma + \Delta) & \\
(x :_s C, \Gamma) + \Delta = x :_s C, (\Gamma + \Delta) & \text{如果 } x \notin \mathsf{dom}(\Delta)
\end{array}
$$

- 一个数乘 $\cdot$

$$
s \cdot \emptyset = \emptyset \qquad\qquad s \cdot (x :_r C, \Gamma) = x :_{s \cdot r} C, (s \cdot \Gamma)
$$

读者可能会注意到，这些针对类型与共效上下文的操作，也可以等价地定义为：将共效上下文上的相应操作（即对共效操作的逐点扩张）提升为同样适用于类型的操作。在这一步中，加法变成了偏运算，因为两个上下文定义域中都出现的变量必须具有相同类型。

类型系统依赖于从类表中提取的类型信息；为简洁起见，这些信息同样抽象地建模如下：

- 类名上的子类型关系 $\leq$ 是`extends` 关系的自反传递闭包；mtype(C，m)为类C的每个方法m给出其扩充的方法类型，其中参数的类型以及 `this` 都带有副效应注释。

此外，fields(C)现在给出一个序列$C_1^{r_1} f_1 ; \ldots C_n^{r_n} f_n$;，这意味着要构造一个类型为$C$的对象，我们需要为每个 $i \in 1..n$提供一个等级至少为r_i的值。分级类型上的子类型关系定义如下：

$$C^r \leq D^s \text{ iff } C \leq D \text{ and } s \preceq r$$

也就是说，如果一个分级类型的类是另一个类的子类，且其等级约束更严格，那么它就是另一个分级类型的子类型。例如，采用示例 2(2)中的亲和性等级代数，返回类型为C^ω的方法调用可以用于需要类型C^1的上下文中，例如用于初始化C^1 变量。

类型判断的形式为 $\Gamma \vdash e : T \rightsquigarrow e'$, ，其中 Γ 是类型与共效上下文，而e' 是带注释的表达式，定义见图 2。也就是说，类型检查会在代码中生成注释，从而确保求值不会陷入停滞，下面将对此进行形式化表述并证明。

在良类型的类表中，方法体应符合方法类型。也就是说，mtype(C, m) 和 mbody(C, m) 应当要么都未定义，要么都已定义且参数数量相同。在后一种情况下，方法体应当是良类型的

XX：14 多重分级 Featherweight Java

$$\text{(T-SUB)}\ \frac{\Gamma \vdash e : T \rightsquigarrow e' \quad \Gamma \preceq \Gamma'}{\Gamma' \vdash e : T' \rightsquigarrow e' \quad T \leq T'} \qquad \text{(T-VAR)}\ \frac{}{x :_r C \vdash x : C^r \rightsquigarrow x}\ r \neq \mathbf{0}$$

$$\text{(T-FIELD-ACCESS)}\ \frac{\Gamma \vdash e : C^r \rightsquigarrow e'}{\Gamma \vdash e.f_i : C_i^{r \cdot r_i} \rightsquigarrow [e']_r.f_i}\ \mathsf{fields}(C) = C_1^{r_1}\ f_1; \ldots C_n^{r_n}\ f_n;$$

$$\text{(T-NEW)}\ \frac{\Gamma_i \vdash e_i : C_i^{r \cdot r_i} \rightsquigarrow e_i' \quad \forall i \in 1..n}{\Gamma_1 + \ldots + \Gamma_n \vdash \texttt{new}\ C(e_1, \ldots, e_n) : C^r \rightsquigarrow \texttt{new}\ C([e_1']_{r_1}, \ldots, [e_n']_{r_n})}\ \mathsf{fields}(C) = C_1^{r_1}\ f_1; \ldots C_n^{r_n}\ f_n;$$

$$\text{(T-INVK)}\ \frac{\Gamma_0 \vdash e_0 : C^{r_0} \rightsquigarrow e_0' \quad \Gamma_i \vdash e_i : C_i^{r_i} \rightsquigarrow e_i' \quad \forall 1 \in 1..n}{\Gamma_0 + \ldots + \Gamma_n \vdash e_0.m(e_1, \ldots, e_n) : T \rightsquigarrow [e_0']_{r_0}.m([e_1]_{r_1}, \ldots, [e_n]_{r_n})}\ \mathsf{mtype}(C, m) = r_0, C_1^{r_1} \ldots C_n^{r_n} \to T$$

$$\text{(T-BLOCK)}\ \frac{\Gamma_1 \vdash e_1 : C^r \rightsquigarrow e_1' \quad \Gamma_2, x :_r C \vdash e_2 : T \rightsquigarrow e_2'}{\Gamma_1 + \Gamma_2 \vdash \{C^r\ x = e_1;\ e_2\} : T \rightsquigarrow \{C\ x = [e_1']_r;\ e_2'\}}$$

$$\text{(T-ENV)}\ \frac{\vdash v_i : C_i^{r_i} \rightsquigarrow v_i' \quad \forall i \in 1..n}{\Gamma \vdash \rho \rightsquigarrow \rho'}\ \begin{array}{l}\Gamma = x_1 :_{r_1} C_1, \ldots, x_n :_{r_n} C_n \\ \rho = x_1 \mapsto \langle v_1, r_1 \rangle, \ldots, x_n \mapsto \langle v_n, r_n \rangle \\ \rho' = x_1 \mapsto \langle v_1', r_1 \rangle, \ldots, x_n \mapsto \langle v_n', r_n \rangle\end{array}$$

$$\text{(T-CONF)}\ \frac{\Delta \vdash e : T \rightsquigarrow e' \quad \Gamma \vdash \rho \rightsquigarrow \rho'}{\Gamma \vdash e|\rho : T \rightsquigarrow e'|\rho'}\ \Delta \preceq \Gamma$$

图4 分级类型系统

相对于方法类型而言，尤其是对方法体进行类型检查时，得到的共效应应当包含在注释所指定的共效应之中。形式化地说，若 $\mathsf{mbody}(C, m) = \langle x_1 \ldots x_n, e \rangle$，且 $\mathsf{mtype}(C, m) = r_0, C_{r_1}{}^1 \ldots C_{r_n}{}^n \to T$，则必须满足以下条件：

$$\text{(T-METH)}\ \texttt{this} :_{r_0} C, x_1 :_{r_1} C_1, \ldots, x_n :_{r_n} C_n \vdash e : T \rightsquigarrow e'$$

此外，我们假设类表满足关于继承的标准相干性条件。也就是说，若 $C \leq D$，则 $\mathsf{fields}(D)$ 是 $\mathsf{fields}(C)$ 的前缀；并且，若 $\mathsf{mtype}(C, m) = r_0, C_{r_1}{}^1 \ldots C_{r_n}{}^n \to T$，则 $\mathsf{mtype}(D, m) = r_0, C_{r_1}{}^1 \ldots C_{r_n}{}^n \to T'$，其中 $T' \leq T$。

在图4中，我们描述了类型规则，它们以底层等级代数为参数化。

在规则 (T-SUB) 中，共效上下文和（分级）类型都可以推广。这意味着，一方面，变量可以获得限制较少的共效应。例如，仍以仿射共效应为例，若某个表达式在假设最多使用给定变量一次（共效应 1）的情况下可以通过类型检查，那么在不施加任何约束（共效应 ω）的情况下也可以通过类型检查。另一方面，回想一下，等级在类型上是逆变的，因此表达式可以获得约束更多的等级。例如，等级为 ω 的表达式可以用于要求等级为 1 的场合。

若取 $r = \mathbf{1}$，则规则 (T-VAR) 类似于共效应系统中关于变量的标准规则，其中，共效上下文是一个映射：给定变量被使用一次，而其他变量均未被使用。更一般地，在此变量可以获得任意等级r，前提是它在上下文中获得相同的等级。然而，变量的使用不能被直接丢弃，这由侧条件 $r \neq \mathbf{0}$ 表示。

在规则 (T-FIELD-ACCESS) 中，字段的等级与接收者的等级相乘。

如前所述，这是视点适配的一种形式 [12]。例如，使用亲和等级时，等级为 ω 的对象的字段最多只能使用一次。

在规则 （t-new）中，与规则 （t-var）类似，构造函数调用可以具有任意等级r，前提是字段的等级也乘以相同的等级。子项的共效应相加，这与类型和共效应系统中的惯例一致。

在规则 （t-invk）中，参数的共效应也会相加。该规则使用类表上的函数 mtype；给定一个类名和一个方法名，该函数会返回其参数类型和返回类型（带等级）。对于隐式参数 `this`，只指定等级。请注意，参数的等级有两种不同的用途：

- 作为类型（的一部分），用于对参数进行类型检查
- 作为共效应，用于对方法体进行类型检查。

在规则 (T-BLOCK) 中，初始化表达式的共效应与方法体的共效应相加，但不包括局部变量。与方法参数类似，局部变量的等级既用作类型的一部分（在对初始化表达式进行类型检查时），也用作副效应（在对方法体进行类型检查时）。

最后，我们给出类型环境和配置的直接规则。由于我们采用按值调用演算，环境中的值被假定为闭合的。另请注意，在环境和配置的判断中，由于没有可用的子类型规则，上下文中的变量恰好是环境定义域中的变量，而后者是表达式中所用变量的超集。

4.2 示例

▶**示例 10.** 我们通过一个简单示例展示分级类型的用法，并假设采用亲和等级。我们用方括号标出隐式 `this` 参数的等级。类 `Pair` 为 `first` 字段声明了三个版本的取值器，它们的区别在于结果的等级：要么为 0，表示方法的结果不能使用；要么为 1，表示最多可以使用一次；要么为 ω，表示使用不受限制。请注意，第一个版本显然在函数式演算中毫无用处，但若加入效应，例如在命令式演算中，则可能有意义，其作用类似于 `void`。

```
class Pair { A¹ first; A¹ second;
  A⁰ getFirstZero() [1]{this.first}
  A¹ getFirstAffine() [1]{this.first}
  Aᵠ getFirst() [1]{this.first}
}
```

`this` 的副效应在所有版本中均为 1，并且实际上在方法体中使用了一次。得益于规则 (T-VAR)，方法体中 `this` 的出现可以获得任意非零等级；字段的等级为 1，这意味着字段访问不会影响接收者的等级，因此这三个方法体同样可以获得任意非零等级，所以相对于方法返回类型中的等级而言，它们都是类型正确的。

在下面的客户端代码中，取值器的调用结果被赋值给一个具有相同等级的局部变量，随后该变量始终以符合该等级的方式使用。

```
Pair¹ p = ...

{A⁰ a = p.getFirstZero(); new Pair(new A(),new A())}
{A¹ a = p.getFirstAffine(); new Pair(a,new A())}
{Aᵠ a = p.getFirst(); new Pair(a,a)}
```

而下面的代码块则因两个不同的原因而类型不正确。

```
{A[1] a = p.getFirst(); new Pair(a,a)}
{A[ω] a = p.getFirstAffine(); new Pair(a,a)}
```

在第一个例子中，根据子类型包含，初始化是正确的，因为我们使用了一个等级约束较少的表达式。然而，随后变量的使用方式与其等级不兼容。而在第二个例子中，变量的使用方式与其等级一致，但初始化的类型不正确，因为我们使用了一个等级约束较多的表达式。

最后请注意，在这三个方法中，可以将 `this` 的副效应安全地改为 ω，从而提供过近似信息；但在这种情况下，客户端代码中的三次调用都是错误的，因为接收者 `p` 必须至多使用一次。

▶ **示例 11.** 考虑示例 5 中表达式的以下源代码（即未标注）版本。

```
{A^public y = new A(); {A^private x = y; x}}
```

私有变量 `x` 使用公有表达式/变量 `y` 进行初始化。块表达式的类型为 $\mathtt{A}^{\mathsf{private}}$，如下列类型推导所示。

$$\text{(T-BLOCK)}\ \frac{\text{(T-NEW)}\ \dfrac{}{\vdash \mathtt{new\ A()} : \mathtt{A}^{\mathsf{pub}}} \quad \mathcal{D}}{\vdash \{\mathtt{A}^{\mathsf{pub}}\ \mathtt{y} = \mathtt{new\ A()};\ \{\mathtt{A}^{\mathsf{priv}}\ \mathtt{x} = \mathtt{y};\ \mathtt{x}\}\} : \mathtt{A}^{\mathsf{priv}}}$$

其中 $\mathcal{D}$ 是如下推导

$$\text{(T-BLOCK)}\ \frac{\text{(T-SUB)}\ \dfrac{\text{(T-VAR)}\ \dfrac{}{\mathtt{y} :_{\mathsf{pub}} \mathtt{A} \vdash \mathtt{y} : \mathtt{A}^{\mathsf{pub}}}}{\mathtt{y} :_{\mathsf{pub}} \mathtt{A} \vdash \mathtt{y} : \mathtt{A}^{\mathsf{priv}}} \quad \text{(T-VAR)}\ \dfrac{}{\mathtt{y} :_{\mathsf{pub}} \mathtt{A}, \mathtt{x} :_{\mathsf{priv}} \mathtt{A} \vdash \mathtt{x} : \mathtt{A}^{\mathsf{priv}}}}{\mathtt{y} :_{\mathsf{pub}} \mathtt{A} \vdash \{\mathtt{A}^{\mathsf{priv}}\ \mathtt{x} = \mathtt{y};\ \mathtt{x}\} : \mathtt{A}^{\mathsf{priv}}}$$

另一方面，像下面这样用私有表达式初始化公有变量

```
{A^private y = new A(); {A^public x = y; x}}
```

并不可行，这符合预期，因为 $\mathtt{y} :_{\mathsf{priv}} \mathtt{A} \not\vdash \mathtt{y} : \mathtt{A}^{\mathsf{pub}}$。

现在考虑类 `Pair`，它包含一个私有字段和一个公有字段。

```
class B { A^public f1; A^private f2; }
```

表达式 e

```
{A^public x = new A(); new B(x,x)}
```

可以按如下方式赋予类型 $\mathtt{Pair}^{\mathsf{public}}$：

$$\text{(T-BLOCK)}\ \frac{\text{(T-NEW)}\ \dfrac{}{\vdash \mathtt{new\ A()} : \mathtt{A}^{\mathsf{pub}}} \quad \text{(T-NEW)}\ \dfrac{\text{(T-VAR)}\ \dfrac{}{\mathtt{x} :_{\mathsf{pub}} \mathtt{A} \vdash \mathtt{x} : \mathtt{A}^{\mathsf{pub}}} \quad \text{(T-SUB)}\ \dfrac{\text{(T-VAR)}\ \dfrac{}{\mathtt{x} :_{\mathsf{pub}} \mathtt{A} \vdash \mathtt{x} : \mathtt{A}^{\mathsf{pub}}}}{\mathtt{x} :_{\mathsf{pub}} \mathtt{A} \vdash \mathtt{x} : \mathtt{A}^{\mathsf{priv}}}}{\mathtt{x} :_{\mathsf{pub}} \mathtt{A} \vdash \mathtt{new\ Pair(x,x)} : \mathtt{Pair}^{\mathsf{pub}}}}{\vdash \{\mathtt{A}^{\mathsf{pub}}\ \mathtt{x} = \mathtt{new\ A()};\ \mathtt{new\ Pair(x,x)}\} : \mathtt{Pair}^{\mathsf{pub}}}$$

通过(T-SUB)，我们还可以推导出$\vdash e : \mathtt{Pair}^{\mathsf{priv}}$，因此得到

$$\text{(T-FIELD)}\ \frac{\vdash e : \mathtt{Pair}^{\mathsf{priv}}}{\vdash e.\mathtt{first} : \mathtt{A}^{\mathsf{priv}}} \qquad \text{(T-FIELD)}\ \frac{\vdash e : \mathtt{Pair}^{\mathsf{pub}}}{\vdash e.\mathtt{second} : \mathtt{A}^{\mathsf{priv}}}$$

也就是说，访问私有表达式的公有字段会得到私有结果，访问公有表达式的私有字段也会得到私有结果。
另请注意，以下表达式 e'

$$\text{(T-SUB)}\ \frac{\Gamma \vdash_a e : T \quad \Gamma \preceq \Gamma'}{\Gamma' \vdash_a e : T' \quad T \leq T'} \qquad \text{(T-VAR)}\ \frac{}{x :_r C \vdash_a x : C^r}\ r \neq \mathbf{0}$$

$$\text{(T-FIELD-ACCESS)}\ \frac{\Gamma \vdash_a e : C^r}{\Gamma \vdash_a [e]_r.f_i : C_i^{r \cdot r_i}}\ \mathsf{fields}(C) = C_1^{r_1}\ f_1; \ldots C_n^{r_n}\ f_n;$$

$$\text{(T-NEW)}\ \frac{\Gamma_i \vdash_a e_i : C_i^{r \cdot r_i} \quad \forall i \in 1..n}{\Gamma_1 + \ldots + \Gamma_n \vdash_a \texttt{new}\ C([e_1]_{r_1}, \ldots, [e_n]_{r_n}) : C^r}\ \mathsf{fields}(C) = C_1^{r_1}\ f_1; \ldots C_n^{r_n}\ f_n;$$

$$\text{(T-INVK)}\ \frac{\Gamma_0 \vdash_a e_0 : C^{r_0} \quad \Gamma_i \vdash_a e_i : C_i^{r_i} \quad \forall 1 \in 1..n}{\Gamma_0 + \ldots + \Gamma_n \vdash_a [e_0]_{r_0}.m([e_1]_{r_1}, \ldots, [e_n]_{r_n}) : T}\ \mathsf{mtype}(C, m) = r_0, C_1^{r_1} \ldots C_n^{r_n} \to T$$

$$\text{(T-BLOCK)}\ \frac{\Gamma_1 \vdash_a e_1 : C^r \quad \Gamma_2, x :_r C \vdash_a e_2 : T}{\Gamma_1 + \Gamma_2 \vdash_a \{C\ x = [e_1]_r; e_2\} : T}$$

$$\text{(T-ENV)}\ \frac{\vdash_a v_i : C_i^{r_i} \quad \forall i \in 1..n}{\Gamma \vdash_a \rho}\ \begin{array}{l}\Gamma = x_1 :_{r_1} C_1, \ldots, x_n :_{r_n} C_n \\ \rho = x_1 \mapsto \langle v_1, r_1 \rangle, \ldots, x_n \mapsto \langle v_n, r_n \rangle\end{array}$$

$$\text{(T-CONF)}\ \frac{\Delta \vdash_a e : T \quad \Gamma \vdash_a \rho}{\Gamma \vdash_a e|\rho : T}\ \Delta \preceq \Gamma$$

图 5 带注释语法的分级类型系统

```
{A^private x = new A(); new B(x,x)}
```

只能赋予类型 $\texttt{Pair}^{\mathsf{private}}$ by

$$\text{(T-BLOCK)}\ \frac{\text{(T-NEW)}\ \dfrac{}{\vdash \texttt{new A()} : \texttt{A}^{\mathsf{priv}}} \quad \text{(T-NEW)}\ \dfrac{\text{(T-VAR)}\ \dfrac{}{\texttt{x} :_{\mathsf{priv}} \texttt{A} \vdash \texttt{x} : \texttt{A}^{\mathsf{priv}}} \quad \text{(T-VAR)}\ \dfrac{}{\texttt{x} :_{\mathsf{priv}} \texttt{A} \vdash \texttt{x} : \texttt{A}^{\mathsf{priv}}}}{\texttt{x} :_{\mathsf{priv}} \texttt{A} \vdash \texttt{new Pair(x,x)} : \texttt{Pair}^{\mathsf{priv}}}}{\vdash \{\texttt{A}^{\mathsf{priv}}\ \texttt{x} = \texttt{new A(); new Pair(x,x)}\} : \texttt{Pair}^{\mathsf{priv}}}$$

我们无法推导 $\vdash e' : \texttt{Pair}^{\mathsf{pub}}$，因为 `first` 的等级是公开的，而 (T-NEW) 要求 $\texttt{x} :_{\mathsf{priv}} \texttt{A} \vdash \texttt{x} : \texttt{A}^{\mathsf{pub} \cdot \mathsf{pub}}$，但这并不成立。

4.3 资源感知可靠性

我们指出，相对于资源感知语义，分级类型系统是可靠的。换言之，分级类型系统既能防止标准类型错误（例如调用缺失字段或方法），也能防止资源使用错误（例如要求已耗尽的资源，即无法以所需方式使用的资源）。

为了陈述并证明一个可靠性定理，我们需要为带注释表达式、环境和配置引入一个（直接的）类型判断 $\vdash_a$。类型规则见图 5。

回顾一下，$\lceil _ \rceil$ 表示擦除注释。容易看出，当且仅当一个带注释的表达式由该类型系统产生时，它才是良类型的：

▶ **命题 12**。 $\Gamma \vdash e : T \rightsquigarrow e'$ 当且仅当 $\lceil e' \rceil e = e$ 且 $\Gamma \vdash_a e' : T$。

环境和配置也具有类似性质。主要结果如下，即资源感知进展性定理。

▶ **定理 13**（资源感知进展）。 如果 $\Gamma \vdash_a e|\rho : C^r$，那么 e 要么是一个值，要么 $e|\rho \to_r e'|\rho'$，并且 $\Gamma' \vdash_a e'|\rho' : C^r$，其中 $dom(\Gamma) \subseteq dom(\Gamma')$ 且 $\Gamma' \preceq \Gamma$、Δ。

XX：18 多重分级 Featherweight Java

当归约具有非确定性时，我们可以区分两种可靠性：$soundness\text{-}must$ 表示任何计算都不会陷入停滞，而$soundness\text{-}may$表示至少存在一种不会陷入停滞的计算。may 和$must$ 性质的术语非常通用，最初源自 [11]；soundness-may 和 soundness- must 这两个具体名称是在 [9, 8] 中，在大步语义的语境下引入的。在我们的情形中，分级归约具有非确定性，因为如前所述，规则 （var） 可以通过不同方式实例化，并且可能消耗超出必要数量的资源。然而，我们预计，对于一个良类型配置，至少存在一种不会陷入停滞的计算，因此可以得到一个可能健全性结果。可以通过类似上述定理的定理证明可能健全性；该定理可视为一个可能保持类型性结果，其中包括标准的进展性。在我们的情形中，如果配置是良类型的，也就是说，注释是由类型系统生成的，那么存在一个步进 ，该步进进而导向一个良类型配置。更具体地说，类型得到保持，最初可用的资源的等级可能降低，并且还可能添加其他可用资源。

为证明这一结果，我们需要一些标准引理。

▶ **引理 14** （环境类型判定）。 以下事实成立

1. 若 $\Gamma \vdash_a \rho$ 且 $\Gamma = \Gamma'$，$x : {}_r C$ ，则 $\rho = \rho'$，$x \mapsto hv, ri$ 且 $\Gamma' \vdash_a \rho'$ and $\vdash v : C\ {}_r$。
2. 若 $\Gamma \vdash_a \rho$ 且 $x \notin dom(\rho)$ 且 $\vdash_a v : C\ {}_r$，则 $\Gamma, x : {}_r C \vdash_a \rho, x \mapsto hv, ri$。

▶ **引理 15** （值的强化）。 若 $\Gamma \vdash_a v : T$，则 $\vdash_a v : T$。

▶ **引理 16** （典型形式）。 若 $\Gamma \vdash_a v : C^r$，则 $v = new D([v_1]_{r_1}, ..., [v_n]_{r_n})$ 且 $D \preceq C$，字段$(D) = D_1^{r_1} f_1; \ldots D_n^{r_n} f_n;$ 且 $\Gamma_1 + \ldots + \Gamma_m \preceq \Gamma$ 且 $\Gamma_i \vdash_a v_i : D_i^{s \cdot r_i}$，对所有$i \in 1..m$，满足 $r \preceq s$。

▶ **引理 17** (重命名)。 若 $\Gamma, x :_r C \vdash_a e : T$，且 $y \notin dom(\Gamma)$，则 $\Gamma, y :_r C \vdash_a e[y/x] : T$.

▶ **引理 18**。 若 $dom(\Theta) \cap dom(\Delta) = \emptyset$ 则 $(\Gamma, \Theta) + \Delta = (\Gamma + \Delta)\ \Theta$。

定理 13 是以下更一般结果的一个特例，该结果明确给出了进行归纳所需的不变量。事实上，通过考察约简规则可以发现，计算规则要么向环境中添加新变量，要么按一定数量降低某个变量的等级，而该数量取决于归约的等级。后一种情况下，这个数量可以任意选择，但须满足唯一的限制：它非零，且至少等于归约的等级。然而，要证明进展性，我们不仅要证明可以进行归约，还必须证明：若归约是在某个上下文中进行的，例如求值构造子的某个参数，那么归约之后仍有足够的资源继续归约，也就是说，能够求值该上下文的其余部分（构造子的其他参数）。这意味着，所得环境拥有足够的资源，可以为整个上下文（构造子调用）定型。因此，在下文定理的陈述中，我们在定理 13 的假设中加入一个类型上下文 Θ ，用于记录归约过程中希望保留的资源数量（见定理第 4 项）。这样一来，当降低变量的等级时，我们就可以选择合适的保留等级，并在使用上下文约简规则时重建类型推导。对于顶层表达式，正如定理 13 的证明所示， Θ 对于表达式所处类型上下文中的所有变量而言都只是 $\mathbf{0}$ 。

▶ **定理 19**。 如果 $\Delta \vdash_a e : C^r$ 且 $\Gamma \vdash_a \rho$ 且 $\Delta + \Theta \preceq \Gamma$ 且 $dom(\Delta) \subseteq dom(\Theta)$ 并且 e 不是一个值，那么存在 $e', \rho', \Delta', \Gamma'$ 和 Θ' ，使得

1. $e|\rho \to_r e'|\rho'$ 和2. $\Delta' \vdash_a e' : C^r$ 与

$\Delta' \preceq \Delta$， Θ' 和

3. $\Gamma' \vdash_a \rho'$ 与 $\Gamma' \preceq \Gamma$， Θ' 和
4. $\Delta' + \Theta \preceq \Gamma'$。

证明。 证明通过对类型规则进行归纳得出。

(t-var) 根据假设，我们知道 $\Delta = x :_r C$ and, with $r \neq 0$. 由于 $dom(\Delta) \subseteq dom(\Theta)$，我们有 $\Theta = \Theta_1, x :_{s'} C$，并且由于 $\Delta + \Theta \preceq \Gamma$，我们有 $\Gamma = \Gamma_1, x :_s C$ with $r + s' \preceq s$。此外，由于 $\Gamma \vdash_a \rho$，根据引理 14(1)，我们有 $\rho = \rho', x \mapsto hv, si$, $\Gamma_1 \vdash_a \rho'$ and $\vdash_a v : C^s$。然后，根据规则（var），我们得到 $x|\rho \to_r v|\rho', x \mapsto hv, s'i$。由于 $s' \preceq r + s' \preceq s$，根据规则（t-sub），我们得到 $\vdash_a v : D^{s'}$，并由引理 14(2)得出 $\Gamma_1, x :_{s'} C \vdash_a \rho', x \mapsto hv, s'i$。由于 $r \preceq r + s' \preceq s$，再次根据规则 (sub)，我们得到 $\vdash_a v : C\, r$。现在，令 $\Delta' = \Theta' = \emptyset$ and $\Gamma' = \Gamma_1, x :_{s'} C$。我们立即得到 $\Delta' \preceq \Delta, \Theta'$ and $\Gamma' \preceq \Gamma, \Theta'$。此外，我们得到 $\Delta' + \Theta \preceq \Gamma'$，因为 $\Delta' + \Theta = \Theta = \Theta_1, x :_{s'} C$；并且从 $\Theta \preceq \Delta + \Theta \preceq \Gamma = \Gamma_1, x :_s C$，我们得到 $\Theta_1 \preceq \Gamma_1$，因此 $\Theta_1, x :_{s'} C \preceq \Gamma_1, x :_{s'} C = \Gamma'$，正如所需。

(t-sub) 根据假设，我们知道 $\Delta_1 \vdash_a e : D^s$ with $\Delta_1 \preceq \Delta$ and $D^s \leq C_r$，这意味着 $D \leq C$ and $r \preceq s$。我们分两种情况讨论。

- 如果 e 是一个值，那么命题成立。否则，注意到由 $+$ 的单调性可知
- $\Delta_1 + \Theta \preceq \Delta\Theta \preceq \Gamma$；然后，根据归纳假设，我们有**1.** $e|\rho \to_s e'|\rho'$ 以及**2.** $\Delta' \vdash_a e' : D^s$ 其中 $\Delta' \preceq \Delta_1$， Θ' 以及**3.** $\Gamma' \vdash_a \rho'$ 其中 $\Gamma' \preceq \Gamma$， Θ' 以及**4.** $\Delta' + \Theta \preceq \Gamma'$。

  根据条目 1 和命题 8，由于 $r \preceq s$，我们得到 $e|\rho \to_r e'|\rho'$。由于 $D^s \leq C_r$，根据条目 2，应用规则 (T-SUB)，我们得到 $\Delta' \vdash_a e' : C^r$ and, 由于 $\Delta_1 \preceq \Delta$，我们得到 $\Delta' \preceq \Delta_1, \Theta' \preceq \Delta, \Theta'$，从而证明了命题。

(t-field-access) 根据假设，我们知道 $\Delta \vdash_a [e]_r.f_i : C_r$ $_{r_i}{}^{i}$, $\Delta \vdash_a e_1 : C^r$，且 $fields(C) = C_{r_1}{}^{1} f_1; \ldots C_{r_n}{}^{n} f_n;$。我们区分两种情况。

- 如果 e_1 是一个值，那么根据引理 16，我们有 $e_1 = \texttt{new}\, D([v_1]_{s_1}, \ldots, [v_m]_{s_m})$，且 $D \leq C$, $fields(D) = D_1^{s_1} f_1'; \ldots D_m^{s_m} f_m'$; 以及 $\Delta_1 + \ldots + \Delta_m \preceq \Delta$ and $\Delta_j \vdash_a v_j : D_j^{s \cdot s_j}$，对于所有满足 $r \preceq s$ 的 $j \in 1..m$。根据类表上的相干性条件，我们知道 $n \leq m$ and，并且对于所有 $j \in 1..n$，$D_j = C_j$ and $s_j = r_j$ and $f_j' = f_j$。因此，由于 $i \in 1..n$，我们有 $i \in 1..m$，所以根据规则（field-access），得到 $[e_1]_r.f_i|\rho \to_{r \cdot r_i} v_i|\rho$。由于 $r \preceq s$，我们有 $r \cdot r_i \preceq s \cdot r_i = s \cdot s_i$ and $D_i = C_i$，因此根据规则（t-sub），可推出 $\Delta_i \vdash_a v_i : C_i^{r \cdot r_i}$。令 $\Delta' = \Delta_i$, $\Gamma' = \Gamma$ and $\Theta' = \emptyset$。那么命题显然由 $\Delta' = \Delta_i \preceq \sum_{j=1}^{m} \Delta_j \preceq \Delta$ 得出。
- 否则，根据归纳假设，我们得到**1.** $e_1|\rho \to_r e_1'|\rho'$ 和**2.** $\Delta' \vdash_a e_1' : C^r$，其中 $\Delta' \preceq \Delta$， Θ' 且**3.** $\Gamma' \vdash_a \rho'$ 其中 $\Gamma' \preceq \Gamma$, Θ' 且**4.** $\Delta' + \Theta \preceq \Gamma'$。

  令 $e' = [e_1']_r.f_i$。根据条目 1 和假设，使用规则 (FIELD-ACCESS-CTX)，我们推导出 $e|\rho \to_{r \cdot r_i} e'|\rho'$。根据条目 2 和假设，使用规则 (T-FIELD-ACCESS)，我们得到 $\Delta' \vdash_a e' : C^{r \cdot r_i}$。最后，根据条目 3 和条目 4，我们得到该命题。

(t-new) 根据假设，我们有 $\Delta = \Delta_1 + \ldots + \Delta_n$ 和 $\Delta_i \vdash_a e_i : C^r$ $_{r_i}{}^{i}$，对所有 $i \in 1..n$ 均成立。我们区分两种情况。

- 若对于所有 $i \in 1..n$，e_i 都是值，那么 e 也是值，由此证明了命题。

- 否则，存在一个 $i \in 1..n$，使得对于所有 $j \in 1..(i-1)$，e_i 不是值，而 e_j 是值。令 $\widehat{\Theta} = \Theta + \sum_{j=1}^{i-1} \Delta_j + \sum_{j=i+1}^{n} \Delta_j$。由于 $\mathrm{dom}(\Delta_i) \subseteq \mathrm{dom}(\Delta) \subseteq \widehat{\mathrm{dom}}(\Theta)$，我们有 $\mathrm{dom}(\Delta_i) \subseteq \mathrm{dom}(\Theta)$，并且根据构造，我们有 $\Delta_i + \Theta = \Delta + \Theta \preceq \Gamma$。于是，由归纳假设可得

1. $e_i|\rho \to_{r \cdot r_i} e_i'|\rho'$ 2. $\Delta_i' \vdash_a e_i' : C^r \cdot r_i$ with $\Delta_i' \preceq \Delta_i, \Theta'$ 和 3. $\Gamma' \vdash_a \rho'$ with $\Gamma' \preceq \Gamma, \Theta'$ 和 4. $\Delta_i' + \Theta \preceq \Gamma'$。

令 $\mathsf{e}' = \mathtt{new}\,\mathsf{C}([\mathsf{v}_1]_{r_1}, \ldots, [\mathsf{v}_{i-1}]_{r_{i-1}}, [\mathsf{e}_i']_{r_i}, \ldots, [\mathsf{e}_n]_{r_n})$ 和 $\Delta' = \sum_{j=1}^{i-1} \Delta_j + \Delta_i' + \sum_{j=i+1}^{n} \Delta_j$。根据条目 1 和假设，使用规则（new-ctx），我们推导出 $\mathsf{e}|\rho \to_r \mathsf{e}'|\rho'$。根据条目 2 和假设，使用规则（t-new），我们得到 $\Delta' \text{`}_a \mathsf{e}' : \mathsf{C}\ r$。根据条目 3，我们有 $\mathrm{dom}(\Theta') \cap \mathrm{dom}(\Gamma) = \emptyset$，并且由于 $\Delta_j \preceq \Delta + \Theta \preceq \Gamma$ 对所有 $j \in 1..n$，我们得到 $\mathrm{dom}(\Delta_j) \cap \mathrm{dom}(\Theta') = \emptyset$。因此，根据 $+$ 的单调性和引理 18，我们得到

$$\Delta' \preceq \sum_{j=1}^{i-1} \Delta_j + (\Delta_i, \Theta') + \sum_{j=i+1}^{n} \Delta_j = \left(\sum_{j=1}^{n} \Delta_j\right), \Theta' = \Delta, \Theta'$$

最后，由于 $\Delta' + \Theta = \Delta_i' + \Theta$，根据条目 4，我们得出命题。

(t-invk) 根据假设，我们知道 $[e_0]_{r_0}.m([e_1]_{r_1}, \ldots, [e_n]_{r_n})$，$\Delta = \Delta_0 + \ldots + \Delta_n$，$\mathrm{mtype}(C_0, m) = r_0, C_1^{r_1} \ldots C_n^{r_n} \to C\,r$ 且 $\Delta_i \vdash_a e_i : C_{r_i}{}^{i}$，对所有 $i \in 0..n$ 均成立。接下来，我们分两种情况讨论。

- 如果对所有 $i \in 0..n$，e_i 都是一个值，记为 $\mathsf{e}_i\mathsf{v}_i$，根据引理 16，我们有 $\mathsf{v}_0 = \mathtt{new}\mathsf{D}(_)$，其中 $\mathsf{D} \le \mathsf{C}_0$；并且根据类表上的相干性条件，我们有 $\mathrm{mbody}(\mathsf{D}, \mathsf{m}) = \mathsf{hx}_1 \ldots \mathsf{x}_n, \mathsf{e}'\mathsf{i}$。于是，根据规则（invk），我们得到 $\mathsf{e}|\rho \to_r \mathsf{e}'[\mathsf{y}_0/\mathtt{this}][\mathsf{y}_1/\mathsf{x}_1 \ldots \mathsf{y}_n/\mathsf{x}_n]|\rho'$，其中 $\mathsf{y}0, \ldots, \mathsf{y}_n \notin \mathrm{dom}(\rho)$ 且 $\rho' = \rho, \mathsf{y}_0 \mapsto \mathsf{hv}_0, \mathsf{r}_0\mathsf{i} \ldots, \mathsf{y}_n \mapsto \mathsf{hv}_n, \mathsf{r}_n\mathsf{i}$。根据引理 15，我们有 $\text{`}_a \mathsf{v}_i : \mathsf{C}_i^{r_i}$，对所有 $i \in 0..n$ 均成立，因此，根据引理 14(2)，我们得到 $\Gamma, \mathsf{y}_0 :_{r_0} \mathsf{C}0, \ldots, \mathsf{y}_n :_{r_n} \mathsf{C}_n \text{`}_a \rho'$。根据条件（t-meth）和规则（t-sub），我们知道 $\mathtt{this} :_{r_0} \mathsf{C}_0, \mathsf{x}_1 :_{r_1} \mathsf{C}1, \ldots, \mathsf{x}_n :_{r_n} \mathsf{C}_n \text{`}_a \mathsf{e}' : \mathsf{C}\ r$。令 $\Delta' = \Theta' = \mathsf{y}_0 :_{r_0} \mathsf{C}0, \ldots, \mathsf{y}_n :_{r_n} \mathsf{C}_n$，且 $\Gamma' = \Gamma, \Theta'$，因此我们立即有 $\Gamma' \preceq \Gamma, \Theta'$ 和 $\Delta' = \Theta' \preceq \Delta, \Theta'$。根据引理 17，我们有 $\Delta' \text{`}_a \mathsf{e}'[\mathsf{y}_0/\mathtt{this}][\mathsf{y}_1/\mathsf{x}_1 \ldots \mathsf{y}_n/\mathsf{x}_n] : \mathsf{C}\ r$。最后，由于 $\Theta \preceq \Delta + \Theta \preceq \Gamma$ 和 $\mathsf{y}0, \ldots, \mathsf{y}_n \notin \mathrm{dom}(\Gamma)$ 蕴含 $\mathsf{y}0, \ldots, \mathsf{y}_n \notin \mathrm{dom}(\Theta)$，我们得到 $\Delta' + \Theta = \Theta, \Delta' \preceq (\Delta + \Theta)$ $\Delta' \preceq \Gamma, \Delta' = \Gamma'$。
- 否则，存在一个 $i \in 0..n$，使得对于所有 $j \in 0..(i-1)$，e_i 不是一个值，而 e_j 是一个值。令 $\widehat{\Delta} = \sum_{j=0}^{i-1} \Delta_j + \sum_{j=i+1}^{n} \Delta_j$ 和 $\widehat{\Theta} = \Theta + \widehat{\Delta}$。因此，$\widehat{\Delta}_i + \Theta = \Delta + \Theta \preceq \Gamma$。根据归纳假设，我们有

1. $e_i|\rho \to_{r_i} e_i'|\rho'$ 2. $\Delta_i' \vdash_a e_i' : C_i^{r_i}$ 其中 $\Delta_i' \preceq \Delta_i$，$\Theta'$ 以及 3. $\Gamma' \vdash_a \rho'$ 其中 $\Gamma' \preceq \Gamma$，Θ' 以及 4. $\Delta_i' + \Theta \preceq \Gamma'$。

令 $\mathsf{e}' = [\mathsf{e}_0]_{r_0}.\mathsf{m}(\ldots, [\mathsf{e}_{i-1}]_{r_{i-1}}, [\mathsf{e}_i']_{r_i}, [\mathsf{e}_{i+1}]_{r_{i+1}}, \ldots, [\mathsf{e}_n]_{r_n})$ 和 $\Delta' = \Delta_i' + \Delta$。根据条目 1 和假设，视 $i = 0$ 是否成立，我们使用规则（invk-rcv-ctx）或（invk-args-ctx），推导出 $\mathsf{e}|\rho \to_r \mathsf{e}'|\rho'$。根据条目 2 和假设，使用规则（t-invk），可得 $\Delta' \text{`}_a \mathsf{e}' : \mathsf{C}\ r$。根据条目 3，$\Theta' \cap \Gamma$ 可得 $\mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Theta') = \emptyset$，并且由于 $\Delta \preceq \Delta + \Theta \preceq \Gamma$，得到 $\mathrm{dom}(\Delta) \cap \mathrm{dom}(\Theta') = \emptyset$。因此，根据 $+$ 的单调性和引理 18，$\Delta' = \Delta_i' + \Delta \preceq \Delta_i, \Theta' + \Delta = (\Delta_i + \Delta), \Theta' = \Delta, \Theta'$。最后，由于 $\Delta' + \Theta = \Delta_i' + \Delta + \Theta = \Delta_i' + \Theta$，根据条目 4 可得该命题。

(t-block) 根据假设，我们有 $\Delta = \Delta_1 + \Delta_2$ 和 $\Delta_1 \vdash_a e_1 : D^s$，以及 $\Delta_2, x :_s D \vdash_a e_2 : C_r$。然后，我们分两种情况讨论。

- 如果 $e_1 = v$ 是一个值，那么根据规则（block），我们有 $\{D\ x\ = [v]_s; e_2\} | \rho \to_r e_2[y/x] | \rho$，$y \mapsto hv$，si，其中 $y \notin dom(\rho)$。根据引理 15，我们得到 $\vdash_a v : D^s$，因此，根据引理 14(2)，我们得到 $\Gamma, y :_s D \vdash_a \rho$，$y \mapsto hv$，si。注意，这意味着 $y \notin dom(\Gamma)$，因此 $y \notin dom(\Delta_1)$ 且 $y \notin dom(\Delta_2)$。然后，令 $\Theta' = y :_s D$，$\Delta' = \Delta_2$，Θ' 以及 $\Gamma' = \Gamma, \Theta'$，于是我们立即有 $\Gamma' \preceq \Gamma, \Theta'$ 以及 $\Delta' = \Delta_2$，$\Theta' \preceq \Delta, \Theta'$，因为 $\Delta_2 \preceq \Delta_1 + \Delta_2 = \Delta$。根据引理 17，我们有 $\Delta_2, y :_s D \vdash_a e_2[y/x] : C\ r$。最后，由于 $\Theta \preceq \Gamma$ 且 $y \notin dom(\Gamma)$ 蕴含 $y \notin dom(\Theta)$，根据引理 18，我们得到 $\Delta' + \Theta = (\Delta_2 + \Theta)$ $\Theta' \preceq (\Delta + \Theta)$ $\Theta' \preceq \Gamma$，$\Theta' = \Gamma'$。

- 否则，令 $\Theta = \Theta + \Delta_2$。由于 $dom(\Delta_1) \subseteq dom(\Delta) \subseteq dom(\Theta)$，我们有 $dom(\Delta_1) \subseteq dom(\Theta)$，并且根据构造，我们还有 $\Delta_1 + \Theta = \Delta + \Theta \preceq \Gamma$。根据归纳假设，我们有
  1. $e_1 | \rho \to_s e_1' | \rho'$ 2. $\Delta_1' \vdash_a e_1' : D^s$ 其中 $\Delta_1' \preceq \Delta_1, \Theta'$ 和 3. $\Gamma' \vdash_a \rho'$ 其中 $\Gamma' \preceq \Gamma, \Theta'$ 和 4. $\Delta_1' + \Theta \preceq \Gamma'$。

  令 $e' = \{D\ x\ = [e_1']_s; e_2\}$ 和 $\Delta' = \Delta_1' + \Delta_2$。根据条目 1 和假设，使用规则 (block-ctx)，我们推导出 $e | \rho \to_r e' | \rho'$。根据条目 2 和假设，使用规则 (t-block)，我们有 $\Delta' \vdash_a e' : C\ r$。根据条目 3，我们有 $dom(\Theta') \cap dom(\Gamma) = \emptyset$，并且由于 $\Delta_2 \preceq \Delta + \Theta \preceq \Gamma$，可得 $dom(\Delta_2) \cap dom(\Theta') = \emptyset$。因此，根据单调性和引理 18，我们得到 $\Delta' \preceq (\Delta_1, \Theta') + \Delta_2 = (\Delta_1 + \Delta_2)$ $\Theta' = \Delta, \Theta'$。最后，由于 $\Delta' + \Theta = \Delta_1' + \Theta$，根据条目 4 可得命题。

◀

我们现在已准备好证明定理 13。

定理 13 的证明。 对规则 (T-CONF) 进行反演，得到 $\Gamma \vdash_a \rho$ 以及 $\Delta \vdash_a e : C^r$，其中 $\Delta \preceq \Gamma$。将定理 19 应用于 $\Theta = \mathbf{0} \cdot \Delta$，得到 1. $e | \rho \to_r e' | \rho'$，以及 2. $\Delta' \vdash_a e' : C^r$，其中 $\delta' \preceq \Delta$，Θ' 以及 3. $\Gamma' \vdash_a \rho$，其中 $\Gamma' \preceq \Gamma$，Θ' 以及 4. $\Delta' + \Theta \preceq \Gamma'$。

由第 4 项可得 $\Delta' \preceq \Gamma'$，因此根据 (T-CONF) 以及第 2 和第 3 项，可推出 $\Gamma' \vdash_a e' | \rho' : C_r$。最后，根据命题 7，有 $dom(\rho) \subseteq dom(\rho')$；又由于根据规则 (T-ENV) 以及第 2 和第 3 项，我们知道 $dom(\rho) = dom(\Gamma)$ 且 $dom(\rho') = dom(\Gamma')$，故得证。◀

利用定理 13，我们可以证明一个资源感知可靠性定理。正如前面所指出的，这是一种可能健全性，也就是说，它指出良类型配置要么收敛到良类型值，要么发散。当存在一个以 $e | \rho \to_r^\omega$ 开头的无限步序列时，我们记作 $\to_r e | \rho$。注意，这个判断也可以通过以下规则用余归纳方式定义：

$$\frac{e' | \rho' \to_r^\omega}{e | \rho \to_r^\omega} \quad e | \rho \to_r e' | \rho'$$

▶ **推论 20**（资源感知可靠性）。若 $\Gamma \vdash_a e | \rho : C^r$ 则以下二者必居其一：$e | \rho \to_r^\star v | \rho'$ 满足 $\Gamma' \vdash_a v | \rho' : C_r$，或者 $e | \rho \to_r^\omega$。

证明。我们称一个良类型配置 $\Gamma \vdash_a \mathsf{e}|\rho : \mathsf{C}^r$ 为良收敛的，当且仅当存在 $\mathsf{v}, \rho', \Gamma'$ 和 Δ，使得 $\mathsf{e}|\rho \to_r^\star \mathsf{v}|\rho'$，且 $\Gamma' \vdash_a \mathsf{v}|\rho' : \mathsf{C}^r$ 满足 $\Gamma' \preceq \Gamma, \Delta$，并且 $\mathrm{dom}(\Gamma) \subseteq \mathrm{dom}(\Gamma')$。该陈述等价于以下命题：若 $\Gamma \vdash_a \mathsf{e}|\rho : \mathsf{C}^r$ 不是良收敛的，则 $\mathsf{e}|\rho \to_r^\omega$。我们通过余归纳证明这一点。考虑一个不是良收敛的良类型配置 $\Gamma \vdash_a \mathsf{e}|\rho : \mathsf{C}^r$。于是，根据定理 13，我们得到 $\mathsf{e}|\rho \to_r^\star \mathsf{e}'|\rho'$，其中 $\Gamma' \vdash_a \mathsf{e}'|\rho' : \mathsf{C}^r$ 满足 $\mathrm{dom}(\Gamma) \subseteq \Gamma'$，且 $\Gamma' \preceq \Gamma, \Delta$。要通过余归纳完成证明，只需验证 $\Gamma' \vdash_a \mathsf{e}'|\rho' : \mathsf{C}^r$ 不是良收敛的。假设它是良收敛的，那么 $\mathsf{e}'|\rho' \to_r^\star \mathsf{v}|\rho''$，其中 $\Gamma'' \vdash_a \mathsf{v}|\rho'' : \mathsf{C}^r$ 满足 $\mathrm{dom}(\Gamma') \subseteq \mathrm{dom}(\Gamma'')$，且 $\Gamma'' \preceq \Gamma', \Delta'$。因此，我们有 $\mathsf{e}|\rho \to_r \mathsf{e}'|\rho' \to_r^\star \mathsf{v}|\rho''$，并且 $\mathrm{dom}(\Gamma) \subseteq \mathrm{dom}(\Gamma') \subseteq \mathrm{dom}(\Gamma'')$，且 $\Gamma'' \preceq \Gamma', \Delta' \preceq \Gamma, \Delta, \Delta'$，由此证明 $\Gamma \vdash_a \mathsf{e}|\rho : \mathsf{C}^r$ 是良收敛的，这与所需结论矛盾。◀

最后，以下推论阐述了标准语义下的主体约化，即类型和共效应得以保留，以及插桩语义的完备性，即对于良类型配置，通常语义中的每一个规约步都可以由插桩语义中的适当步来模拟。

▶ **推论 21**（主体约化）。如果 $\Gamma_1 \vdash_a e_1|\rho_1 : C^r \rightsquigarrow e_1'|\rho_1'$ 且 $e_1|\rho_1 \to e_2|\rho_2$，那么 $\Gamma_2 \vdash_a e_2|\rho_2 : C^r \rightsquigarrow e_2'|\rho_2'$，其中 $dom(\Gamma_1) \subseteq dom(\Gamma_2)$ 且 $\Gamma_2 \preceq \Gamma_1$、$\Delta$，并且 $e_1'|\rho_1' \to_r e_2'|\rho_2'$。

证明。由命题 12 可得 $\Gamma_1 \vdash_a \mathsf{e}_1'|\rho_1' : \mathsf{C}^r$；由定理 13 可得 $\mathsf{e}_1'|\rho_1' \to_r \mathsf{e}_2'|\rho_2'$ 且 $\Gamma_2 \vdash_a \mathsf{e}_2'|\rho_2' : \mathsf{C}^r$，其中 $\mathrm{dom}(\Gamma_1) \subseteq \mathrm{dom}(\Gamma_2)$ 且 $\Gamma_2 \preceq \Gamma_1$、$\Delta$。由命题 12 可得 $\Gamma_2 \vdash \mathsf{de}_2'\mathsf{e}|\mathsf{d}\rho_2'\mathsf{e} : \mathsf{C}^r \rightsquigarrow \mathsf{e}_2'|\rho_2'$；由命题 9 可得 $\mathsf{e}_1|\rho_1 \to \mathsf{de}_2'\mathsf{e}|\mathsf{d}\rho_2'\mathsf{e}$。根据标准语义的确定性，我们有 $\mathsf{de}_2'\mathsf{e} = \mathsf{e}_2$ 且 $\mathsf{d}\rho_2'\mathsf{e} = \rho_2$，因此命题得证。◀

5 组合等级

正如我们所见，每种等级代数都编码了一种特定的资源使用概念。然而，在程序中，可能需要针对不同种类的资源或不同的代码片段（例如不同的类）采用不同的使用概念。这意味着需要同时使用多个等级代数，也就是说，需要一个由集合 $\mathcal{K}$ 上的等级种类索引的等级代数族 $(H_k)_{k\in\mathcal{K}}$ [6]。我们假设等级种类始终包含 N 和 T，其中 H_N 和 H_T 分别是自然数等级代数和平凡等级代数，如示例 2 所示，因为它们发挥着特殊作用，后文将对此进行说明。

▶ 示例 22。假设在某个程序中使用等级种类 N、A、P、PP、AP 和 T，其中：H_A 是亲和性等级代数，如示例 2(3) 所示。H_P 和 H_PP 是隐私级别等级代数的两种不同实例化，如例 6所示；具体而言，在 H_P 中只有两个隐私级别：公开和私有；而在 H_PP 中，隐私级别为 a、b、c、d，其中 $\mathsf{a} \preceq \mathsf{b} \preceq \mathsf{d}$，且 $\mathsf{a} \preceq \mathsf{c} \preceq \mathsf{d}$。最后，$H_\mathsf{AP}$ 是 $H_\mathsf{A} \times H_\mathsf{P}$，如示例 2(8) 所示，可同时跟踪亲和性和隐私。

我们希望让程序员能够同时使用所有这些种类的等级。为此，我们应当说明如何通过各自独特的运算符组合不同种类的等级；例如，一个种类为 k 的对象可以包含一个种类为 μ 的字段，因此字段访问应由它们的乘法来确定等级。

换言之，我们需要从族 $(H_k)_{k\in K}$ 出发，构造一个具有异构等级的单一分级代数。这样，前几节中针对

[6] H 表示“异质”。

任意等级代数所发展的元理论也适用于同时使用多个等级代数的情形。请注意，这种构造是必要的，因为我们不希望可用等级像 [22] 中那样被固定；相反，程序员应当能够为特定应用定义等级，并借助某种语言支持来实现，而这种支持可以就是语言本身，具体内容将在第6节中介绍。

5.1 直接细化

显而易见的方法是将异构等级定义为对 hk, ri，其中 k $\in$ K，r $\in H_k$。关于运算符的定义，在以往处理副效应而非等级的工作中，[3] 我们采用了最简单的选择：将不同种类的等级进行组合（无论是求和还是求积）时，总是返回 h∞, Ti，这在某种意义上表示我们“并不知道”该如何进行组合。唯一的例外是种类为 N 的等级；实际上，由于相应的等级代数是初始的，我们知道，对于任意种类 k，都存在唯一的等级同态 ι_k from Nat to H_k，因此，要组合 hn, Ni 和 hr, ki，我们可以通过该同态将 n 映射为种类 k 的等级，然后使用种类 k 的运算符。本文将这一想法加以推广，允许程序员为每一对种类 k 和 μ指定一个唯一确定的种类 k $\oplus\mu$，以及两个唯一确定的等级同态 $\mathrm{lh}^H_{\kappa,\mu}\colon H_\kappa \to H_{\kappa\oplus\mu}$ 和 $\mathrm{rh}^H_{\kappa,\mu}\colon H_\mu \to H_{\kappa\oplus\mu}$。这样，要组合 h$\kappa$, ri 和 h$\mu$, si，我们可以将二者都映射为种类 k $\oplus\mu$的等级，然后使用种类 k $\oplus\mu$ 的运算符。

运算符 $\oplus$ 以及唯一同态族（每一对种类对应一个同态），程序员可以通过定义一个（直接）细化关系 $\sqsubset^1$（定义如下）以及一个等级同态族$H_{\kappa,\ \mu}$，以最简洁且易于检查的方式指定：$H_\kappa \to H_\mu$，其索引取遍各对 $\kappa \sqsubset^1 \mu$。

给定种类上的关系 $\Rightarrow$，从k_0到k_n的一条路径 是一个序列$k_0 \ldots k_n$，满足$k_i \Rightarrow k_{i+1}$，对所有 $i \in 1..n-1$ 均成立。如果存在一条从 κ 到 μ 的路径，我们就称 μ 是 κ 的祖先 。

那么，（直接）细化关系 是定义在$\mathcal{K} \setminus \{\mathrm{N}, \mathrm{T}\}$ 上的关系 $\sqsubset^1$，满足以下条件：

1. 对于每个 κ，μ，从 κ 到 μ 至多存在一条路径
2. 对于每个 κ，μ 若它们有公共祖先，则存在一个 最小的 公共祖先，记作 $\kappa \oplus \mu$ ；也就是说，对于任意公共祖先 ν，ν 都是$k \oplus \mu$的祖先。

N请注意，由于条件 (1)，条件 (2) 意味着，例如，从 κ 到 ν 的唯一路径，由从 κ 到$k \oplus \mu$的唯一路径，以及从$k \oplus \mu$ 到 ν 的唯一路径组成。

给定一个直接细化关系 $\sqsubset^1$，我们可以在 $\mathcal{K}$ 上导出以下结构：$\sqsubset^1$ 可以扩展为 K 上的偏序 ν，方法是取 $\sqsubset^1$ 的自反传递闭包，并添加 N $\sqsubseteq$ $\kappa \sqsubseteq$ T 对所有 $\kappa \in \mathcal{K}$。

$\oplus$可以通过定义 $\kappa \oplus \mu =$T 来扩展到所有数对，条件是 κ 和 μ 没有公共祖先。总之，我们得到一个称为等级签名的结构实例，具体内容将在定义 24 中详述。此外，给定一个 $\sqsubset^1$-同态族：

- 它们可以通过复合[7], 扩展到所有等级对 $\langle\kappa,\mu\rangle \in \mathcal{K} \setminus \{\mathrm{N},\mathrm{T}\}$，只要存在一条从 κ 到 μ的路径；由于这条路径是唯一的，因此每种种类 κ对应的同态都有唯一确定的定义。我们再加入从自然数出发以及到 Triv 的唯一同态。

[7] 请注意，通过这种方式，我们尤其可以得到所有恒等映射。

总而言之，除了每种种类对应一个等级代数外，对于满足 $\kappa \sqsubseteq \mu$的每一对$\langle \kappa, \mu \rangle$，我们还得到一个等级同态。也就是说，我们获得了一种称为异构等级代数的结构实例，详情将在定义 25 中说明。

因此，正如所期望的那样，可以通过将 r 和 s 都映射为种类 $\kappa \oplus \mu$ 的等级，然后应用种类为 $\kappa \oplus \mu$ 的运算符，来定义种类 $\langle \kappa, r \rangle$ 和 $\langle \mu, s \rangle$ 的等级组合。

下一小节将针对任意等级签名和异构等级代数这一更一般的情形，证明通过这种方式确实能得到一个等级代数，也就是说，所有必需的公理都得到满足。

请注意等级种类 N 和 T 及其对应等级代数所起的特殊作用。前者是等级签名所需的最小种类（定义 24）；这一点很重要，因为所得等级代数中的零和一（因此也是类型系统中使用的零和一）将取自该种类。后者如上所示，用作没有公共祖先的种类对的默认公共祖先。

▶ **例 23.** 回到我们的例子，程序员可以定义直接细化关系及相应的同态，如下所示：

- PP $\sqsubset^1$ P，而相应的同态映射会将 a、b 和 c 等映射为私有，将 d 映射为公开
- AP $\sqsubset^1$ A，且 AP $\sqsubset^1$ P，相应的同态是投影。因此，例如，等级 $\langle$AP, $\langle \omega$,私有$\rangle\rangle$ 表示我们可以在私有模式下任意次使用该资源；而 $\langle$PP,d$\rangle$ 表示我们可以在 d 模式下使用该资源，它映射为私有。事实上，这两个等级都映射到隐私级别的等级代数 $0 \prec$私有 $\prec$ 公开；对于前者，亲和性信息会丢失；对于后者，隐私级别 d 会映射为公开；最后，我们得到私有 $=$ 私有 $\cdot$公开。

直接细化关系如图 6 所示。虚线态射表示为自然数和 T 添加的（部分）序关系。

请注意，通过直接细化关系及相应的同态间接指定等级签名和异构等级代数，具有一项根本优势：对于每个 κ、μ，我们可以将等级κ 映射到种类 μ 的等级，且映射方式唯一（也就是说，至多存在一个同态）这一语义检查，需要检查函数定义是否等价；而现在，这项检查被直接细化定义中的检查（1）和（2）取代。这些检查纯粹是句法上的，并且可以轻松地在类型系统中实现（一个简单而更强的条件是规定每个种类在直接细化关系中都有唯一的父级，正如单继承的情形）。

在第6节中，我们将看到如何在 Java 中表示等级代数和同态；粗略地说，两者都将由实现适当泛型接口的类来表示。

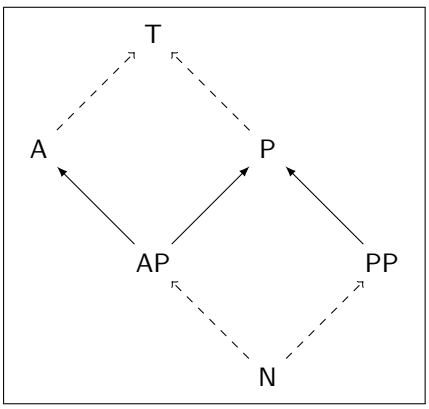


图 6 直接细化图

5.2 一般构造

我们给出一种构造：从具有适当结构的等级代数族出发，得到一个概括整个族的唯一等级代数。因此，前文针对单一分级代数所建立的元理论也适

用于同时使用多个等级代数的情形。

为了发展这一构造，我们使用简单且标准的范畴论工具，更多详情请参见 [19, 25] 。给定一个范畴$\mathcal{C}$ ，我们用$\mathcal{C}_0$ 表示$\mathcal{C}$ 中对象的集合，并称$\mathcal{C}$ 为小范畴，当$\mathcal{C}_0$ 是一个集合时。回顾一下，任意偏序集$\mathcal{P}$ $= \langle \mathcal{P}_0, \sqsubseteq \rangle$ 都可以视为一个小范畴，其对象是$\mathcal{P}_0$ 的元素，并且对于所有 $x, y \in \mathcal{P}_0,$ ，存在一个态射 $x \to y$ 当且仅当 $x \sqsubseteq y$；因此，$\mathcal{P}_0,$ 中的任意一对对象之间至多有一个态射，且唯一的同构是恒等映射。

▶ **定义 24.** 一个等级签名$\mathcal{S}$ 是一个具有有限上确界的偏序集，也就是说，它由以下数据构成：

- 一个偏序集 $h\mathcal{S}_0, \mathcal{V}i$；一个函数 $\oplus: \mathcal{S}_0 \times \mathcal{S}_0 \to \mathcal{S}_0$，它对两个参数均单调，并且对于所有 κ、
- μ、 $\nu \in \mathcal{S}_0, \kappa \oplus \mu \mathcal{V}$ ν 当且仅当 $\kappa \mathcal{V}$ ν 且 $\mu \mathcal{V}$ ν；

- 一个指定对象 $I \in \mathcal{S}_0$ ，使得 $I \mathcal{V}$ κ，对所有 $\kappa \in \mathcal{S}_0$ 均成立。

直观地说，$\mathcal{S}$ 中的对象表示我们希望处理的等级种类，而态射（即序关系）则表示这些种类之间的细化： $\kappa \sqsubseteq \mu$ 表示种类 κ 比种类 μ更具体。运算 $\oplus$ 将两个种类合并，得到一个能够概括二者的最具体种类。最后，种类 I 是其中最具体的。将等级签名$\mathcal{S}$ 视为范畴，等级签名意味着它具有有限余积。

容易验证，以下性质对所有 $\kappa, \mu, \nu \in \mathcal{S}_0$ 均成立：

$$(\kappa \oplus \mu) \oplus \nu = \kappa \oplus (\mu \oplus \nu) \qquad \kappa \oplus \kappa = \kappa$$
$$\kappa \oplus \mu = \mu \oplus \kappa \qquad \kappa \oplus I = \kappa$$

也就是说，$\langle \mathcal{S}_0, \oplus, I \rangle$ 是一个交换幂等幺半群。

▶ 定义 *25.* 等级签名 S 上的异构等级代数就是一个函子$H : S \to GrAlg$. 。这意味着，对于每个种类 $\kappa \in S_0,$ ，它都由一个等级代数 $H(\kappa)$（也写作 H_κ）以及对于每个态射$\kappa \mathcal{V}$ μ，一个等级代数同态 $H_{\kappa,\mu}$: $H_\kappa \to H_\mu$ 构成，并且与复合和恒等映射相容[8] , ；也就是说，若 $\kappa \mathcal{V}$ $\mu \mathcal{V}$ ν ，则 $H_{\kappa,\nu} = H_{\mu,\nu} \circ H_{\kappa,\mu}$，且 $H_{\kappa,\kappa} = id_{H_\kappa}$。

本质上，同态 $H_{\kappa,\mu}$ 实现了细化 $\kappa \sqsubseteq$ μ，将种类为 κ 的等级转换为种类为 μ的等级，同时保持等级代数结构。

注意，$\mathcal{S}$中的态射 $I \sqsubseteq \kappa$、 $\kappa \sqsubseteq \kappa \oplus \mu$和 $\mu \sqsubseteq \kappa \oplus \mu$给出了以下等级代数同态：

$$\mathsf{in}^H_\kappa = H_{I,\kappa} : H_I \to H_\kappa \quad \mathsf{lh}^H_{\kappa,\mu} = H_{\kappa,\kappa\oplus\mu} : H_\kappa \to H_{\kappa\oplus\mu} \quad \mathsf{rh}^H_{\kappa,\mu} = H_{\mu,\kappa\oplus\mu} : H_\mu \to H_{\kappa\oplus\mu}$$

这为我们提供了一种将种类为 I 的等级映射到任意其他种类的等级，以及将种类为 κ 和 μ 的等级映射到其复合 $\kappa \oplus \mu$ 的等级的方法。根据 H 的函子性，并利用 $\mathcal{S}$ 的交换幂等幺半群结构，我们得到以下等式

[8] 记号 $H_{\kappa,\mu}$ 是合理的，因为 κ 与 μ 之间至多存在一个态射。

这些等式在范畴 $\mathcal{GrAlg}$ 中成立，从而确保此类变换的一致性：

$$\mathsf{lh}^H_{\kappa\oplus\mu,\nu} \circ \mathsf{lh}^H_{\kappa,\mu} = \mathsf{lh}^H_{\kappa,\mu\oplus\nu} \tag{1}$$
$$\mathsf{rh}^H_{\kappa\oplus\mu,\nu} \circ \mathsf{lh}^H_{\kappa,\mu} = \mathsf{rh}^H_{\kappa,\mu\oplus\nu} \circ \mathsf{lh}^H_{\mu,\nu} \tag{2}$$
$$\mathsf{lh}^H_{\kappa,\mu} = \mathsf{rh}^H_{\mu,\kappa} \tag{3}$$
$$\mathsf{lh}^H_{\kappa,\kappa} = \mathsf{id}_{H_\kappa} \tag{4}$$
$$\mathsf{lh}^H_{\kappa,I} = \mathsf{id}_{H_\kappa} \tag{5}$$
$$\mathsf{rh}^H_{\kappa,I} = \mathsf{in}^H_\kappa \tag{6}$$

接下来，我们将展示如何将异构等级代数转化为单一分级代数。我们将介绍的过程基于 Grothendieck 提出的一般构造，该构造定义于索引范畴之上 [16] 。

设等级签名 $\mathcal{S}$和异构等级代数 $H : \mathcal{S} \to \mathcal{GrAlg}$。考虑以下集合：

$$|G(H)| = \{\langle \kappa, r\rangle \mid \kappa \in \mathcal{S}_0,\ r \in |H_\kappa|\}$$

也就是说， $G(H)$ 的元素将是带种类的分级，即由一个种类 κ 及该种类的一个等级组成的有序对。注意，这确实是一个集合，因为$\mathcal{S}$ 是小的，也就是说，$\mathcal{S}_0$ 是一个集合。然后，我们在 $|G(H)|$ 上定义二元关系 $\preceq_H$ 如下：

$$\langle \kappa, r\rangle \preceq_H \langle \mu, s\rangle \quad \text{iff} \quad \kappa \sqsubseteq \mu \text{ and } H_{\kappa,\mu}(r) \preceq_\mu s$$

也就是说，种类 κ 必须比种类 μ 更具体；并且，将 r 通过 $H_{\kappa,\ \mu}$进行变换后，我们得到一个种类为 μ 的等级，它小于 s。如下命题表明，这些数据定义了一个偏序集。

▶ **命题 26**。 $\langle |G(H)|, \preceq_H \rangle$ 是一个偏序集。

加法结构由一个二元运算 $+_H$: $|G(H)| \times |G(H)| \to |G(H)|$以及一个元素 $\mathbf{0}_H$ （属于 $|G(H)|$ ）给出，定义如下：

$$\langle \kappa, r\rangle +_H \langle \mu, s\rangle = \langle \kappa \oplus \mu, \mathsf{lh}^H_{\kappa,\mu}(r) +_{\kappa\oplus\mu} \mathsf{rh}^H_{\kappa,\mu}(s)\rangle \qquad \mathbf{0}_H = \langle I, \mathbf{0}_I\rangle$$

这意味着，两个元素 $\langle \kappa, r\rangle$ 和 $\langle \mu, s\rangle$ 的加法，首先要将r 和 s 映射到同时推广 κ 和 μ的最具体种类，即 $\kappa \oplus \mu$，然后在该种类上的等级代数中将它们相加。零元素就是最具体种类的零。

▶ **命题 27.** $\langle |G(H)|, \preceq_H, +_H, \mathbf{0}_H \rangle$ 是一个有序交换幺半群。

▶ **命题 28.** $\mathbf{0}_H \preceq_H \langle \kappa, r\rangle$ 对于每个 $\langle \kappa, r\rangle \in |G(H)|$。

类似地，乘法结构由一个二元运算 $\cdot_H$: $|G(H)| \times |G(H)| \to |G(H)|$ 以及一个元素 $\mathbf{1}_H$ （属于 $|G(H)|$ ）给出，定义如下：

$$\langle \kappa, r\rangle \cdot_H \langle \mu, s\rangle = \begin{cases} \langle \kappa \oplus \mu, \mathsf{lh}^H_{\kappa,\mu}(r) \cdot_{\kappa\oplus\mu} \mathsf{rh}^H_{\kappa,\mu}(s)\rangle & \langle \kappa, r\rangle \neq \mathbf{0}_H \text{ and } \langle \mu, s\rangle \neq \mathbf{0}_H \\ \mathbf{0}_H & \text{otherwise} \end{cases} \qquad \mathbf{1}_H = \langle I, \mathbf{1}_I\rangle$$

请注意，上述定义与加法运算几乎遵循相同的模式，但我们强制要求乘以 $\mathbf{0}_H$ 后再次得到 $\mathbf{0}_H$，这是等级代数的一项关键性质。

▶ **命题 29**。 $\langle |G(H)|, \preceq_H, \cdot_H, \mathbf{1}_H \rangle$ 是一个有序幺半群。

综上，我们最终得到以下结果。

▶ **定理 30**。 $G(H) = \langle |G(H)|, \preceq_H, +_H, \cdot_H, \mathbf{0}_H, \mathbf{1}_H \rangle$ 是一个等级代数。

6 作为 Java 表达式的等级

在第4节中，我们描述了如何为类 Java 语言配备等级，以装饰类型，这些等级取自任意等级代数。此外，在第 5 节中，我们展示了通过以特定方式组合不同的等级代数，可以得到这样的等级代数；这种特定方式由提供一组最小等级同态来确定，而不同的等级代数分别对应于跟踪资源使用情况的不同方式。这样一来，同一程序中可以共存异构等级，无需一劳永逸地固定下来，而是可以通过用户定义的等级代数和同态加以扩展。在本节中，我们探讨如何为此提供语言层面的支持。我们可以使用一种专门的配置语言来实现这一点；不过，我们认为一种有趣的解决方案是，类型中的分级注释本身可以用 Java 编写。

其核心思想是，这类分级注解是预定义接口 `Grade`（由实现该接口的类实现）的 Java 表达式，类似于 Java异常是 `Exception`（其子类）的表达式。此外，等级同态也由用户定义。也就是说，用户程序可以包含：

- 成对的 等级类 和 等级工厂类，每一对都为特定应用程序所需的等级代数建模，其中工厂类提供其常量等级同态类，每个等级同态类都对从一个等级代数（类）到另一个等级代数（类）的同态进行建模。

在对带有分级注解的代码进行类型检查时，类型系统内部使用的等级，是通过已声明的等级同态类，按照第 5 节所述的构造，将所有已声明的等级代数（类）组合后得到的等级。对同一等级代数（类）中的等级进行的运算源自用户定义的方法，具体将在下文进一步讨论；而对异构等级进行的运算，则按照第 5 节中的构造派生。

下面我们将逐一描述各个要点。

6.1 等级和等级工厂类

它们分别是实现以下泛型接口的类：[9]

```
interface Grade<T extends Grade<T>> {
  boolean leq(T x);
  T sum(T x);
  T mult(T x);
}
interface GradeFactory<T extends Grade<T>>{
  T zero();
  T one();
}
```

例如，以示例 22 为例，下面展示了亲和等级的完整实现，而其他类仅给出框架，以求简洁：

```
abstract class Affinity implements Grade<Affinity>{}

class AffinityFactory implements GradeFactory<Affinity>{
```

[9] 为使代码更简洁，我们省略了访问修饰符。

```
  Affinity zero(){return new ZeroA();}
  Affinity one(){return new One();}
}

class ZeroA extends Affinity{
  boolean leq(Affinity x){return true;}
  Affinity sum(Affinity x){return x;}
  Affinity mult(Affinity x){return this;}
}

class One extends Affinity{
  boolean leq(Affinity x){return !(x instanceof ZeroA);}
  Affinity sum(Affinity x){
    if (x instanceof ZeroA) return this;
    return new Omega();
  }
  Affinity mult(Affinity x){return x;}
}
class Omega extends Affinity{
  boolean leq(Affinity x){return (x instanceof Omega);}
  Affinity sum(Affinity x){return this;}
  Affinity mult(Affinity x){
    if (x instanceof ZeroA) return x;
    return this;
  }
}

abstract class Privacy implements Grade<Privacy> {...}

class ZeroP extends Privacy{...}
class Private extends Privacy{...}
class Public extends Privacy{...}
class PrivacyFactory implements GradeFactory<Privacy>{...}

abstract class PPrivacy implements Grade<PPrivacy> {...}

class ZeroPP extends PPrivacy{...}
class LevelA extends PPrivacy{...}
...
class LevelD extends PPrivacy{...}

class PPrivacyFactory implements GradeFactory<PPrivacy>{...}

class AffinityPrivacy implements Grade<AffinityPrivacy>{
  Affinity first; Privacy second;
  ...
}

class AffinityPrivacyFactory implements GradeFactory<AffinityPrivacy>{
...
}
```

请注意，对于 `Affinity`、 `Privacy` 和 `PPrivacy`，我们也可以使用带常量的 `enum` 类；这里我们选择使用类层次结构，这是 FJ 演算提供的一项核心面向对象特性。更一般地说，正如预期的那样，实现

这在很大程度上取决于目标语言的特性；例如，在 Haskell 中，我们可以将 `Grade` as 表达为类型类，将等级代数作为其实例，对于下面的 `GradeHom` 也可以采用同样的方式。

6.2 等级同态类

它们是实现以下泛型接口的类：

```
   interface GradeHom<T extends Grade<T>、 R extends Grade<R>> {
```

```
Privacy apply (PPrivacy p){
```

6.3 预定义等级类

等级代数自然数可由以下类实现，这些类假定已预先定义。

```
abstract class Nat implements Grade<Nat> {
  abstract <R extends Grade<R>>  R transform(GradeFactory<R> factory);
  }

class Zero extends Nat {
  <R extends Grade<R>> R transform(GradeFactory<R> factory){
    return factory.zero();
  }
  boolean leq(Nat x){return true;}
  Nat sum(Nat x){return x;}
  Nat mult(Nat x){return this;}
}

class Succ extends Nat {
  Nat pred;
  Succ(Nat pred){this.pred=pred;}

  <R extends Grade<R>> R transform(GradeFactory<R> factory) {
    return factory.one().sum(pred.transform(factory));
  }
  boolean leq(Nat x){
  if (x instanceof Zero){ return false;}
```

```
    return pred.leq(((Succ) x).pred);}
```

自然数上的序、加法和乘法均按预期的归纳方式定义。类 `NtoGradeHom<R>` 的对象表示从自然数 Nat 到等级代数（由） `R` 建模的唯一同态 ι_R，它将自然数 n 映射为 R 中 n 个 $\mathbf{1}_R$ 的和，定义见第 4 页。为此，创建该对象时需指定相应工厂类的构造器参数。[10] 将同态应用于自然数 `n` 时（调用 `apply`，参数为 `n`），辅助方法 `transform` 会使用工厂提供的 `zero` 和 `one` 方法，构造与 `n` 对应的类 `R` 的对象。

类似地，对于等级代数 Triv，我们有：

```
class Triv implements Grade<Triv>{boolean leq(Triv t){return true;}
```

6.4 作为 Java 表达式的分级注解

假设已经有了上文所述的类，程序员就可以编写带有作为 Java 表达式的分级注解的代码。例如，示例 10 可以写成如下形式，其中注解用方括号括起来：

```
class Pair { A[new One()] first; A[new One()] second;
  A[new ZeroA()] getFirstZero() [new One()]{this.first}
  A[new One()] getFirstAffine() [new One()]{this.first}
  A[new Omega()] getFirst() [new One()]{this.first}
}
```

[10]为此，我们需要一种构造工厂类名称的约定。

类型检查随后可以分两步进行：
1. 用于定义等级的代码假定为标准（即非分级）Java 代码，并由标准编译器进行类型检查。

2. 带等级代码（包含以 Java 编写的分级注解）按照图4中的分级类型系统进行类型检查。其中，底层等级代数是通过第 5 节所述的构造，将用户定义的等级代数经由用户定义的等级同态组合而成。每个用户定义的代数都以对应类的实例所对应的 Java 值作为载体（等级集合），其运算则通过执行该类中用户定义的方法来计算。例如，要计算作为某个等级类的值的两个等级之和 $v_1 + v_2$，我们对v_1`.sum(`v_2`)`求值。计算等级同态的结果时也同理。

为了使整个过程正确运行，以下是程序员的职责：等级类、等级工厂类和等级同态类应满足其所建模结构所要求的公理，例如，由 `sum`方法导出的加法应满足交换律和结合律。在 Haskell 中定义 `Functor` 或 `Monad`的实例时，情况也是如此。

- 定义等级的代码应当能够终止，因为如上所述，第二步类型检查要求执行在第一步中通过类型检查的代码。
- 最后，通过声明等级同态类而隐式定义的等级类之间的关系，实际上应当是直接细化关系，也就是说，应满足以下两个要求：(1) 两个等级类之间至多存在一条路径；(2) 具有公共祖先的任意两个等级类都具有一个最小公共祖先。这些要求很容易检查，类似于检查继承是否无环，或多重继承中是否不存在菱形结构。

一个有趣之处在于，实现可以以参数化的方式使用辅助工具，尤其是终止性检查器，以防止实现等级操作的方法发生发散，和/或使用验证器来确保它们具备所需的性质。

7 相关工作

对本文工作最具启发性的两项贡献是 [7] 中提出的插桩语义，以及 Granule 语言 [22]。在 [7], 中，作者开发了 GraD，这是一种分级依赖类型系统，包含函数、张量积、加法和以及单位类型。此外，他们定义了一种跟踪资源使用情况的插桩操作语义，并证明该分级类型系统相对于这种插桩语义是健全的。本文采用相同的方法来定义一种以任意等级代数为参数的资源感知语义。然而，不同于 [7], 中该语义定义在带类型项上且唯一目的在于展示类型系统作用的做法，我们的语义定义独立于类型系统，正如演算系统中的标准做法。也就是说，其目标还在于提供一个简单的纯语义模型，将资源使用纳入考量。

Granule [22] 是一种支持分级模态类型的函数式语言，可以同时使用不同种类的等级，包括用于精确使用的自然数、安全级别、区间、无穷以及副效应产品。我们借鉴了 Granule 允许不同种类的等级共存这一理念，也借鉴了在程序设计语言中利用分级模态类型这一总体目标。对于异构等级，本文进一步推进了 Granule 的方法：我们不希望等级代数是固定的，而是希望程序员能够通过用户定义的等级对其进行扩展。为此，我们在

第 5 节。关于分级编程语言的设计，本文探讨的是面向对象范式，而非函数式范式，并采用了一些在这一语境下似乎更合适的方案，例如使用一次分级类型且不进行装箱/拆箱。设计并实现一种真正的类 Java 语言并非本文的目标；不过，我们在第6节概述了一种可能颇有意思的方案，即直接在语言中书写分级注解。

进一步从更一般的资源感知类型系统来看，副效应最初由 [23] 引入，并由 [24] 进一步分析。具体而言， [24] 提出了一种通用的副效应系统，它通过由共效应形状索引的上下文注解，扩展了简单类型的 λ 演算。所提出的框架非常抽象，作者仅关注两种相反的实例：结构化（按变量）副效应和平坦（整个上下文）副效应，它们由特定的上下文形状选择来区分。

后续关于副效应的大部分文献都聚焦于结构化副效应，其代数描述可通过半环清晰地表达。 [6]，最先注意到这一点，并为一种函数式语言构建了结构化副效应框架。这种方法受到线性逻辑中指数模态的一般化启发，例如参见 [5]。也就是说，线性系统中线性变量与非受限变量之间的区别被推广为用副效应或等级修饰变量，以决定它们可以被使用的次数。在这种设定下，人们在将副效应与其他程序设计特性相结合方面取得了许多进展，例如计算效应 [13, 22, 10], 、依赖类型 [2, 7, 21], 和多态 [1]。 [2, 14, 1]，还探讨了其他分级类型系统，并结合了效应与副效应 [13, 22]。在所有这些论文中，通过等级跟踪使用情况，是一种强大的方法，可为类型系统添加无关性和线性性的分析，从而带来实际收益，例如擦除无关项（从而提升速度）以及编译器优化（如原地更新）。

如前所述， [21]和 [26] 观察到，结构化共效应系统中的上下文构成了等级半环上的一个模，尽管他们没有充分利用这一结构的一般性，而是将其限制为自由模，也就是结构化共效应系统。最近， [4] 展示了一个重要的非结构化实例，即一种用于追踪命令式范式中共享的共效应系统。

8 结论

本文的贡献可总结如下：

- 对 FJ 归约进行资源感知扩展，并以任意等级代数为参数。对类型系统进行资源感知扩
- 展，并证明其能够确保资源感知语义的可能健全性。形式化构造：给定不同种类的等级以及对应于种类间细化关系的等级变换（形式上是等级签名上的一个函子），即可构造
- 异构等级的等级代数。提出直接细化的概念，从而能够以简洁且易于检查的方式指定上述函子。概述一种 Java 扩展，其中等级由用户定义，分级注解则直接写在语言中。
-
-

如前所述，上述贡献中的关键新颖思想大多独立于语言。因此，未来工作的一个自然起点是探索它们在另一种范式中的实现，例如函数式范式。这包括定义一种独立于类型的参数化资源感知归约，设计一种具有一次性

分级类型的类型系统，并可能在函数式语言中设计用户定义的等级，例如在 Haskell 中借助类型类特性来实现。虽然总体方法仍应适用，但我们预计，鉴于该范式的特定特性，相关研究会颇具挑战。

本文定义的资源感知操作语义要求在子项中加入注释，唯一目的是确定外围项归约时它们的归约等级。如第 3 节所述，采用大步风格显然可以免去这种技术性手段；只需保留构造子子项中的注释，因为它们表达了对语义的真正约束。因此，值得研究的一个非常有趣的替代方案是资源感知语义的大步版本，以便进行更抽象、更简洁的呈现。采用这一方案时，我们应使用 [9, 8] 中近期提出的技术来证明可能健全性。

回到类 Java 语言，本文考察的 FJ 语言不包含命令式特性。引入可变内存将带来许多重要的研究方向。首先，除了本文以及一般文献中所呈现的模型——在这些模型中，“使用资源”意味着“用变量的值替换变量”——还可以采用另一种观点：资源就是内存，而“使用”意味着“与内存交互”。此外，我们还希望更深入地探究如何通过等级代数表达命令式范式中典型的使用形式，例如只读修饰符，以及更一般的能力 [17, 15]。

参考文献

1 Andreas Abel 和 Jean-Philippe Bernardy。类型系统中模态的统一视角。ACM 程序设计语言汇刊，4(ICFP):90:1–90:28，2020 年。 doi:10.1145/3408972。2 Robert Atkey。定量类型论的语法与语义。载于 Anuj Dawar 和 Erich Grädel 编，IEEE 计算机科学逻辑研讨会，LICS 2018，第 56–65 页。ACM Press，2018 年。 doi:10.1145/3209108.3209189。3 Riccardo Bianchini、Francesco Dagnino、Paola Giannini 和 Elena Zucca。带有用户定义的共效应的类 Java 演算。载于 Ugo Dal Lago 和 Daniele Gorla 编，ICTCS’22 - 意大利理论计算机科学会议，CEUR 研讨会论文集第 3284 卷，第 66–78 页。CEUR-WS.org，2022 年。4 Riccardo Bianchini、Francesco Dagnino、Paola Giannini、Elena Zucca 和 Marco Servetto。用于共享与变异的共效应。ACM 程序设计语言汇刊，6(OOPSLA):870–898，2022 年。 doi:10.1145/3563319。5 Flavien Breuvart 和 Michele Pagani。在线性逻辑的关系语义中对共效应进行建模。载于 Stephan Kreutzer 编，第 24 届 EACSL 计算机科学逻辑年会，CSL 2015，LIPIcs 第 41 卷，第 567–581 页。Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik，2015 年。 doi:10.4230/LIPIcs.CSL.2015.567。6 Aloïs Brunel、Marco Gaboardi、Damiano Mazza 和 Steve Zdancewic。核心定量共效应演算。载于 Zhong Shao 编，欧洲编程研讨会，ESOP 2013，《计算机科学讲义》第 8410 卷，第 351–370 页。Springer，2014 年。 doi:10.1007/978-3-642-54833-8_19。7 Pritam Choudhury、Harley Eades III、Richard A. Eisenberg 和 Stephanie Weirich。具有用量感知语义的分级依赖类型系统。ACM 程序设计语言汇刊，5(POPL):1–32，2021 年。 doi:10.1145/3434331。8 Francesco Dagnino。大步语义的元理论。ACM 计算逻辑汇刊，23(3):20:1–20:50，2022 年。 doi:10.1145/3522729。9 Francesco Dagnino、Viviana Bono、Elena Zucca 和 Mariangiola Dezani-Ciancaglini。大步语义的健全性条件。载于 Peter Müller 编，欧洲编程研讨会

XX:34 多重分级 FeatherweightJava

，ESOP 2020，《计算机科学讲义》第 12075 卷，第 169–196 页。Springer，2020 年。 doi:10.1007/978-3-030-44914-8_7。10 Ugo Dal Lago 和 Francesco Gavazzo。效应与反效应的关系理论。ACM 程序设计语言汇刊，6(POPL):1–28，2022 年。 doi:10.1145/3498692。11 Rocco De Nicola 和 Matthew Hennessy。进程的测试等价。理论计算机科学，34(1):83–133，1984 年。 doi:https://doi.org/10.1016/0304-3975(84)90113-0。12 Werner Dietl、Sophia Drossopoulou 和 Peter Müller。泛型宇宙类型。载于 Erik Ernst 编，欧洲面向对象编程会议，ECOOP 2007，《计算机科学讲义》第 4609 卷，第 28–53 页。Springer，2007 年。13 Marco Gaboardi、Shin-ya Katsumata、Dominic A. Orchard、Flavien Breuvart 和 Tarmo Uustalu。通过分级组合效应与反效应。载于 Jacques Garrigue、Gabriele Keller 和 Eijiro Sumii 编，ACM 函数式编程国际会议，ICFP 2016，第 476–489 页。ACM Press，2016 年。 doi:10.1145/2951913.2951939。14 Dan R. Ghica 和 Alex I. Smith。资源半环中的有界线性类型。载于 Zhong Shao 编，欧洲编程研讨会，ESOP 2013，《计算机科学讲义》第 8410 卷，第 331–350 页。Springer，2014 年。 doi:10.1007/978-3-642-54833-8_18。15 Colin S. Gordon。使用静态能力与效应进行设计：使用、提及与不变量（珍珠）。载于 Robert Hirschfeld 和 Tobias Pape 编，欧洲面向对象编程会议，ECOOP 2020，LIPIcs 第 166 卷，第 10:1–10:25 页。Schloss Dagstuhl - Leibniz-Zentrum für Informatik，2020 年。 doi:10.4230/LIPIcs.ECOOP.2020.10。16 Alexander Grothendieck。纤维范畴与下降。载于《Revêtements étales et groupe fondamental》，第 145–194 页。Springer，1971 年。17 Philipp Haller 和 Martin Odersky。唯一性与借用能力。载于 Theo D'Hondt 编，欧洲面向对象编程会议，ECOOP 2010，《计算机科学讲义》第 6183 卷，第 354–378 页。Springer，2010 年。18 Atsushi Igarashi、Benjamin C. Pierce 和 Philip Wadler。轻量级 Java：Java 和 GJ 的最小核心演算。载于 ACM 面向对象程序设计：系统、语言与应用研讨会 1999，第 132–146 页。ACM Press，1999 年。 doi:10.1145/320384.320395。19 Saunders Mac Lane。面向实践数学家的范畴论，第 5 卷。Springer Science & Business Media，2013 年。20 Daniel Marshall、Michael Vollmer 和 Dominic Orchard。线性与唯一性：友好协约。载于 Ilya Sergey 编，欧洲编程研讨会，ESOP 2022，《计算机科学讲义》第 13240 卷，第 346–375 页。Springer，2022 年。 doi:10.1007/978-3-030-99336-8_13。21 Conor McBride。我有的是一无所有。载于 Sam Lindley、Conor McBride、Philip W. Trinder 和 Donald Sannella 编，《能够改变世界的一系列成功案例——献给 Philip Wadler 60 岁生日的论文集》，《计算机科学讲义》第 9600 卷，第 207–233 页。Springer，2016 年。 doi:10.1007/978-3-319-30936-1_12。22 Dominic Orchard、Vilem-Benjamin Liepelt 和 Harley Eades III。利用分级模态类型进行定量程序推理。ACM 程序设计语言汇刊，3(ICFP):110:1–110:30，2019 年。 doi:10.1145/3341714。23 Tomas Petricek、Dominic A. Orchard 和 Alan Mycroft。共效应：上下文依赖的统一静态分析。载于 Fedor V. Fomin、Rusins Freivalds、Marta Z. Kwiatkowska 和 David Peleg 编，自动机、语言与编程，ICALP 2013，《计算机科学讲义》第 7966 卷，第 385–397 页。Springer，2013 年。 doi:10.1007/978-3-642-39212-2\35。_24 Tomas Petricek、Dominic A. Orchard 和 Alan Mycroft。共效应：上下文依赖计算的演算。载于 Johan Jeuring 和 Manuel M. T. Chakravarty 编，ACM 函数式编程国际会议，ICFP 2014，第 123–135 页。ACM Press，2014 年。 doi:10.1145/2628136.2628160。25 Emily Riehl。语境中的范畴论。Courier Dover Publications，2017 年。

26 James Wood 和 Robert Atkey。子结构类型系统的一个框架。载于 Ilya Sergey 编，欧洲编程研讨会，*ESOP 2022*，第 13240 卷，收录于《计算机科学讲义》，第 376–402 页。Springer，2022 年。doi:10.1007/978-3-030-99336-8_14。

第 5 节的证明

命题 3 的证明。条目 2 很直接，因为单元素集合是集合与函数范畴中的终对象。为证明条目 1，设 f: 自然数 $\to$ R 是一个等级代数同态，并注意到，由于 $n = 1 + \cdots + 1$ (n 次方)，对所有 $n \in \mathbb{N}$ 都成立，且 f保持和与单位元，因此有 $f(n) = f(1) +_R \cdots +_R f(1) = \mathbf{1}_R +_R \cdots +_R \mathbf{1}_R$ (n次方)。也就是说，对所有 $n \in \mathbb{N}$，都有 $f(n) = \iota_R(n)$。因此，为了得出结论，只需证明映射 ι_R 是一个等级代数同态。 $\iota_R(0) = \mathbf{0}_R$ 和 $\iota_R(1) = \mathbf{1}_R$ 这一事实显而易见。 $\iota_R(n+m) = \iota_R(n) +_R \iota_R(m)$ 以及 $\iota_R(n \cdot m) = \iota_R(n) \cdot_R \iota_R(m)$ 这一事实，可通过对 n 作直接归纳，并利用等级代数 R 的分配律和零元性质得出。最后，为证明单调性，考虑 $n \leq m$，并对 $m-n$ 作归纳。若 $m-n = 0$，则 $n = m$，所以命题显然成立。若 $m-n = k+1$，有 $m-(n+1) = k$，于是根据归纳假设可得 $\iota_R(n+1) \preceq_R (\iota_R(m)$。由于 $\iota_R(n+1) = \iota_R(n) +_R \iota_R(1)$ 且 $\mathbf{0}_R \preceq_R \iota_R(1)$，可得 $\iota_R(n) = \iota_R(n) +_R \mathbf{0}_R \preceq_R \iota_R(n) +_R \iota_R(1) \preceq_R \iota_R(m)$，如所需。◀

命题 26 的证明。 我们必须证明 $\preceq_H$ <style id='6}>具有自反性、传递性和反对称性。

给定一个元素 $\langle \kappa, r \rangle \in |G(H)|$，由于 $\kappa \sqsubseteq \kappa$ 和 $H_{\kappa,\kappa} = \mathsf{id}_{H_\kappa}$，根据 H 的函子性，我们有

$$H_{\kappa,\kappa}(r) = \mathsf{id}_{H_\kappa}(r) = r$$

因此 $\langle \kappa, r \rangle \preceq_H \langle \kappa, r \rangle$，这证明了自反性。

给定 $\mathsf{h}\kappa, r\mathsf{i} \preceq_H \mathsf{h}\mu, s\mathsf{i} \preceq_H \mathsf{h}\nu, t\mathsf{i}$，我们知道 $\kappa \vee \mu \vee \nu$ 和 $H_{\kappa,\mu}(r) \preceq_\mu s$ 以及$H_{\mu,\nu}(s) \preceq_\nu t$，并且根据 H 的函子性，$H_{\kappa,\nu} = H_{\mu,\nu} \circ H_{\kappa,\mu}$。因此，我们得到

$$H_{\kappa,\nu}(r) = H_{\mu,\nu}(H_{\kappa,\mu}(r)) \preceq_\nu H_{\mu,\nu}(s) \preceq_\nu t$$

因此 $\langle \kappa, r \rangle \preceq_H \langle \nu, t \rangle$，这证明了传递性。

给定 $\mathsf{h}\kappa, r\mathsf{i} \preceq_H \mathsf{h}\mu, s\mathsf{i} \preceq_H \mathsf{h}\kappa, r\mathsf{i}$，我们知道 $\kappa \vee \mu \vee \kappa$ 且 $H_{\kappa,\mu}(r) \preceq_\mu s$ 且$H_{\mu,\kappa}(s) \preceq_\kappa r$。由于 $\vee$ 具有反对称性，我们得到 $\kappa = \mu$，因此 $H_{\kappa,\mu} = H_{\mu,\kappa} = H_{\kappa,\kappa}$，而根据 H 的函子性，这等于恒等态射 id_{H_κ}。这意味着 $r \preceq_\kappa s$ 且 $s \preceq_\kappa r$，进而由 $\preceq_\kappa$ 的反对称性可得 $r = s$。◀

命题 27 的证明。 我们检查这四个性质。

$+_H$ 的单调性考虑 $\mathsf{h}\kappa, r\mathsf{i} \preceq_H \mathsf{h}\mu, s\mathsf{i}$ 和 $\mathsf{h}\kappa', r'\mathsf{i} \preceq_H \mathsf{h}\mu', s'\mathsf{i}$，则我们知道$\kappa \vee \mu$ 且 $\kappa' \vee \mu'$ 且 $H_{\kappa,\mu}(r) \preceq_\mu s$ 且 $H_{\kappa',\mu'}(r') \preceq_{\mu'} s'$。根据 $\mathsf{lh}^H_{\mu,\mu'}$ and $\mathsf{rh}^H_{\mu,\mu'}$ and $+_{\mu\oplus\mu'}$ 的单调性，我们得到

$$\mathsf{lh}^H_{\mu,\mu'}(H_{\kappa,\mu}(r)) +_{\mu\oplus\mu'} \mathsf{rh}^H_{\mu,\mu'}(H_{\kappa',\mu'}(r')) \preceq_{\mu\oplus\mu'} \mathsf{lh}^H_{\mu,\mu'}(s) +_{\mu\oplus\mu'} \mathsf{rh}^H_{\mu,\mu'}(s')$$

由 $\oplus$ 的单调性，我们得到 $\kappa\oplus\kappa' \sqsubseteq \mu\oplus\mu'$，然后，由于 H 的函子性以及 lh^H 和 $injr$ 的定义，我们有

$$\mathsf{lh}^H_{\mu,\mu'} \circ H_{\kappa,\mu} = H_{\kappa\oplus\kappa',\mu\oplus\mu'} \circ \mathsf{lh}^H_{\kappa,\kappa'} \qquad \mathsf{rh}^H_{\mu,\mu'} \circ H_{\kappa',\mu'} = H_{\kappa\oplus\kappa',\mu\oplus\mu'} \circ \mathsf{rh}^H_{\kappa,\kappa'}$$

因此，我们得到

$$H_{\kappa\oplus\kappa',\mu\oplus\mu'}(\mathsf{lh}^H_{\kappa,\kappa'}(r)+_{\kappa\oplus\kappa'}\mathsf{rh}^H_{\kappa,\kappa'}(r'))\preceq_{\mu\oplus\mu'}\\ \mathsf{lh}^H_{\mu,\mu'}(H_{\kappa,\mu}(r))+_{\mu\oplus\mu'}\mathsf{rh}^H_{\mu,\mu'}(H_{\kappa',\mu'}(r'))\preceq_{\mu\oplus\mu'}\\ \mathsf{lh}^H_{\mu,\mu'}(s)+_{\mu\oplus\mu'}\mathsf{rh}^H_{\mu,\mu'}(s')$$

这就证明了$\langle\kappa,\ r\rangle+_H\langle\kappa',\ r'\rangle\preceq_H\langle\mu,\ s\rangle+_H\langle\mu',\ s'\rangle$，如所需。

$+_H$的结合律考虑元素$\langle\kappa、\ r\rangle$、$\langle\mu、\ s\rangle$和$\langle\nu、\ t\rangle$，它们属于$|G(H)|$。利用方程（1）和（2），可得以下结果

$$\begin{aligned}(\langle\kappa,r\rangle+_H\langle\mu,s\rangle)+_H\langle\nu,t\rangle&=\langle\kappa\oplus\mu,\mathsf{lh}^H_{\kappa,\mu}(r)+\mathsf{rh}^H_{\kappa,\mu}(s)\rangle+_H\langle\nu,t\rangle\\&=\langle(\kappa\oplus\mu)\oplus\nu,\mathsf{lh}^H_{\kappa\oplus\mu,\nu}(\mathsf{lh}^H_{\kappa,\mu}(r)+\mathsf{rh}^H_{\kappa,\mu}(s))+\mathsf{rh}^H_{\kappa\oplus\mu,\nu}(t)\rangle\\&=\langle\kappa\oplus(\mu\oplus\nu),\mathsf{lh}^H_{\kappa,\mu\oplus\nu}(r)+(\mathsf{rh}^H_{\kappa,\mu\oplus\nu}(\mathsf{lh}^H_{\mu,\nu}(s)+\mathsf{rh}^H_{\mu,\nu}(t))\rangle\\&=\langle\kappa,r\rangle+_H(\langle\mu,s\rangle+_H\langle\nu,t\rangle)\end{aligned}$$

$+_H$的交换律考虑元素$\langle\kappa、\ r\rangle$和$\langle\mu、\ s\rangle$，它们属于$|G(H)|$。利用方程（3），可得以下结果

$$\begin{aligned}\langle\kappa,r\rangle+_H\langle\mu,s\rangle&=\langle\kappa\oplus\mu,\mathsf{lh}^H_{\kappa,\mu}(r)+_{\kappa\oplus\mu}\mathsf{rh}^H_{\kappa,\mu}(s)\rangle=\langle\kappa\oplus\mu,\mathsf{rh}^H_{\kappa,\mu}(s)+_{\kappa\oplus\mu}\mathsf{lh}^H_{\kappa,\mu}(r)\rangle\\&=\langle\mu\oplus\kappa,\mathsf{lh}^H_{\mu,\kappa}(s)+_{\mu\oplus\kappa}\mathsf{rh}^H_{\mu,\kappa}(r)\rangle=\langle\mu,s\rangle+_H\langle\kappa,r\rangle\end{aligned}$$

0_H的幺元性考虑$\langle\kappa、\ r\rangle$，它们属于$|G(H)|$。利用方程（3）、（5）和（6），我们得到以下结果

$$\begin{aligned}\langle\kappa,r\rangle+_H\mathbf{0}_H&=\langle\kappa\oplus I,\mathsf{lh}^H_{\kappa,I}(r)+_{\kappa\oplus I}\mathsf{rh}^H_{\kappa,I}(\mathbf{0}_I)\rangle=\langle\kappa,r+_\kappa\mathsf{in}^H_\kappa(\mathbf{0}_I)\rangle\\&=\langle\kappa,r+_\kappa\mathbf{0}_\kappa\rangle=\langle\kappa,r\rangle\end{aligned}$$

◀

命题 28 的证明。 由于in^H_κ是等级代数同态，因此有$\mathsf{in}^H_\kappa(\mathbf{0}_I)=\mathbf{0}_\kappa$，根据等级代数的定义，有$\mathbf{0}_\kappa\preceq_\kappa r$。因此，$\mathsf{in}^H_\kappa(\mathbf{0}_I\preceq_\kappa r$，根据$\preceq_H$的定义，这就证明了命题。◀

命题 29 的证明。 要证明幺半群的等式公理（结合律和幺元性），当所有涉及的元素都不等于$\mathbf{0}_I$时，证明与命题 27 相同；否则则显然成立。事实上，如果其中某个元素是$\mathbf{0}_I$，那么整个乘法根据定义得到$\mathbf{0}_I$。

要证明$\cdot_H$的单调性，考虑$|G(H)|$中的$\mathbf{h}\kappa,r\mathbf{i}\preceq_H\mathbf{h}\mu,s\mathbf{i}$和$\mathbf{h}\kappa',r'\mathbf{i}\preceq_H\mathbf{h}\mu',s'\mathbf{i}$。若它们都不等于$\mathbf{0}_I$，证明过程与命题 27 相同。若$\mathbf{h}\kappa,r\mathbf{i}=\mathbf{0}_I$或$\mathbf{h}\kappa',r'\mathbf{i}=\mathbf{0}_I$，则$\mathbf{h}\kappa,r\mathbf{i}\cdot_H\mathbf{h}\kappa',r'\mathbf{i}=\mathbf{0}_I$，因此由命题 28 可得该命题。若$\mathbf{h}\mu,s\mathbf{i}=\mathbf{0}_I$，则$\mathbf{h}\kappa,r\mathbf{i}\preceq_H\mathbf{h}\mu,s\mathbf{i}=\mathbf{h}I,\mathbf{0}_I\mathbf{i}$蕴含$\kappa\ \mathrm{v}\ I$，且$H_{\kappa,I}(r)\preceq_I\mathbf{0}_I$；又由于根据等级签名的定义，$I\mathrm{v}\ \kappa$，可得$\kappa=I$。因此，由$H$的函子性，有$H_{\kappa,I}=\mathrm{id}_{H_I}$，故$r\preceq_I\mathbf{0}_I$，这蕴含$r=\mathbf{0}_I$，因为$H_I$是等级代数。这证明了$\mathbf{h}\kappa,r\mathbf{i}=\mathbf{h}I,\mathbf{0}_I\mathbf{i}=\mathbf{0}_H$，因此得到$\mathbf{h}\kappa,r\mathbf{i}\cdot_H\mathbf{h}\kappa',r'\mathbf{i}=\mathbf{0}_H=\mathbf{h}\mu,s\mathbf{i}\cdot_H\mathbf{h}\mu',s'\mathbf{i}$，如所需。最后，情形$\mathbf{h}\mu',s'\mathbf{i}=\mathbf{h}I,\mathbf{0}_I\mathbf{i}$类似，因此该命题得证。◀

定理 30 的证明。根据命题27和命题29，我们得到了加法和乘法幺半群结构。命题 28 证明了$\mathbf{0}_H$是序$\preceq_H$的最小元。根据定义，乘以$\mathbf{0}_I$后仍得到$\mathbf{0}_I$。因此，接下来只需证明$\cdot_H$对$+_H$具有分配性。为此，考虑$\langle\kappa,r\rangle$、$\langle\mu,s\rangle$和$\langle\nu,t\rangle$属于$|G(H)|$

并假设它们都不等于 $\mathbf{0}_I$。利用方程 (1)–(4)，我们得到以下方程：

$$
\begin{aligned}
\mathsf{lh}^H_{\kappa\oplus\mu,\kappa\oplus\nu}\circ\mathsf{lh}^H_{\kappa,\mu} &= \mathsf{lh}^H_{(\kappa\oplus\mu)\oplus\kappa,\nu}\circ\mathsf{lh}^H_{\kappa\oplus\mu,\kappa}\circ\mathsf{lh}^H_{\kappa,\mu}\\
&= \mathsf{lh}^H_{\kappa\oplus(\kappa\oplus\mu),\nu}\circ\mathsf{rh}^H_{\kappa,\kappa\oplus\mu}\circ\mathsf{lh}^H_{\kappa,\mu}\\
&= \mathsf{lh}^H_{(\kappa\oplus\kappa)\oplus\mu,\nu}\circ\mathsf{lh}^H_{\kappa\oplus\kappa,\mu}\circ\mathsf{lh}^H_{\kappa,\kappa}\\
&= \mathsf{lh}^H_{\kappa\oplus\mu,\nu}\circ\mathsf{lh}^H_{\kappa,\mu} = \mathsf{lh}^H_{\kappa,\mu\oplus\nu}\\
\mathsf{rh}^H_{\kappa\oplus\mu,\kappa\oplus\nu}\circ\mathsf{lh}^H_{\kappa,\nu} &= \mathsf{lh}^H_{(\kappa\oplus\mu)\oplus\kappa,\nu}\circ\mathsf{rh}^H_{\kappa\oplus\mu,\kappa} = \mathsf{lh}^H_{\kappa\oplus(\kappa\oplus\mu),\nu}\circ\mathsf{lh}^H_{\kappa,\kappa\oplus\mu}\\
&= \mathsf{lh}^H_{(\kappa\oplus\kappa)\oplus\mu,\nu}\circ\mathsf{lh}^H_{\kappa\oplus\kappa,\mu}\circ\mathsf{lh}^H_{\kappa,\kappa}\\
&= \mathsf{lh}^H_{\kappa\oplus\mu,\nu}\circ\mathsf{lh}^H_{\kappa,\mu}\\
&= \mathsf{lh}^H_{\kappa,\mu\oplus\nu}\\
\mathsf{lh}^H_{\kappa\oplus\mu,\kappa\oplus\nu}\circ\mathsf{rh}^H_{\kappa,\mu} &= \mathsf{lh}^H_{(\kappa\oplus\mu)\oplus\kappa,\nu}\circ\mathsf{lh}^H_{\kappa\oplus\mu,\kappa}\circ\mathsf{rh}^H_{\kappa,\mu}\\
&= \mathsf{lh}^H_{\kappa\oplus(\kappa\oplus\mu),\nu}\circ\mathsf{rh}^H_{\kappa,\kappa\oplus\mu}\circ\mathsf{rh}^H_{\kappa,\mu} = \mathsf{lh}^H_{(\kappa\oplus\kappa)\oplus\mu,\nu}\circ\mathsf{rh}^H_{\kappa\oplus\kappa,\mu}\\
&= \mathsf{rh}^H_{\kappa,\mu\oplus\nu}\circ\mathsf{lh}^H_{\mu,\nu}\\
\mathsf{rh}^H_{\kappa\oplus\mu,\kappa\oplus\nu}\circ\mathsf{rh}^H_{\kappa,\nu} &= \mathsf{rh}^H_{(\kappa\oplus\mu)\oplus\kappa,\nu} = \mathsf{rh}^H_{\kappa\oplus\mu,\nu}\\
&= \mathsf{rh}^H_{\kappa,\mu\oplus\nu}\circ\mathsf{rh}^H_{\mu,\nu}
\end{aligned}
$$

由此可推出

$$
\begin{aligned}
&(\langle\kappa,r\rangle\cdot_H\langle\mu,s\rangle)+_H(\langle\kappa,r\rangle\cdot_H\langle\nu,t\rangle) =\\
&\quad= \langle(\kappa\oplus\mu)\oplus(\kappa\oplus\nu),\mathsf{lh}^H_{\kappa\oplus\mu,\kappa\oplus\nu}(\mathsf{lh}^H_{\kappa,\mu}(r)+\mathsf{rh}^H_{\kappa,\mu}(s))+\mathsf{rh}^H_{\kappa\oplus\mu,\kappa\oplus\nu}(\mathsf{lh}^H_{\kappa,\nu}(r)+\mathsf{rh}^H_{\kappa,\nu}(t))\rangle\\
&\quad= \langle\kappa\oplus(\mu\oplus\nu),(\mathsf{lh}^H_{\kappa,\mu\oplus\nu}(r)\cdot\mathsf{rh}^H_{\kappa,\mu\oplus\nu}(\mathsf{lh}^H_{\mu,\nu}(s)))+(\mathsf{lh}^H_{\kappa,\mu\oplus\nu}(r)\cdot\mathsf{rh}^H_{\kappa,\mu\oplus\nu}(\mathsf{rh}^H_{\mu,\nu}(t)))\rangle\\
&\quad= \langle\kappa\oplus(\mu\oplus\nu),\mathsf{lh}^H_{\kappa,\mu\oplus\nu}(r)\cdot\mathsf{rh}^H_{\kappa,\mu\oplus\nu}(\mathsf{lh}^H_{\mu,\nu}(s)+\mathsf{rh}^H_{\mu,\nu}(t))\rangle\\
&\quad= \langle\kappa,r\rangle\cdot_H(\langle\mu,s\rangle+_H\langle\nu,t\rangle)
\end{aligned}
$$

这在所有元素都不等于 $\mathbf{0}_H$ 时证明了分配律。

现在，假设 $\mathrm{h}\kappa,r\mathrm{i}=\mathbf{0}_H$，那么我们有 $\mathrm{h}\kappa,r\mathrm{i}\cdot_H(\mathrm{h}\mu,s\mathrm{i}+_H\mathrm{h}\nu,t\mathrm{i})=\mathrm{h}\kappa,r\mathrm{i}\cdot_H\mathrm{h}\mu,s\mathrm{i}=\mathrm{h}\kappa,r\mathrm{i}\cdot_H\mathrm{h}\nu,t\mathrm{i}=\mathbf{0}_H$，因此分配律显然成立。最后，假设 $\mathrm{h}\mu,s\mathrm{i}=\mathbf{0}_H$（$\mathrm{h}\nu,t\mathrm{i}=\mathbf{0}_H$ 的情形类似），那么我们有 $\mathrm{h}\mu,s\mathrm{i}+_H\mathrm{h}\nu,t\mathrm{i}=\mathrm{h}\nu,t\mathrm{i}$，且 $\mathrm{h}\kappa,r\mathrm{i}\cdot_H\mathrm{h}\mu,s\mathrm{i}=\mathbf{0}_H$。因此，我们得到

$$
\begin{aligned}
\langle\kappa,r\rangle\cdot_H(\langle\mu,s\rangle+_H\langle\nu,t\rangle) &= \langle\kappa,r\rangle\cdot_H\langle\nu,t\rangle = \mathbf{0}_H+_H\langle\kappa,r\rangle\cdot_H\langle\nu,t\rangle\\
&= \langle\kappa,r\rangle\cdot_H\langle\mu,s\rangle+_H\langle\kappa,r\rangle\cdot_H\langle\nu,t\rangle
\end{aligned}
$$

◀