Ziv：用于正确舍入数学库的语言

LUCAS YUNKYU LEE，浦项科技大学，韩国
SUNGJIN HWANG，浦项科技大学，韩国
WONYEOL LEE，浦项科技大学，韩国

开发高性能的正确舍入数学库具有挑战性。此类实现通常依赖扩展精度算术来实现正确舍入，并依赖预计算常数来提升性能。然而，现有实现语言对以下方面的支持有限：（i）针对此类算术的高级抽象，以及（ii）将预计算与运行时计算结合在同一程序中的分阶段计算。因此，现有实现往往错失抽象机会，并将使用独立程序预计算的常数硬编码进去。这些做法使得编写、理解和修改这些库变得困难。

我们提出 Ziv，一种用于开发正确舍入数学库的编程系统。它由一种小型语言、一个丰富的标准库，以及一个面向C语言、可证明可靠的编译器组成。Ziv 支持：（i）通过面向扩展精度数的类型与运算实现高级数值抽象，以及（ii）通过对已知独立于输入的表达式进行编译时求值来实现分阶段计算。这些特性使高级代码能够统一预计算与运行时计算。我们将 Ziv 嵌入 Julia，并通过重新实现两个面向正确舍入的最先进库中的19个函数来评估它；我们在代码行数减少 2.63× 的同时取得了有竞争力的性能。我们还实现了三个新函数，并自动探索了824,256个变体，相对任意精度库实现了 96.5× 的平均加速。

arXiv:2611.30346v1 [cs.PL] 23 Nov 2026

1 引言

一个数学库是实现广泛使用的数学函数（例如指数函数和三角函数）的软件，也是编程语言的核心组件 [43, 50, 53, 62]。数学库若满足以下条件，便是正确舍入的：每个函数都返回精确数学结果舍入到目标浮点格式后的值 [26]：对于每个库函数f 和浮点输入x，都有f(x) = $rnd(f(x))$。这里，f是与f相对应的精确数学函数，而rnd会根据给定的舍入模式，将实数舍入为浮点数（或浮点数）。

正确舍入被视为数学库的黄金标准，在研究 [9] 和实践 [42, 57]中也日益重要。根据定义，正确舍入的数学库
有两大优点：它能达到目标格式和舍入模式所允许的最高准确性；并且具有可复现性，在不同平台上始终返回相同的输出。体现这些优点，C标准最近为正确舍入函数保留了 cr_* 标识符（例如 cr_指数函数） [28], ，而下一版 IEEE 浮点标准预计将要求核心函数 [27]正确舍入。

然而，开发实用的正确舍入库通常颇具挑战 [46, 47]。要实现正确舍入，实现必须具有极高的精度；这往往需要扩展精度计算（例如，多字算术）。要满足实用性，实现必须具备高性能；这往往需要进行各种优化（例如，预先计算常数）。每项要求都会增加实现的复杂性，而要同时满足两者，则会使实现变得极其复杂。例如，CORE-MATH [57]和 LLVM-libc [42],这两个以正确舍入为目标的最先进库，分别使用 161K 行和 61K 行 C语言及 C++ 代码，实现了 193 个和 117 个函数。

我们的目标是从编程语言的角度，让开发正确舍入数学库变得更容易。为此，我们提出了Ziv 编程系统。我们关注使用 64 位或更多位的定精度浮点格式上的函数；对于这些函数，穷举测试 [41]不可行。现在我们将讨论关键挑战（§1.1）以及我们的方法（§1.2）。

1.1 现有方法与关键挑战

现有以正确舍入为目标的数学库 [13, 25, 42, 57, 58]通常使用 C 语言或 C++编写，并具有两个实现特点。首先，高级计算（例如定点算术）通常用低级运算（例如整数算术）来表达。其次，数值常数

作者联系信息：Lucas Yunkyu Lee，lucaslee@postech.ac.kr；Sungjin Hwang，sungjin.hwang@postech.ac.kr；Wonyeol Lee，wonyeol.lee@postech.ac.kr。所有作者均隶属于韩国浦项的浦项科技大学。

LLVM-libc：对数

```
size_t i3 = ((int64_t)(0x284ull ))  >> 55;
Int128 v3 = ((((Int128)S3[i3] << 55) + v2) << 21)
          + (Int128)S3[i3] * v2;
大小_t i4 =((整数)(v3 >> 68)  + 131)  >> 1;
Int128 v4 = ((((Int128)S4[i4] << 69) + v3) << 28)
          + (Int128)S4[i4] * v3;
```

CORE-MATH：反正切

```
常量双精度浮点数表[129][2] = {{0x0p+0,  0x0p+0},
  {0x1.9224e047e368ep-7, 0x1.a3ca6c727c59dp-62},
  {0x1.92346247a91f0p-6, 0x1.138b0ef96a186p-64},
  /*... 其余126行 ... */ };
常量双精度浮点数 系数[5] = {0x1p+0,
  -0x1.5555555555555p-2,  0x1.99999999998c1p-3,
  -0x1.249249176aec0p-3,  0x1.c711fd121ae80p-4};
```

(a) 现有实现。

Ziv：来自 LLVM-libc 的对数

```
i3 = round(Int, ldexp(v2,  21), 舍入到最近值, 平局时向上) + 80
v3 = (S3[i3] + v2) + S3[i3] * v2
i4 =四舍五入(整数, ldexp(v3,  28), 舍入到最近值, 平局时向上) + 65
v4 = (S4[i4] + v3) + S4[i4] * v3
# vj,  Sj[k]具有 128 位定点类型。
```

Ziv：来自 CORE-MATH 的反正切

```
表 = 制表(i-> i==128 ?(0.0,          0.0) :
  令 θ = 符号"$i*pi/256",  t = Float64(正切(θ));
     (t, Float64(反正切(Sym(t))-θ))
  结束; 形状=129)
系数 = Float64。(gridremez(
  x-> x==0? 1: 反正切(sqrt(x))/sqrt(x);
  次数=4,   定义域=(0,  XMAX^2)))
```

(b) Ziv 中对应的实现。

先进的数学库中针对正确舍入的编辑摘录 [42, 57] 。

（例如，表项）由单独的程序使用外部工具（例如 Sage [63]）预先计算，然后硬编码。图1a 通过最先进的库中的摘录说明了这些特点。

这些特性反映了实现语言的两个主要局限。首先，这些语言对高层数值抽象的支持有限：即能够直接表达扩展精度计算的类型和运算。例如，图1a （上）对 128 位定点数执行基本算术运算。由于缺乏对这些数的内置支持，即使是这些简单计算，也需要一系列复杂的整数运算，并使用看似随意的常数。

其次，这些语言对分阶段计算的支持有限：有些代码在编译时求值，其余代码则在运行时求值。例如，生成图1a（下方）中的常数，需要以高精度计算实函数，并使用复杂的多项式逼近方法。由于这些计算难以在编译时完成，开发人员会使用独立的程序（通常不公开）预先计算这些常数，并将结果硬编码。

这些局限带来了三个难题，如图1a所示。首先，编写新的函数实现很困难。用低级代码表达高级计算，以及预先计算并硬编码常数，都费时费力且容易出错。其次，理解现有实现很困难。仅凭代码往往难以推断其中的高级计算和常数的数学定义，有时甚至根本无法推断。第三，修改现有实现很困难。即使已经理解了这些计算和定义，一处细微的高级改动也可能需要大幅修改低级代码，并重新计算常数。

1.2 我们的方法与主要贡献

我们介绍了 Ziv，这是一个用于开发正确舍入数学库的编程系统。Ziv 由一种语言、标准库和编译器组成，其设计围绕两项原则展开：**DP1.** 为 正确舍入数学库中常见的技术提供高级抽象。**DP2.** 支持 阶段化，以统一扩展精度预计算与高效运行时计算。

Ziv 语言提供了一组刻意精简且可组合的构造，我们以此构建*ZivStd*，这是一个内容丰富的数值运算与算法标准库。Ziv 编译器通过部分求值和符号追踪，将涉及这些构造及库函数的程序转换为 C 代码。为确立这种翻译的正确性，我们对该语言和编译器进行了形式化定义，并证明成功编译能够保留 Ziv 程序的语义。

该Ziv 语言和*ZivStd* 库提供了高层数值抽象(DP1)，解决了§1.1中讨论的第一个局限。该Ziv 语言提供了扩展精度数的类型，包括精确实数和定点数，而*ZivStd* 库则提供了这些类型的运算。这些抽象使开发者能够直接表达数值计算，而无需编写低级代码。例如，图1b (上)使用 128 位定点类型表示`vj` 和`Sj[k]`, ，并使用定点运算 + 和* 来计算`vj`。这些

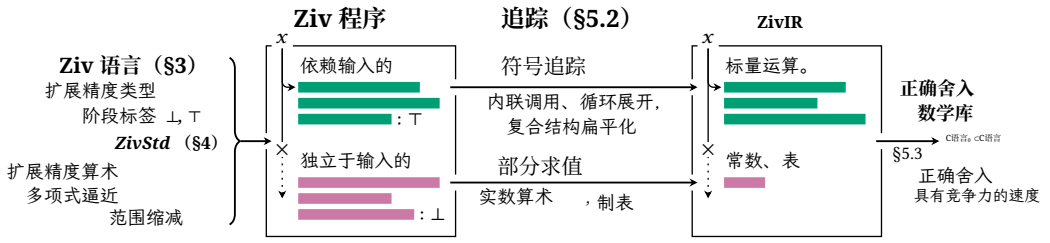


图 2. Ziv 编程系统概览 tem.

运算在内部处理了图1a (上)所示的复杂整数算术和位移。该Ziv 语言和编译器支持分阶段计算(DP2)，解决了§1.1中讨论的第二个局限。Ziv的类型系统能够可靠地跟踪表达式对运行时输入的依赖关系，Ziv编译器会在编译时求值所有已知独立于输入的表达式。这些特性共同使开发者能够在单个Ziv 程序中同时编写预计算和运行时计算，而无需使用单独的程序或硬编码常数。例如，图1b (下)使用精确实数和浮点表达式定义一个表，例如`Float64`(正切($i\pi$/256))。类型系统会检查该表是否独立于输入，因此编译器会自动预计算表项，并将其嵌入生成的 C 代码中（图2，紫色）。

为了评估 Ziv 如何解决§1.1中指出的难题，我们进行了三项案例研究。

- 我们使用 Ziv 重新实现了两个先进的数学库 CORE-MATH 和 LLVM-libc 中的 19 个代表性函数。我们的实现返回相同的结果，性能也具有竞争力，同时将源代码行数分别减少到原来的 1/2.17 和 1/3.59。

- 我们使用 Ziv 实现了三个现有库（任意精度库除外）均不支持正确舍入的函数。对于 10^9 个随机输入，我们实现了正确舍入；与先进的任意精度库 MPFR 相比，速度提升了 22.4–696.8× 倍。

- 我们通过改变工作精度、表大小等参数，使用 Ziv 自动探索这三个函数的 824,256个变体。我们发现，其中的变体速度比手写实现快 1.6–2.4× 倍，同时不牺牲准确性；此外，还发现了许多准确性较低但速度快得多的变体。

这些结果表明，Ziv 能够有效地简洁表达现有实现、为新函数开发实用实现，并探索大量实现变体。

贡献。 总而言之，本文的主要贡献如下。

- **Ziv 语言（§2 和 §3）。 我们介绍了**Ziv，这是一种用于开发正确舍入数学库的静态类型命令式语言。它提供高级数值抽象（DP1），并支持类型引导的分阶段计算（DP2）。它刻意保持小巧，同时具有很强的表达能力和可扩展性。我们形式化定义了其语法、类型系统和语义，并证明了类型保持。

- **ZivStd 库（§4）。 我们识别出数学库中常见的高层抽象，包括扩展精度算术及其变体；范围归约和多项式近似/求值方法；以及表函数和优化函数**（DP1）。我们在 ZivStd 中系统地组织并实现了这些抽象。*ZivStd* 是一个功能丰富的标准库，用于数学库开发。

- **Ziv 编译器（§5）**。我们介绍一种编译器，它通过部分求值和符号追踪将 Ziv 程序转换为 C 代码。我们形式化了编译过程，并证明它能保留程序语义，且会预计算所有已知独立于输入的表达式。这使开发者能够在同一个程序中表达预计算和运行时计算（DP2）。

- **评估（§6 和§7）**。我们将Ziv 实现为 Julia 中的一种嵌入式语言。我们展示了Ziv能够表达两个先进数学库中的19个函数，同时使源代码行数减少2.17–3。59×我们还实现了三个新函数，并系统地探索了824,256个变体，相较于任意精度库，实现了22.4–696。8× 倍的加速。

2 概述：正确舍入数学库与 Ziv

实现高性能的正确舍入函数，关键在于确定如何（§2.1）、何处（§2.2）以及何时（§2.3）使用更高精度。我们概述正确舍入数学库如何做出这些决策，以及Zɪv 如何简化其实现，并使用Zɪv的具体语法进行说明。

2.1 如何使用更高精度：扩展精度类型

首先要决定的是如何使用比机器浮点数所支持精度更高的表示方式。我们重点介绍三种可提供扩展精度的表示：实数、多字数和二进有理数。Zɪv 在核心语言中支持这些表示（§3），并在标准库中提供对它们的运算（§4）。

实数。数学函数实现需要高精度常数，例如用于范围缩减和多项式求值的常数（§2.2）。在运行时计算这些常数效率低下，而使用机器浮点数进行预计算可能无法达到足够的精度。作为替代，可以在编译时执行实数算术，将实数结果舍入到有限精度，并将其折叠进程序以供运行时使用。为此，Zɪv 提供了一种语法受限的实数类型，以确保对其进行的所有计算仅在编译时求值（§3.1）。该语言和标准库还支持实数算术以及舍入到有限精度类型。

多字数。多字数 [14] 由多个机器浮点字组成，表示这些字的未计算总和。每个后续字都存储一个修正值，该值太小，无法保留在前面的字中；如果计算总和，这些修正值就会丢失。例如，将 $1 + 2^{-58}$ 计算为 64 位浮点数会舍入为 1，但二元组 $(1,\ 2^{-58})$ 保留了所有位。

多字数的算术运算使用无误差变换 [4], 运算，同时计算舍入结果及其舍入误差。例如，给定合适的浮点数输入 $(a,\ b)$，*fast2sum* [14,56]会返回 $(s,\ r)$，使得 $s = a \oplus b$和 $s + r = a + b$ 精确成立，其中 $\oplus$表示浮点加法。下面的 C 代码使用两次 fast2sum 运算实现双字加法。Ziv 使用其双字类型 Float64x2 和中缀 + 运算符，更清晰地表达了相同的意图。

双字加法 C

```c
双精度x_高位 = 1.0,       x_lo = 0x1p-58;
双精度y_高位 = 0x1p-53, y_低位 = 0.0;
双精度 s    =x_高位 + y_高位, r  = y_你好 -(s-x_hi);
双精度 低   =(x_lo + y_lo) + r;
双精度z_嗨 = s  + 低,  z_低 = 低 - (z_嗨-s);
```

双字加法 Zɪv

```
x = Float64x2(1.0, 0x1p-58)
y = Float64x2(0x1p-53, 0.0)
z = x + y
```

二进制有理数。浮点二进制有理数 [18] 以整数三元组 (s, m, e) 的形式存储，分别对应数值的符号、有效数和指数。表示的值为 $(-1)^s \times m \times 2^e$。例如，使用 128 位无符号整数来表示 m，可提供 128 位精度。浮点二进制有理数上的运算通过整数算术实现。

定点二进制有理数 也基于整数，但指数 e 由类型固定。因此，其表示形式为符号和有效数的二元组 (s, m)，其值为 $(-1)^s \times m \times 2^e$。在下面的定点算术示例中，C语言使用手动定点对齐，要求程序员跟踪每个值的指数。相比之下，Ziv 会在其 Dyadic 类型中存储并跟踪指数，支持定点算术运算符，也支持由浮点数构造。

定点算术 C

```c
无符号整数128_tx =(无符号整数128_t)3 << 15;          // *  2^-16
uint128_t y = (uint128_t)1 << 58;          // *  2^-60
无符号整数128_tz =(x<<60)  + (y<<16)  + x *y;//* 2^-76
```

定点算术 Zɪv

```
x = 二进有理数{128,值{-16}}(1.5)
y = 二进有理数{128,值{-60}}(0.25)
z = x +y + x*y; @assert z是 二进有理数{128,值{-76}}
```

2.2 在何处使用更高精度：实现步骤

接下来要决定的是在哪里使用更高精度。实现必须有选择地使用扩展精度类型，以平衡准确性与性能。这一决策贯穿以下三个

正确舍入数学库的实现步骤：范围缩减、多项式求值和重构 [9]。我们以一种可能的指数函数 e^x 实现为例。

范围缩减。 由于浮点数的动态范围很大，函数的输入可能极大，因而难以直接计算。范围缩减利用数学恒等式将输入缩减到一个较小的范围，从而解决这一问题，使得近似结果更加准确。

对于 e^x，我们希望将 x缩减为某个 r，使得 $|r|$较小，以便近似计算 e^r，而不是直接计算e^x。为此，我们选择一个整数 k，使得 $\ln_{2^{12}}^2 k$是最接近 x的 $\ln_{2^{12}}^2$的整数倍。然后，将 r定义为余数 $x - \ln_{2^{12}}^2 k$，即可得到界限 $|r| \le \ln_{2^{13}}^2$。在精确实数运算中：

$$e^x = 2^{k/2^{12}} e^r \qquad \text{for } x = \frac{\ln 2}{2^{12}} k + r, \qquad \text{where } k = \operatorname{round}\left(x \,/\, \frac{\ln 2}{2^{12}}\right).$$

可以用多项式在 $\left[-\frac{\ln 2}{2^{13}}, \ln_{2^{13}}^2\right]$上近似计算缩减后的指数函数 e^r，而缩减后的输入 r可以通过扩展精度算术精确计算，包括将 $\ln_{2^{12}}^2$存储为多字数。不过，直接计算 $2^{k/2^{12}}$需要计算代价高昂的分数次幂。

相反，基于查表的方法 [60, 61]将运行时计算替换为查找预先计算好的值。我们将 $2^{k/2^{12}} = 2^i 2^{j/2^{12}}$改写为 $k = 2^{12} i + j$，其中整数 i和 j，并制表计算分数次幂 $T[j] \approx 2^{j/2^{12}}$。下面的示例将 T存储为双字，以确保精度足够。C语言要求程序员在外部计算表中的值，并将其硬编码为常数。相比之下，Ziv 代码使用`tabulate`，它会在编译时对每个索引应用一个函数来制表（§4.2）。此外，还提供了常见范围缩减算法的完整实现，包括 Cody-Waite [12] 和 Payne-Hanek [52] 方法（§4.2）。

基于查表的范围缩减 C

```c
静态常量双精度TAB[4096][2] = {
  {0x1.0000000000000p+0, 0x0.0000000000000p+00},
  {0x1.000b175effdc7p+0, 0x1.ae8e38c59c72ap-54},
  /* ... 4094 more lines ... */ };
doublea_嗨 = ldexp(TAB[j][0], i),  // 重构
       a_lo = ldexp(TAB[j][1], i);
```

基于查表的范围缩减 Ziv

```
表 =制表(j-> Float64x2(符号"2^($j* 2^-12)"),
                2^12)
一个 =ldexp(表[j], i)  # 重构
```

多项式求值。 范围缩减后，我们使用多项式在缩减后的区间上逼近并求值该函数。我们用多项式逼近 仅指系数的预计算。逼近方法包括泰勒多项式和各种极小极大多项式 [8, 11]。必要时，可以使用扩展精度计算并存储系数。

使用预计算系数进行运行时计算称为多项式求值。下面，我们用其三次泰勒多项式近似 e^r，并使用霍纳方法 [33], 通过双字算术对其求值。在 C语言中，多项式通过外部工具进行近似并硬编码到程序中，求值则手动完成。Ziv 在单个程序中完成所有这些步骤。使用`taylor` 进行近似并将系数舍入为`Float64x2`均在编译时完成。在运行时，使用霍纳(§4.2)对近似多项式求值。

泰勒近似与求值 C

```c
静态常量 双精度 C语言[4][2] = {
  {1.0, 0.0}, {1.0, 0.0}, {0.5, 0.0},
  {0x1.5555555555555p-3, 0x1.5555555555555p-57} };
双精度浮点数y_高位 = C语言[3][0], y_低位 = C语言[3][1],    ry_hi,   ry_lo;
对于 (整数 k = 2; k >= 0; k--)
  dw_mul(r_hi,r_lo, y_hi,y_lo,    &ry_嗨, &ry_lo),
  dw_add(C语言[k][0],C语言[k][1],   ry_hi,ry_lo,    &y_嗨, &y_lo);
```

泰勒近似与求值 Ziv

```
C语言 = Float64x2。(taylor(指数函数,  3, 0))
y = 霍纳(r,    C)
```

重构。 最后的重构 步骤将前面各步骤的结果组合起来，生成最终输出。因此，所需的计算自然地由范围约化方案推导而来。对于指数函数，若 $P(r)$ 是 e^r 的多项式逼近，我们可以使用范围缩减期间定义的表 T，执行重构 $e^x \approx 2^i \otimes T[j] \otimes P(r)$。这项计算会产生舍入误差，可通过使用扩展精度来减少。

2.3 何时使用更高精度：预计算与 Ziv 方法

剩下的最后一个决定是何时 使用更高精度。通过在编译时进行预计算，可以在不增加运行时开销的情况下使用更高精度。即使在运行时，也并非总是需要额外精度；只有少数可由 Ziv 方法判定的情况才需要更高精度。

预计算。预计算使独立于输入的值能够以高精度预先计算，并通过将常数内联到程序中而不产生运行时开销。对于指数函数，可预计算的值包括常数 $\ln_{2^{12}}^{2}$、表项 $T[j]$ 和多项式系数。

与硬编码常数的传统方法不同，Zɪv 将预计算与运行时代码整合到一个程序中。Zɪv 编译器会计算独立于输入的代码，并将结果嵌入编译后的程序中，只留下依赖输入的代码供运行时执行（§5.2）。

Ziv 方法。到目前为止，我们一直假设每次计算所用的精度是预先选定的，但正确舍入所需的精度实际上取决于输入。*Ziv 方法* [66]使用误差界来检验，对于给定输入，近似值是否足以实现正确舍入。这样便可仅在必要时使用更高精度，从而实现高效的实现；运行时执行会根据输入分为快速路径和精确路径。快速路径使用较低的精度和较低的多项式次数，只有在舍入检验通过时才返回结果。否则，精确路径会以更高精度和更高的多项式次数重新计算。

2.4 端到端示例：指数函数

现在，我们将 §§2.1–2.3 中的技术结合起来，实现 CORE-MATH 的 [57] 指数函数。图 3 比较了 CORE-MATH 指数函数的简化版本，分别采用原始 C语言和 Zɪv 实现。

A：特殊情况。 首先，先筛除特殊输入情况并立即返回，将其余输入留给快速路径。C语言通过将输入位模式与硬编码整数进行比较来处理，而Zɪv 则用谓词和函数来表达这些情况。优化函数 最大值参数 和最小值参数 (§4.3) 用于计算 Zɪv 第 11 行和12行中的下溢和上溢阈值。

B：范围缩减。 快速路径使用类似于 §2.2 中的方法，将输入 x 缩减为 dx。C语言仍使用硬编码常数和表，而 Zɪv 则在代码中清晰地定义它们。第 18 行和19中，Zɪv 代码使用 `sym` 创建实数值，并在编译时将其舍入为双字。这里，Zɪv的双字类型可以容纳非机器类型，其中高位部分的类型为 `Dyadic{29,Big}`，即精度为 29 位的浮点二进定点数。第 1–5 行中的 tabulate 调用也使用实数表达式来生成包含双字的查找表。

C语言：多项式求值。 缩减后的自变量用于近似 e^{dx}，即 `1+dx·p`，其中p 是自变量 dx处多项式的值。C 代码使用手写的 Estrin 方案 [16]来计算外部近似得到的系数。在 Ziv 的第 22–24 行中，通过 `gridremez` [54] 进行系数近似并舍入为 `Float64`，这些操作均在编译时完成，从而得到与 C 版本完全相同的三次多项式。在运行时，第 25 行使用 Estrin (§4.2)计算该多项式。

D：重构。 为了重构完整的指数函数，将求得的多项式与查表因子 `t` 通过双字算术相乘。在这里，`t` 是将表元素 `TAB1[idx1]` 与 `TAB2[idx2]` 通过第 27 行中的双字乘法运算符 * 相乘得到的，该行位于 Zɪv 代码中。第 28 行中，求得的多项式进行双字乘法时改用变体 $*_{\mathrm{h}}$，该变体省略了无误差变换，以更高的速度换取准确性。

E：*Ziv 方法*。快速路径结束时，重构出的结果会进行 Ziv 舍入测试。为了确保次正规数舍入正确，测试会根据预期输出是次正规数（Zɪv 第 33–36 行）还是正规数（第 38–40 行）而分别执行。如果测试通过，结果就是正确舍入的，并会被返回。否则，指数函数_精确路径会在更高精度下于第 36 行和第 40 行重新计算，并从范围缩减步骤重新开始。图中省略了这条精确路径。

CORE-MATH（原始 C语言，简化版）1 static const double TAB1[64][2] = { 2 { 0x0.0000000000000p+00, 0x1.0000000000000p+0}, 3 {-0x1.19083535b085ep-56, 0x1.02c9a3e778061p+0}, 4 { 0x1.d73e2a475b466p-55, 0x1.059b0d3158574p+0}, 65 /* ... 61 more lines ... */ }; 66 static const double TAB2[64][2] = { 67 { 0x0.0000000000000p+00, 0x1.0000000000000p+0}, 68 { 0x1.ae8e38c59c72ap-54, 0x1.000b175effdc7p+0}, 69 {-0x1.7b5d0d58ea8f4p-58, 0x1.00162f3904052p+0}, 130 /* ... 61 more lines... */ }; 131 132 double 指数函数(double x) { 133 union {double f; uint64_t u;} 重新解释 = {.f = x}; 134 uint64_t ax_u = 重新解释.u & (~(uint64_t)0 >> 1); 135 if (ax_u <= 0x3c900000000000ull) 返回 1.0 + x; 136 if (ax_u >= 0x40862e42fefa39f0ull) { 137 if (ax_u > 0x7ff0000000000000ull) 返回 x + x; 138 if (ax_u == 0x7ff0000000000000ull) { 139 if (重新解释.u >> 63) 返回 0.0; 140 else 返回 x; } 141 if (!(重新解释.u >> 63)) { 142 volatile double z = 0x1p1023; 143 返回 z * z; } 144 if (ax_u >= 0x40874910d52d3052ull) { 145 volatile double z = 0x1p-1022; 146 返回 z * z; } 147 } 148 149 double k_f = roundeven(x * 0x1.71547652b82fep+12); 150 int64_t k =k_f, idxe = k >> 12; 151 int64_t idx1 = (k >> 6) & 0x3f, idx2 = k & 0x3f; 152 _ = 0 1 62 42 13, 153 = 0 1 718432 1 const double R hi x . e ffp- R_lo_neg x . a b0e26p-47; 154 double dx =(x - R_hi * k_f) + R_lo_neg * k_f; 155 156 double dx2 =dx * dx; 157 [4] = 0 1 +0 0 1 1, 158 0 1 55555557 54 3, 0 1 55555553 12 4 5 static const double POLY { x p , x p- x . e ffp- x . a f p- }; 159 double p =(POLY[0] + dx * POLY[1]) +160 dx2 * (POLY[2] + dx * POLY[3]); 161 162 double t1_hi=TAB1[idx1][1], t1_lo=TAB1[idx1][0]; 163 double t2_hi=TAB2[idx2][1], t2_lo=TAB2[idx2][0]; 164 double t_lo, t_hi = muldd(t1_hi, t1_lo, 165 t2_hi, t2_lo, &t_lo); 166 double f_hi = t_hi, f_lo = t_lo +t_hi * dx * p; 167 168 const double ERROR = 1.64e-19; 169 if (重新解释.u > 0xc086232bdd7abcd2ull) { 170 重新解释.u = (1 - idxe) << 52; 171 double e; 172 double g_hi = fasttwosum(重新解释.f, f_hi, &e), 173 g_lo =f_lo + e; 174 double upper = g_hi + (g_lo +ERROR), 175 lower = g_hi + (g_lo - ERROR); 176 if (upper != lower) 177 返回 指数函数_accurate(x); 178 f_hi =to_subnormal(lower); 179 } else { 180 double upper = f_hi + (f_lo +ERROR), 181 lower = f_hi + (f_lo - ERROR); 182 if (upper != lower) 183 返回 指数函数_accurate(x); 184 f_hi =ldexp_fast(lower, idxe); 185 } 186 返回 f_hi; 187 }

Ziv（简化版）const F_53_52 = DoubleWord{Float64,Dyadic{52,Big}} const TAB1 = tabulate(i -> Float64x2(F_53_52(sym"2^($i * 2^-6)")), 64) const TAB2 = tabulate(i -> Float64x2(F_53_52 (sym"2^($i * 2^-12)")), 64) @ziv function exp(x::Float64) isnan(x) && return x x == Inf && return Inf x == -Inf && return 0. x <= argmax(x-> exp(x)==0, Float64) && return 0. x >= argmin(x-> exp(x)==Inf, Float64) && returnInf abs(x) <= 0x1p-54 && return 1.0 +x k = round(Int64, x * Float64(sym" 2^12/ log(2)")) k_f, idxe = Float64(k), k >> 12idx1, idx 2 =(k>>6)&Int64(0x3f), k&Int64(0x3f) R = DoubleWord{Dyadic{29,Big},Float64}(sym"log(2) * 2^- 12") dx = x - Float64(R.hi) * k_f - R.lo * k_f POLY = Float64.(gridremez(exp, 4; domain=log(2)/2^13 .* (-1, 1), error=:rel), RoundNearest)[2:end]p = estrin(dx, POLY) t = TAB1[idx1]* TAB2[idx2]f = t *h DoubleWord(1., dx * p) ERROR = 1.64e-19 if (x <= argmax(x -> exp(x) <=maxsubnormal(Float64), Float64)) g = f +ldexp_fast(1.0, -1022 - idxe) lower, upper =g.hi .+(g.lo ±ERROR) upper ==lower && return to_subnormal(lower) return exp_accurate(x) end lower, upper =f.hi .+ (f.lo ± ERROR) upper == lower && return ldexp_fast(lower, idxe) return exp_ accurate(x) end

A B C D E

图 3. CORE-MATH 的指数函数 [57]原始 C语言和Zɪv的简化版本。包含代码块**A**：特殊情况、**B**：范围缩减、**C**：多项式求值、**D**：重构，以及**E**：Ziv 方法。

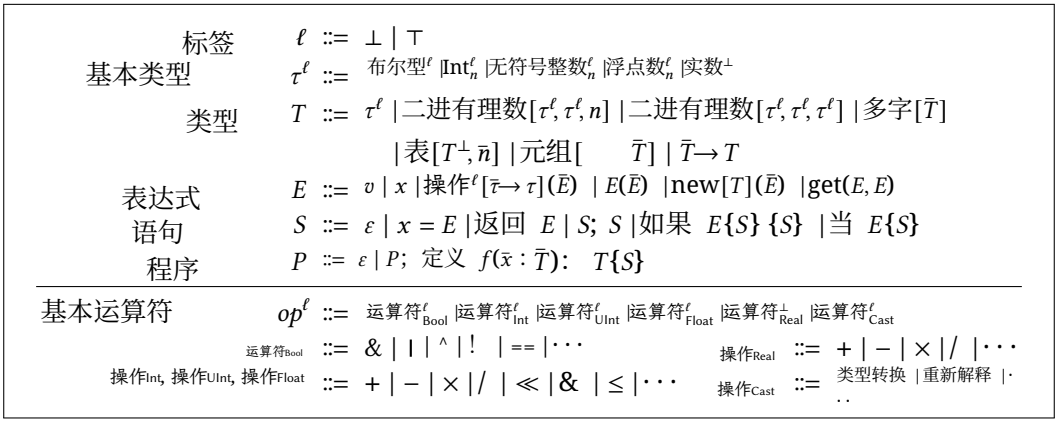


图 4. Ziv 的核心语法。

3 Ziv 语言

现在，我们正式介绍 Zɪv 语言，用于开发正确舍入数学库。Zɪv是一种静态类型命令式语言，通过高级抽象支持扩展精度计算（DP1），并通过类型支持分阶段计算（DP2）。我们刻意将Zɪv的核心设计得尽可能精简，只包含我们认为是简洁实现正确舍入数学库所用技术所必需的构造（§2）。尽管如此，这些构造仍具有足够的表达力和可组合性，使我们能够开发丰富的标准库（§4）以及多种数学函数实现（§7），并以 Zɪv编写。本节将介绍 §3.1）、类型系统（§3.2）和语义（§3.3）的语法。Zɪv。本文中的所有定理均以 Lean 形式化（参见附录 D 和 E）。

3.1 语法

图4 展示了Zɪv的核心语法；其完整定义见附录A.1。下面讨论其中的关键构造。我们用 n 表示一个整数，用 $\bar{a}$ 表示一个有限序列 a_1、……、 $a_k (k \geq 0)$。

标签。为了支持阶段化，Zɪv 区分独立于输入的表达式（它们不依赖于运行时输入）与可能依赖输入的表达式（它们可能依赖于运行时输入）。类型标签 $\ell \in \{\bot,\ \top\}$ 记录了这种区分： $\bot$ 表示不依赖输入，而$\top$ 表示可能依赖输入。类型中只有 $\bot$ 标签的表达式总是在编译时求值（§5.2，定理5.1）；其他表达式可能需要在运行时求值。标签按 $\bot \sqsubseteq \top$排序，我们用 $\ell \sqcup \ell'$ 表示 ℓ 和 ℓ'的最小上界（或并）。

类型。原始类型 τ^ℓ可以是标准标量类型，例如浮点数$_n^\ell$，也可以是实数$^\perp$，表示精确实数。下标 $n \geq 1$ 表示位宽。我们要求实数为 $\bot$标签（即，禁止使用实数$^\top$），这样此类型的表达式就会在编译时求值，详见 §2.1。

类型 T可以是原始类型 τ^ℓ、复合类型或函数类型 $\bar{T} \to T$。复合类型二进有理数$[\tau_1^{\ell_1}, \tau_2^{\ell_2}, n]$表示带有符号类型 $\tau_1^{\ell_1}$、有效数类型 $\tau_2^{\ell_2}$和定点指数 n的定点二进制有理数（§2.1）。类型二进有理数$[\tau_1^{\ell_1}, \tau_2^{\ell_2}, \tau_3^{\ell_3}]$表示带有符号类型 $\tau_1^{\ell_1}$、有效数类型 $\tau_2^{\ell_2}$和指数类型 $\tau_3^{\ell_3}$的浮点二进制有理数。类型多字$[\bar{T}]$表示由类型为 $\bar{T}$的浮点或二进有理数组件构成、尚未求值的和。类型表$[T^\perp, \bar{n}]$表示形状为 $\bar{n}$、元素类型为$T^\perp$的查找表（§2.2），其标签全为 $\bot$。对$T^\perp$的这一限制可确保所有表项都在编译时预先计算。 相比之下，类型元组$[T]$表示元素类型为 T的元组，这些类型不必带有 $\bot$标签。

表达式。 表达式 E 可以是值 v、变量 x、运算符/函数应用，或复合值构造/投影。此处， v是基本值/复合值或函数名 f。

定义（类型上下文、类型判断结果、写集合） $\boxed{\Gamma, \Omega, W}$

$$\Gamma ::= \varepsilon \mid \Gamma,\ x:\ T \qquad \Omega ::= \text{返回} \mid \Gamma \qquad W \subseteq \text{变量}$$

表达式 $\boxed{P; \Gamma \vdash E:\ T}$

$$\frac{\forall i \in [n].\ P; \Gamma \vdash E_i : (\tau_i)^{\ell_i} \qquad \forall i \in [n].\ \ell_i \sqsubseteq \ell \qquad \ell_1 \sqcup \cdots \sqcup \ell_n = \ell'}{P;\ \Gamma \vdash \text{运算符}^{\ell}[(\tau_1,\ \ldots\ldots,\ \tau_n) \to \tau'](E_1,\ \ldots\ldots,\ E_n):\ (\tau')^{\ell'}} \text{T-Op}$$

$$\frac{\begin{array}{c} T' = \text{表}[T^{\perp},\ (m_1,\ \ldots\ldots,\ m_k)] \\ \forall i \in [n].\ P;\ \Gamma \vdash E_i:\ T^{\perp}\ \ n = m_1 \times \cdots \times m_k \end{array}}{P;\ \Gamma \vdash \text{new}[T'](E_1,\ \ldots,\ E_n):\ T'} \text{T-TabNew} \qquad \frac{\begin{array}{c} P;\ \Gamma \vdash E_1:\ \text{表}[T^{\perp}, \bar{n}] \\ P;\ \Gamma \vdash E_2:\ \text{无符号整数}^{\ell}_{k} \end{array}}{P;\ \Gamma \vdash \text{get}(E_1,\ E_2):\ T^{\perp} \sqcup \ell} \text{T-TabGet}$$

语句 $\boxed{P; \Gamma \vdash_T S : (\Omega,\ W)}$

$$\frac{P;\ \Gamma \vdash E:\ T'}{P;\ \Gamma \vdash_T x = E : ((\Gamma, x:\ T'), \{x\})} \text{T-Assn} \qquad \frac{P; \Gamma \vdash E:\ T}{P;\ \Gamma \vdash_T \text{返回}\ E : (\text{返回}\ \varnothing)} \text{T-Ret}$$

$$\frac{P; \Gamma \vdash E : \text{布尔型}^{\ell} \qquad \begin{array}{c} P; \Gamma \vdash_T S_1 : (\Omega_1,\ W_1) \\ P; \Gamma \vdash_T S_2 : (\Omega_2,\ W_2) \end{array} \qquad \begin{array}{c} \text{合并}(\ell,\ (\Omega_1,\ W_1),\ (\Omega_2,\ W_2)) = (\Omega',\ W') \\ \text{返回}(S_1) \vee \text{返回}(S_2) \implies T \sqcup \ell = T \end{array}}{P; \Gamma \vdash_T \text{如果}\ E\ \{S_1\}\ \{S_2\} : (\Omega', W')} \text{T-If}$$

程序 $\boxed{\vdash P : \text{确定}}$

$$\frac{P = (\text{定义}\ f_j(\bar{x_j} : \bar{T_j}) :\ T'_j\{S_j\})_{j \in [n]} \qquad \forall i \in [n].\ P;\ (\bar{x_i} : \bar{T_i}) \vdash_{T'_i} S_i :\ (\text{返回}\ \varnothing)}{\vdash P :\ \text{确定}} \text{T-程序}$$

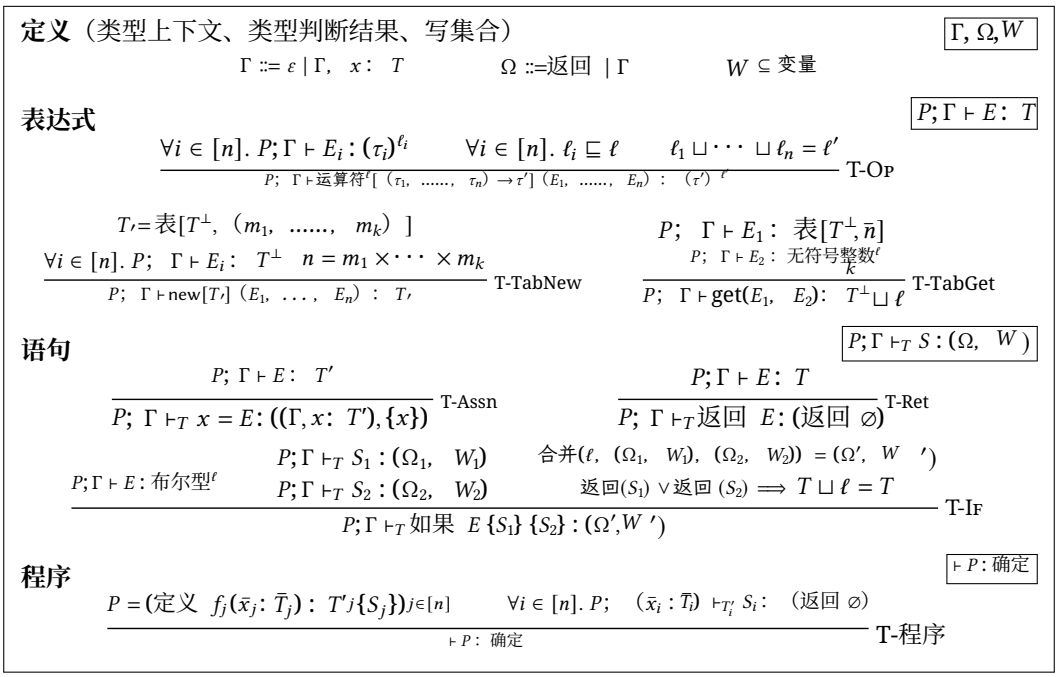


图 5. Ziv 的部分类型规则。

表达式 $op^{\ell}[\bar{\tau} \to \tau'](\bar{E})$ 将内置的基本运算符 op 应用于参数 $\bar{E}$，其中 $\bar{\tau} \to \tau'$ 指定无标签的参数和结果类型，而 ℓ 指定参数类型允许的最高标签。基本运算符包括标准的布尔型、整数或浮点运算符 $op_{\text{Bool}}, \ldots, op_{\text{Float}}$；实数运算符 op_{Real}；以及类型转换运算符 op_{Cast}（例如 cast 和 reint，它们类似于 C语言++的静态_类型转换和重新解释_类型转换）。对于以实数为参数的运算符（例如 op_{Real}），我们要求其标记为 $\perp$，以便在编译时对其应用进行求值。

表达式 $f(\bar{E})$ 用户定义函数 f_j，的名称和参数调用 $\text{new}[T](E)$ 会根据 $\bar{E}$ 创建一个类型为 T 的复合值，而 $\text{get}(E, E')$ 会选取复合值中的第 E' 个分量 E。

语句与程序。 语句 S 可以是空操作、赋值、返回、顺序组合、条件语句或循环。程序 P 是由以下定义组成的列表：用户定义函数，每个函数的形式如下：$\text{def}\ f(\bar{x} : \bar{T}) : T' S\}$。用户定义函数可以是实现正确舍入函数（例如 指数函数）的主函数、提供通用运算（例如二进有理数加法）的标准库函数（§4），或仅供少数实现使用的辅助函数（例如用于生成 指数函数表）。

用户定义函数和函数调用使 Ziv 具有高度的可扩展性和可组合性：用户可以使用核心构造和现有函数定义新的运算，再将生成的函数组合起来，构建更高层次的抽象。例如，标准库（§4）中的复合值运算（如二进制/多字运算（§2.1））是通过用户定义函数实现的，而非作为核心语言中的基本运算符提供。

3.2 类型系统

图 5 展示了 Ziv 类型系统的一个片段；其完整定义见附录 A.2。该类型系统跟踪运行时输入上的数据依赖和控制依赖，以指导阶段化，并采用流敏感类型检查，允许变量在重新赋值时改变类型。我们将其写作 $[n] := \{1、2、\ldots\ldots、n\}$。

表达式。 表达式 E的类型判断为 P； $\Gamma \vdash E : T$，其中 Γ是一个将变量映射到类型的类型上下文。该判断表示，在程序 P和上下文 Γ下， E的类型为 T。

类型规则限制了某些表达式的输入依赖。例如，规则 T-Op要求每个参数 E_i 的类型标签至多为运算符 op的标签。由于带有 实数 参数的运算符必须为 $\bot$ 标记（§3.1），它们的应用都会在编译时求值。类似地，规则 T-TabNew 要求每个表元素 E_i的类型为 $T^{\bot}$，其标签全都为 $\bot$。这确保所有表都独立于运行时输入，并在编译时预先计算。

类型规则还会可靠地追踪对运行时输入的数据依赖。例如，规则 T-Op通过取并 $\ell_1 \sqcup \cdots \sqcup \ell_n$，将参数类型标签 ℓ_i传播到结果类型。类似地，规则 T-TabGet 通过取并 $T^{\bot} \sqcup \ell$，将索引类型标签 ℓ传播到结果类型，其中$T' \sqcup \ell'$表示将 T，中的每个标签与 ℓ'合并后得到的类型。

语句。语句的类型判断 S是 P； $\Gamma \vdash_T S : (\Omega, W)$ 其中 Ω是 ret 或类型上下文 Γ'，而W是变量集合。该判断表示，在程序 P和上下文Γ下， S返回的每个值的类型都是 T。此外，它还表示：(i) 如果$(\Omega, W) = (\text{ret}\ \varnothing)$，那么 S会在每条终止路径上返回；(ii) 如果$(\Omega, W) = (\Gamma', W)$ 那么 S可能终止而不返回，此时 Γ'给出执行 S后的变量类型，而W记录 S可能赋值的变量。例如，针对 $S \equiv (\text{return}\ E)$的规则 T-Ret 要求 E具有返回类型 T，并得出$(\text{ret}\ \varnothing)$；而针对$S \equiv (x = E)$的规则 T-Assn 得出$(\Gamma', \{x\})$，并在$W$中记录 x。后一条规则还展示了流敏感类型：Γ' 将 x映射到 E的类型，因此变量类型可以在重新赋值时发生变化。

类型规则能够可靠地追踪对运行时输入的控制依赖。例如，规则 T-If用于 $S \equiv (\text{if}\ E\ \{S_1\}\ \{S_2\})$ 通过两个步骤实现这一点。首先，该规则追踪变量值可能如何依赖于条件 E，这通过函数$merge$ （其形式化定义见附录A.2）实现。直观而言，此函数使用的类型标签 ℓ，按如下方式合并分支类型输出（Ω_i、 W_i）($i = 1$、2)：

- 它将 Γ，计算为各结果 Ω_i中类型上下文 Γ_i 的“交集”：仅保留所有 Γ_i中无标签类型相同的变量，并合并相应的类型标签。
- 它将 W，计算为相应变量集合 W_i的并集。
- 它通过将 ℓ并入 $\text{dom}(\Gamma') \cap W'$中每个变量的类型，来更新 Γ'。

其次，该规则追踪返回值可能如何依赖于条件 E。具体而言，如果任一分支包含返回语句（即$ret(S_1) \vee ret(S_2)$），则要求返回类型T中的每个标签都至少与条件标签 ℓ一样高；其中$ret(S')$表示 S' 是否包含返回语句。

对于 while 循环，规则 T-While (附录 A.2) 施加了两项限制：(i) 循环条件的类型必须为 布尔型$^{\bot}$，因此不受运行时输入影响；(ii) 循环体不得包含返回语句，从而排除循环中的提前返回。这些限制以极低的实际代价简化了 Zɪv 编译器（§5），因为现有数学库很少使用依赖运行时输入或包含返回语句的循环。

程序。 对于程序 P，其判断式为 $\vdash P : \text{ok}$，表示 P是良类型的。规则 T-Prog要求每个函数体在终止时都返回一个符合声明的返回类型的值。

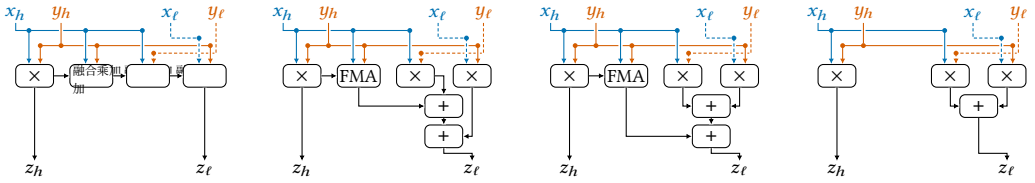
3.3 语义与类型保持

我们使用判断式 Zɪv 的大步操作语义 $P \vdash f(\bar{v}) \Downarrow v'$，它表示函数调用 $f(\bar{v})$ 在程序 P中求值为一个值 v'。规则见附录 A.3。

我们证明了 Ziv 满足类型保持：在类型良好的程序中，如果一个带有预期类型参数的函数调用求值为一个值，那么该值具有声明的返回类型。

定理 3.1（类型保持）。设 P是一个类型正确的Zɪv 程序。假设函数 f的类型为$(T_1, \ldots, T_n) \to T'$， 属于 P， 即 $P; \varepsilon \vdash f : (T_1, \ldots, T_n) \to T'$。那么，对于所有值 $v_1, \ldots, v_n, v'$，

$$(\forall i \in [n].\ P; \varepsilon \vdash v_i : T_i) \quad \wedge \quad P \vdash f(v_1, \ldots, v_n) \Downarrow v' \quad \Longrightarrow \quad P; \varepsilon \vdash v' : T'.$$



(a) $x \times y$ 在*LLVMLibc* (b) $x \times y$ 在 *CoreMathHyp* (c) $x \times y$ 在 *CoreMathExp* (d) $x \times_{hi} y$ 在 *CoreMathExp*

图 6. *ZivStd* 中选定的双字乘法变体。FMA 表示融合乘加 [26]。

4 Ziv 标准库

§3 中介绍的Ziv 核心小语言为更高级的抽象提供了基础（DP1），而这些抽象在库层面表达更为合适。本节将讨论Ziv标准库ZivStd，并使用标准语法糖提高可读性。表*1*列出了ZivStd的关键函数，包括算术运算（§4.1）、实现步骤抽象（§4.2）以及与浮点数相关的函数（§4.3）。

4.1 算术

实值函数。 *ZivStd* 提供针对实数值的函数，以及将其舍入到有限精度的构造函数。例如，*makeMultiWord* 可从实数构造多字$[T]$：

```
def makeMultiWord[T_1,...,T_n](x: 实数^⊥) : 多字[T_1,...,T_n]^⊥{  w_i =
类型转换[T_i](x − ∑_{j=1}^{i−1} 类型转换[实数](w_j))(i = 1,...,n); 返回 new[
多字[T_1,...,T_n]](w_1,...,w_n); }
```

扩展精度算术。 对扩展精度类型（§2.1）进行算术运算时，可以使用具有不同属性的多种运算符变体。为了让 Ziv 保持精简，这些运算符属于 *ZivStd*。

对 多字$[T]$ 的算术运算通过无误差变换（§2.1）实现，包括*2sum*、*fast2sum*和 *2mul* [4, 14, 45]。图 6 展示了 *ZivStd* 的双字乘法的四种变体。LLVM-libc [42] 使用三个串联的融合乘加运算（图6a），而 CORE-MATH 的 [57] 双曲函数会展开最后两个融合乘加运算（图6b）。CORE-MATH 的指数函数既采用不同的运算顺序（图6c），也采用省略高位部分残差的版本（图6d）。

浮点和定点二进有理数算术也支持多种运算符变体，这些变体通过整数算术实现。尤其是，由于 128 位精度的浮点二进有理数被广泛使用，*ZivStd* 为二进有理数[布尔型、无符号整数$_{128}$,整数$_{64}$]提供了专门的实现。

4.2 实现步骤抽象

除基本算术外，*ZivStd* 还提供用于范围缩减、多项式逼近和多项式求值的高级抽象，以支持 §2.2 中的实现步骤。

范围缩减。 *ZivStd* 支持 Cody-Waite [12] 和 Payne-Hanek [52] 范围缩减算法。给定输入 x和缩减常数 C，这两种算法都会计算一个整数 k和一个缩减后的自变量 $r \approx x - kC$，但适用于不同的输入范围。下面给出了使用三字算术和 *makeMultiWord* 函数实现 Cody-Waite 范围缩减的示例，该函数见 §4.1。

```
def codywaite(x: Float64, C: 实数^⊥) : 元组[多字[浮点数_64, 浮点数_64],整数_64]{  c =
makeMultiWord [二进有理数[布尔型, 无符号整数_32, 整数_64], 二进有理数[布尔型, 无符号整数_32, 整数
_64], 浮点数_64](−C);k_i = 类型转换[整数_64](x × 类型转换[浮点数_64](1/C)); k_f = 类型转换[浮点数_64](k_i); 返
回 (fma(k_f, c, x)  k_i); }
```

要实现基于查表的范围缩减（§2.2），可以使用 [60, 61], *tabulate*(f, $\bar{n}$)，它会在编译时将函数映射到表的每个索引上，从而创建一个 表$[T^{\perp}, \bar{n}]$。 f

表 1。*ZivStd* 的代表性函数。

	函数	Type	描述
Arithmetic (§4.1)	**实值函数** 指数函数、对数、三角、……	$(\text{实数}^n)^\perp \to (\text{实数}^m)^\perp$	实函数
	makeMultiWord $[\ \bar{T}\ ]$	实数$^\perp \to$ 多字$[\ \ \bar{T}\ ]^\perp$	多词构造
	makeDyadic $[\tau, \tau', \alpha]$	实数$^\perp \to$ 二进有理数$[\tau, \tau', \alpha]^\perp$	二进有理数构造（$\alpha = \tau''$ 浮点， n 定点）
	无误差变换		
	2sum、*fast2sum*	浮点数$_n{}^2 \to$ 浮点数$_n{}^2$	舍入和及其残差 [14, 33, 45, 56]
	2mul	浮点数$_n{}^2 \to$ 浮点数$_n{}^2$	舍入积及其残差 [32, 49]
	2fma，*fast2fma*	浮点数$_n{}^3 \to$ 浮点数$_n{}^2$	舍入融合乘加及其残差 [5, 6, 29, 34]
	扩展精度算术 （T =多字$[\ \ \bar{T}\ ]$，二进有理数$[\tau, \tau', \alpha]$）		
	$+, -, \times, /$	$T^2 \to T$	扩展精度算术，默认
	$+_{hi}, \times_{hi}, +_{nm}, \times_{nm}, \ldots$	$T^2 \to T$	扩展精度算术，变体
	inv	$T^1 \to T$	扩展精度倒数
	fma	$T^3 \to T$	扩展精度FMA
	ldexp	$(T, \text{整数}_n) \to T$	按二的幂缩放
	frexp	$T \to$ 元组$[T, \text{Int}_n]$	分解为小数有效数和指数
Impl. Step Abstractions (§4.2)	**表操作**		
	制表	$((\text{无符号整数}_{64}{}^{\lvert\bar{n}\rvert} \to T),\ \text{无符号整数表}[T^\perp, \bar{n}]$	通过对索引映射构造表（形状 $\bar{n}$）
	getelem	$(\text{表}[T^\perp, \bar{n}],\ \text{无符号整数}_{64}{}^{\lvert\bar{n}\rvert}) \to T$	多维表访问
	范围缩减		
	Cody-Waite	（浮点数$_n$，实数$^\perp$） $\to$ 元组$[T, \text{Int}_m]$	Cody-Waite 范围缩减 [12]
	paynehanek	（浮点数$_n$，实数$^\perp$）$\to$ 元组$[T, \text{整数}_m]$	Payne-Hanek 范围约简 [52]
	多项式逼近与求值		
	taylor	$((\text{实数} \to \text{实数}),\ \text{选项...})^\perp \to$ 元组$[\overline{\text{实数}}]^\perp$	泰勒近似
	remez，*gridremez*	$((\text{实数} \to \text{实数}),\ \text{选项...})^\perp \to$ 元组$[\overline{\text{实数}}]^\perp$	在区间和固定网格上进行极小极大逼近 [11, 54]
	fpminimax	$((\text{实数} \to \text{实数}),\ \text{选项...})^\perp \to$ 元组$[\overline{\text{浮点数}_n}]^\perp$	极小极大逼近，浮点系数 [8, 11]
	霍纳、*Estrin*	（T、元组$[\bar{T}]^\perp$） $\to T$	多项式求值、霍纳方法和埃斯特林方法 [16, 33]
Float-Related (§4.3)	**表示相关函数**		
	分解 是否为非规格化数，是否为无穷大……	浮点数$_n \to$ 元组[布尔型，无符号整数$_n$，整数$_n$]	分解为符号、有效数和指数
		浮点数$_n \to$ 布尔型	浮点值分类
	最小指数，偏置 …	$\text{Int}_{64} \to \text{Int}_n$	IEEE-754 相关常数的计算
	下一个浮点数，前一个浮点数	浮点数$_n \to$ 浮点数$_n$	下一个和前一个浮点值
	最小值参数，最大值参数	（（浮点数$_n \to$ 布尔型），浮点数$_n{}^2$）$\to$ 浮点数$_n$	单调谓词优化
	测试		
	ULP 误差	（浮点数$_n$，实数$^\perp$）$\to$ 浮点数$_{64}$	误差，以参考值末位为单位 [47]
	随机测试	（（浮点数$_n \to$ 浮点数$_n$），（实数$\to$ 实数）$^\perp$）$\to$ 浮点数$_{64}$	对浮点函数进行随机 ulp 误差测试

多项式逼近与求值。 支持使用泰勒多项式（*taylor*）进行多项式逼近，也支持使用里米兹算法在区间（*remez*）或固定网格（*gridremez*）上逼近极小极大多项式，或使用浮点系数（*fpminimax*） [8,11]。逼近所得的多项式可以通过霍纳 和 *Estrin*方案进行求值，这些方案利用 Ziv 的分阶段机制将求值展开为直线代码。

4.3 浮点相关函数

除了上述扩展精度数值抽象外，*ZivStd* 还提供了用于检查和操作机器浮点表示的底层实用工具。

表示相关函数。与浮点值 [26] 的位表示相关的实用工具包括常量计算（例如，偏置、符号掩码）、分类（例如，是否为非规格化数、是否为无穷大）、字段提取（例如，分解）和值操作（例如，前一个浮点数、下一个浮点数）。Ziv 的阶段化机制允许这些位级函数接受格式参数，并在编译时进行特化。

优化函数。优化函数 最小值参数和最大值参数 分别在编译时搜索满足单调谓词的最小和最大浮点值。在内部，这两个函数都会对浮点位模式执行二分搜索。

基本类型	τ	::=	布尔型 |整数$_n$ |无符号整数$_n$ |浮点数$_n$
基本运算符	op	::=	运算符$_{\text{Bool}}$ |运算符$_{\text{Int}}$ |运算符$_{\text{UInt}}$ |运算符$_{\text{Float}}$ |运算符$_{\text{Cast}}$
表达式	E	::=	$v \mid x \mid op[\bar{\tau}\to\tau](\bar{x}) \mid \text{get}(x,x)$
语句	S	::=	$\varepsilon \mid x\colon\ \tau = E \mid x\colon\ \tau[n] = \text{new}(\bar{v}) \mid S;\ S \mid (\bar{x}\colon\ \bar{\tau})\ = \text{if}\ x\{S;\ \bar{x}\}\ \{S;\ \bar{x}\}$
程序	P	::=	定义 $f(\bar{x}\colon\ \bar{\tau})\colon\ \tau\{S;$ 返回 $x\}$

图 7. ZivIR 的核心语法。

5 Ziv 编译器

现在，我们正式介绍Zɪv 编译器，它将以Zɪv (§3) 编写的程序与*ZɪvSᴛD* (§4) 一起编译为C语言$_0$, 的一个简化片段。该编译器通过DP2支持分阶段计算，具体方式是执行追踪步骤，将部分求值与符号追踪结合起来：前者在输入值已知时，于编译时执行计算；后者则将剩余计算记录为中间代码，以便在运行时执行。对于良类型程序，编译器确保带标签集类型（§3）的表达式会在编译时求值。

编译器分三个步骤进行：规范化（§5.1）、追踪（§5.2）和代码生成（§5.3）。首先，规范化通过将 Ziv 翻译为 ZivNorm 来消除提前返回语句。ZivNorm 是 Ziv 的一个子集，其中每个函数都只在函数体末尾有一个返回语句。其次，追踪通过部分求值和符号追踪，将 ZivNorm 翻译为中间语言 ZivIR。第三，代码生成将 ZivIR 通过语法制导翻译转换为 C_0。在 §5.4 中，我们证明端到端编译能够保留良类型程序的语义。为了区分这些语言，我们使用黄色背景表示 Ziv 和 ZivNorm，蓝色表示 ZivIR，白色表示 C_0。

5.1 规范化（Ziv → ZivNorm）

规范化步骤会重写每个函数，使其只在函数体末尾有一个 返回语句。这样可以简化追踪（§5.2），因为此后所有分支都可以用相同的方式进行追踪，无论它们原本是提前返回还是继续执行。我们定义了一个从 Ziv $_{\text{norm}}$到 ZivNorm的语法制导翻译 T_{norm}，并使用辅助函数 $\mathcal{T}_{\text{norm}}(S,\ S')$ 来规范化语句 S，其中 y 是结果变量，而 S' 是在 S 未返回时[illegible]行的规范化语句。选取的规则如下

$$\mathcal{T}_{\text{norm}}\big(\text{def } f(\bar{x}:\bar{T}):T'\ \{S\}\big) \coloneqq \text{def } f(\bar{x}:\bar{T}):T'\ \{\mathcal{T}^{y}_{\text{norm}}(S,\varepsilon);\text{return } y\}\quad \text{for fresh } y,$$

$$\mathcal{T}^{y}_{\text{norm}}(\text{return } E, S') \coloneqq y = E,\qquad \mathcal{T}^{y}_{\text{norm}}(S_1;S_2, S') \coloneqq \mathcal{T}^{y}_{\text{norm}}(S_1, \mathcal{T}^{y}_{\text{norm}}(S_2, S')).$$

函数情形从 $S' = \varepsilon$ 开始，并[illegible]返回 y。返回情形[illegible]值存储在 y 并丢弃 S'；顺序执行情形则先[illegible]规范化，再[illegible]理[illegible]。完整翻译见附录 C.1，类型保持定理和语义保持定理见附录E.1。

5.2 追踪（ZivNorm → ZivIR）

中间语言。 ZɪvIR是一种静态类型命令式语言，用于表达留待运行时执行的计算。图7 展示了核心语法。它采用静态单赋值形式（即每个变量只有一个定义），并要求表达式只能出现在赋值语句中。与 Zɪv不同，ZɪvIR 仅支持布尔型、整数和浮点数类型（及其数组类型），因为其他类型上的计算会在编译时求值，或在追踪期间降级为标量和数组运算。它不支持函数调用和 while 循环，因为这些内容会在追踪期间内联和展开。它的条件语句会将分支结果赋给新变量，从而简化代码生成（§5.3）。ZɪvIR的语义、类型系统和类型保持定理见附录 B和 D.2。

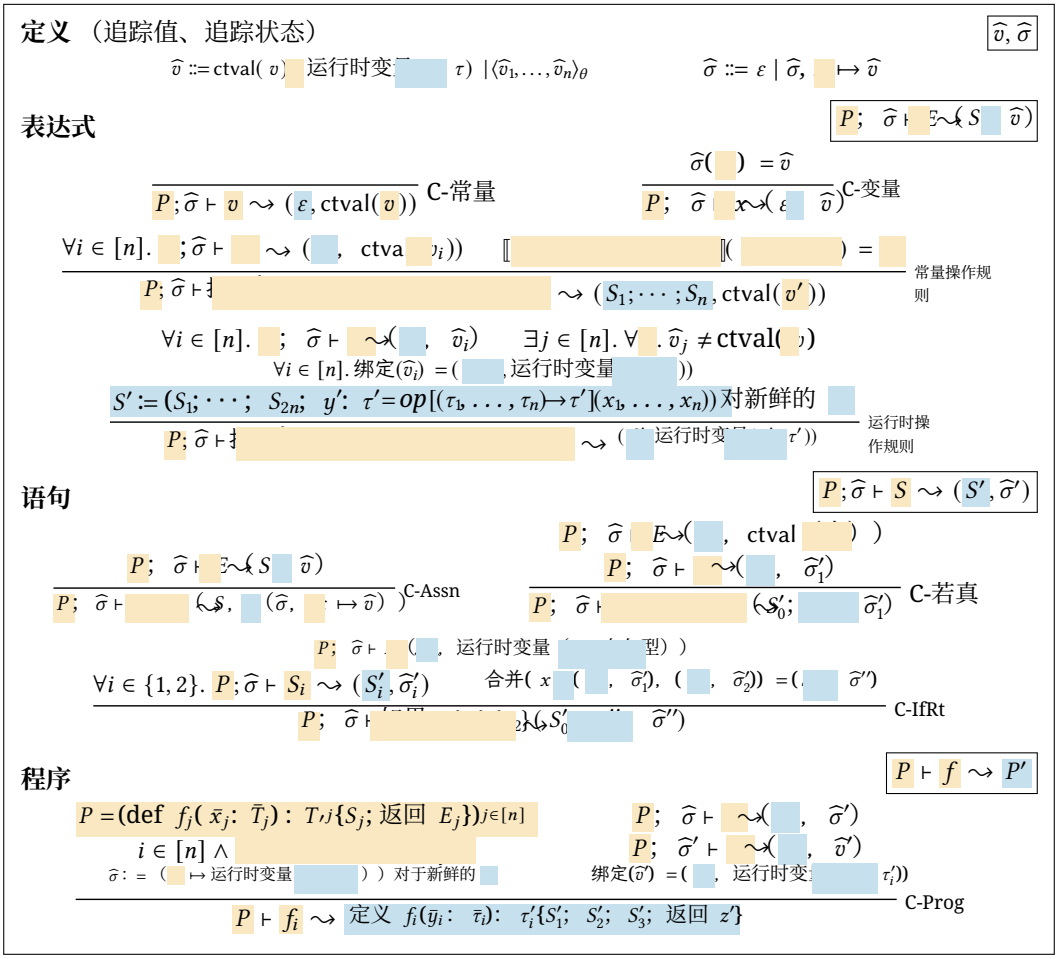


图 8. 部分追踪规则（ZivNorm → ZivIR）。

跟踪概述。跟踪步骤在非标准语义下运行 ZivNorm 程序，该语义会 (i) 求值输入在编译时已知的计算，以及 (ii) 为其余计算生成 ZivIR代码。我们使用基于跟踪值和状态的操作语义来形式化这一过程。跟踪值 $\widehat{v}$可以是编译时值 ctval(v)、运行时变量 rtvar(x: τ)，也可以是复合值 $\widehat{v}_1, \ldots, \widehat{v}_{n\theta}$。其中，ctval($v$)表示编译时已知的 ZivNorm 值 v，rtvar(x: τ)表示一个 ZivIR 变量 x，它将在运行时存储类型为 τ的值，而 $\widehat{v}_1, \ldots, \widehat{v}_{n\theta}$表示一个由 $\widehat{v}_1, \ldots, \widehat{v}_n$组成、类型为 θ的复合值。跟踪状态$\widehat{\sigma}$将 ZivNorm 变量映射到跟踪值。随着跟踪的进行，语义会更新此状态并生成 ZivIR代码。图 8 展示了部分规则；完整定义见附录 C.2。

追踪表达式。 表达式 E的追踪判断为 $P; \widehat{\sigma} \vdash E(S, \widehat{v}$ 它表示，在程序 P和状态 $\widehat{\sigma}$下追踪 E会产生一条 ZivIR 语句 S和一个追踪值$\widehat{v}$。例如，运算符规则会根据参数的追踪值，在部分求值和符号追踪之间作出选择。如果所有参数都追踪为编译时值，规则 常量操作规则就会对运算符求值并返回结果 v'作为编译时值，其中 $[\![\cdot]\!]$ 表示运算符求值。否则，规则 运行时操作规则 会生成用于运行时运算符求值的 ZivIR 代码，并返回

一个新的 ZivIR 变量 [illegible] 来保存结果。在该规则中，绑定($\widehat{v}'$) 返回 (S'，运行[illegible]变量([illegible] τ))，其中 [illegible] 将在 $\widehat{v}'$ 运行后存储由 S' 表示的值；其定义[illegible]见附录 C.2。

语句追踪。语句 S 的追踪判断为 P；$\widehat{\sigma} \vdash S(S', \widehat{\sigma}')$。[illegible]表示，在程序 P 和状态 $\widehat{\sigma}$ 下追踪 S，会[illegible]生成一个 ZivIR [illegible]句 S' 和更新后的状态 $\widehat{\sigma}'$。例如，条件[illegible]语句的规则会根据条件 E 的追踪值，在部分求值和符号追踪之间进行选择。如果 E 追踪为 ctval(true)，规则 C-若真只追踪真分支；ctval(false) 的情况与此类似。如果 E 追踪为 rtvar(x: 布尔型)，规则 C-IfRt 会从同一初始状态追踪两个分支，并将生成的 ZivIR代码和追踪状态合并为单个 ZivIR 条件 S'' 以及供后续追踪[illegible]用的合并状态 $\widehat{\sigma}''$。随后，该规则返回 S_0'；S'' 和 $\widehat{\sigma}''$，其中 [illegible] 是追[illegible]件 E 时生成的 ZivIR代码。定义 merge(·) 时，需要谨慎处理跨分支合并编译时值和运行时值，并使用 ZivIR 原语变量表示复合值；其完整定义见附录 C.2。

程序追踪。 函数的追踪判断 [illegible] 为 [illegible] $\vdash$ [illegible]。它表示，对程序 P 的入口函数 [illegible] 进行追踪，会生成一个 ZivIR 程序 P'。规则 [illegible] Prog 从一个初始状态开始追[illegible]。该状态将 ZivNorm 参数映射到 [illegible] 中对应的 ZivIR 参数 [illegible]。

分阶段保证与可靠性。 我们证明，追踪遵循 Ziv 的输入依赖标签：对具有 $\perp$ 标签的原始类型表达式进行追踪，会产生一个编译时值。这确保了 Ziv 类型系统标记为独立于输入的计算会在编译时求值。

定理 5.1（$\perp$ = ctval）。 设 [illegible] 是一个类型正确的 ZivNorm 程序。假设表达式 E 在 [illegible] 下文 Γ下具有原[illegible]类型 $\tau^{\perp}$，即 [illegible]；[illegible] 对于所有与 Γ“兼容”的追踪状态 $\widehat{\sigma}$，[illegible]

$$P; \widehat{\sigma} \vdash E \rightsquigarrow (S, \widehat{v}) \quad \Longrightarrow \quad \exists v.\, \widehat{v} = \mathrm{ctval}(v) \wedge P; \varepsilon \vdash v : \tau^{\perp}.$$

该定理的一般版本（包括“兼容性”的形式化定义），以及追踪的类型保持和语义保持定理，见附录E.2。

5.3 代码生成（ZivIR $\to$ C语言$_0$）

代码生成步骤将 ZivIR 程序翻译为 C语言$_0$, 的简化片段。附录 C语言$_0$ 给出了其语法、语义、类型系统和类型保持定理，详见附录 B 和D.3。由于追踪步骤已经完成了编译时计算并降低了高级构造，因此这一步是一个简单的语法制导翻译，我们将其记为 $\mathcal{T}_{\mathrm{emit}}$。例如：

$\mathrm{T}_{\mathrm{emit}}$([illegible]) := $\bar{\tau}\ \bar{x}$；若 (y) {$\mathrm{T}_{\mathrm{emit}}$([illegible]) $\bar{x} = \bar{z}_1$；} 否则 {$\mathrm{T}_{\mathrm{emit}}$([illegible]) $\bar{x} = \bar{z}_2$；}

该规则在条件语句之前声明 $\bar{x}$，并在每个分支中为其赋值。完整的翻译见附录 C.3，其类型保持定理和语义保持定理见附录 E.3。

5.4 编译器可靠性

我们为从 Ziv 到 C语言$_0$的端到端编译建立了一个可靠性定理。该定理指出：(i) 成功编译函数 [illegible] 会在类型良好的 Ziv 程序 P 中生成一个类型良好的 C语言$_0$程序 R；(ii) 对于每个类型良好的输入，编译后的函数 f 在 R 中求值得到某个值，当且仅当原函数 [illegible] 求值得到相同的值。证明结合了三个编译步骤的类型保持定理和语义保持定理（见附录 E.4）。

定理 5.2（编译器可靠性）。设 P 是一个[illegible]型良好的 Ziv 程序，其中包含一个类型为 $(T_1, \ldots, T_n) \to T'$ 的 [illegible] 译 [illegible] 生成了 R，即 $T_{\mathrm{norm}}(P) \vdash$ [illegible] 且 T_{emit} [illegible] $) = R$。那么，R 是一个类型良好的 C语言$_0$程序，并且对于所有值 $v_1, \ldots, v_n, v'$，其中每个 v_i 的类型均为 [illegible]，

$$P \vdash f(v_1, \ldots, v_n) \Downarrow v' \quad \Longleftrightarrow \quad R \vdash f(v_1, \ldots, v_n) \Downarrow v'.$$

所有定理的 Lean 形式化证明均收录于补充材料中（参见附录 D和 E）。

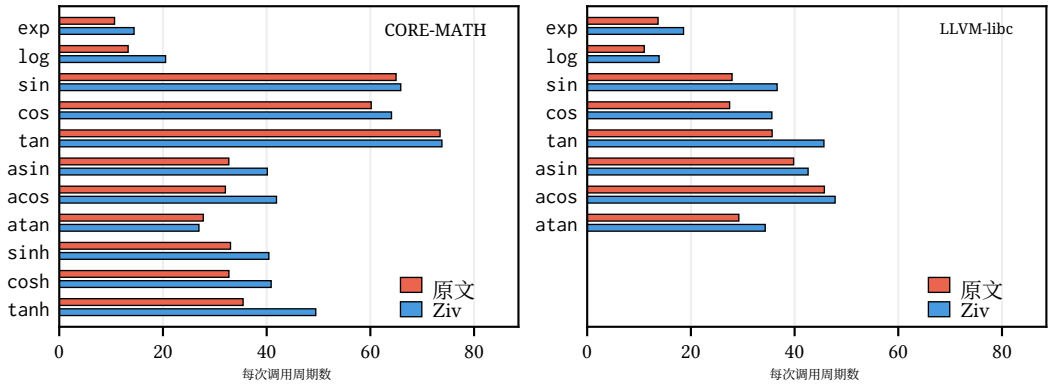


图 9. 原始 CORE-MATH 和 LLVM-libc 函数及其 Ziv 重实现的每次调用周期数。

6 实现

我们利用 Julia 的多重分派、类型系统和元编程功能，将 Zɪv实现为 Julia 中的嵌入式语言 [3], 。下面我们将讨论 Zɪv 及其编译器的实现。

语言。 Ziv 的基本运算符和用户定义函数都实现为 Julia 方法，并且可以重载。Julia 的 多重分派 会根据所有参数的类型选择要调用的方法。因此，整数加法和浮点加法可以共用语法 `+`，正如 §3.1 中所假设的那样。 *ZɪvSᴛD* 中的用户定义函数使用相同机制，将 `+` 扩展到两个多字值之间，或多字值与浮点值之间的加法。

大部分类型检查都交由 Julia 的类型系统处理。Ziv 的原始类型以原生 Julia 类型表示，复合类型则以 Julia 结构体表示。虽然我们在讨论 Ziv 时省略了类型推断，但它是通过 Julia 的类型系统实现的，其中也包括对标签的推断。

编译器。 编译器的规范化步骤（§5.1）需要进行语法转换，这些转换通过 Julia 的元编程实现。`@ziv` 宏会应用这些转换。

在追踪过程中（§5.2），同一个函数通过多重分派同时支持部分求值和符号追踪。静态值具有 Julia 类型 `T`，表示一种 ZɪvIR 原始类型（例如`Float64`）；而动态值具有参数化类型 `Tracer{T}`。多重分派允许同一个函数针对 `T` 和 `Tracer{T}`分别定义重载。例如，当 `x` 的类型为 `T`时，函数调用 `f(x)` 会进行部分求值；而当 `x` 的类型为 `Tracer{T}`时，则会进行符号追踪。

7 评估

我们评估 Zɪv 是否能让正确舍入数学库更易于编写、理解和修改，从而应对 §1.1 中指出的挑战。我们考察四个研究问题：

RQ1。 Zɪv 能否表达现有实现，同时实现有竞争力的性能？**RQ2**。 Zɪv 能否更简洁地表达现有实现？**RQ3**。 Zɪv 是否适合编写新的实现？**RQ4**。 Zɪv 是否适合探索实现变体？

我们通过三个案例研究来探讨这些问题：重新实现现有数学库（§7.1）、编写新的实现（§7.2），以及探索实现变体（§7.3）。

实验设置。我们使用嵌入在 Julia 中的 Zɪv (§6) 实现来编译Zɪv程序为 C 代码。我们使用 Clang 21.1 和以下标志编译 Zɪv生成的 C 代码以及基准 C/C++ 代码：`-O3 -march=native`。所有实验均在 AMD EPYCTM 9575F 处理器上运行。

表 2. CORE-MATH 和 LLVM-libc 原始实现及其 Ziv 重实现中的源代码行数（SLOC）。为保证公平，我们排除数值支持代码；各个函数的 SLOC 仅统计函数专用代码，而函数族内可共享的代码仅计入其 共享行。

		CORE-MATH			LLVM-libc		
函数族	函数	原文	Ziv	原文：Ziv	原文	Ziv	原文：Ziv
指数函数	exp	210	136	1.54 : 1	466	123	3.79 : 1
对数函数	log	1,051	155	6.78 : 1	941	125	7.53 : 1
三角函数	sin	140	12	11.67 : 1	112	69	1.62 : 1
	cos	138	12	11.50 : 1	111	67	1.66 : 1
	tan	195	109	1.79 : 1	138	87	1.59 : 1
	共享	1,012	565	1.79 : 1	588	119	4.94 : 1
	小计	1,485	698	2.13 : 1	949	342	2.77 : 1
反三角函数	asin	133	76	1.75 : 1	89	59	1.51 : 1
	acos	108	63	1.71 : 1	99	79	1.25 : 1
	atan	192	133	1.44 : 1	163	58	2.81 : 1
	共享	91	95	0.96 : 1	406	82	4.95 : 1
	小计	524	367	1.43 : 1	757	278	2.72 : 1
双曲函数	sinh	230	115	2.00 : 1	未在以下版本中实现 LLVM-libc 提交 efe31d6d。		
	cosh	193	86	2.24 : 1			
	tanh	181	116	1.56 : 1			
	共享	69	147	0.47 : 1			
	小计	673	464	1.45 : 1			
	总计	**3,943**	**1,820**	**2.17: 1**	**3,113**	**868**	**3.59: 1**

我们从准确性和性能两方面，将 Ziv 实现与基准库进行比较。准确性通过精确值与计算值之间的误差来衡量，以 最后一位单位 (末位单位) [47]表示。一般来说，以最后一位单位表示的误差，计算的是精确值与计算值之间相隔的浮点步数。在舍入到最近模式下，正确舍入意味着最大误差为 0.5 个最后一位单位；当精确结果恰好位于两个相邻浮点数的中间时，误差达到该最大值。性能以倒数吞吐量衡量，单位为每次函数调用所需的 CPU 周期数。

我们将性能和准确性测试集中在 64 位浮点输入上，因为对更大格式的支持仍处于起步阶段 [9, 42]。准确性测试使用 10^9 位随机输入，以及可用的难以正确舍入的输入 [38, 39]。性能测试报告 10 次试验中每次调用周期数的最小值。每次试验均从函数特定区间 [57] 中随机抽取 200,000 个输入进行。

7.1 使用 Ziv（RQ1、RQ2）重新实现现有库

我们在 Ziv 中重新实现现有的正确舍入数学库，以评估我们系统的表达能力和性能（RQ1），以及简洁性（RQ2）。我们选择 CORE-MATH [57] 和 LLVM-libc [42] 作为重新实现的基准库，因为它们代表了声称对 64 位输入提供正确舍入的数学库的最新水平。我们重新实现了 CORE-MATH 提交 `07cf01e1`中的 11 个函数，以及 LLVM-libc 提交 `efe31d6d`中的 8 个函数。

这些Ziv 重实现与 CORE-MATH 和 LLVM-libc 在所有测试输入上的结果逐位一致，证明了Ziv能够表达正确舍入函数（RQ1）。在 Ziv中重实现基线实现还揭示了数值错误，据我们所知，这些错误此前未曾见诸文档。LLVM-libc 的 余弦 未能正确舍入，其浮点二进制加法可能返回错误的指数。Julia 的 `ldexp`也未能正确舍入。此外，我们还还原了 CORE-MATH 中一些未记录多项式和表的数学定义。

我们在图 Ziv 的重新实现与原始基线实现的性能进行了比较，结果如图 9所示，证明了 Ziv 能够实现有竞争力的性能（RQ1）。相较于 CORE-MATH，Ziv的平均 0.83× 性能（几何平均值）范围为 0.65× 至 1.03×。相较于

表3。Ziv 中实现的新函数及其基线实现的观测到的最大 ulp 误差和每次调用周期数。括号中的 MPFR 工作精度是在测试中发现的最低充分精度。C语言基线实现重新实现了 PyTorch 的数值算法。†括号中报告的是有限 ulp 误差的最大值。

函数	目标精度	实现	最大误差（最后一位单位）	每次调用周期数
二重对数	≤ 0.5最后一位单位	MPFR（53位）	0.5	44,197.6
		Ziv（手写）	0.5	1,209.8
		Ziv（已搜索）	0.5	768.0
	> 0.5最后一位单位	GSL	∞（3,060,232.6）†	277.7
		Ziv（已搜索）	2,364,539.2	45.0
GELU	≤ 0.5最后一位单位	MPFR（80 位）	0.5	267,155.9
		Ziv（手写）	0.5	742.8
		Ziv（搜索）	0.5	383.4
	> 0.5最后一位单位	C	9.0×10^{15}	36.2
		Ziv（搜索）	1.3×10^{13}	37.7
Softmax（双变量）	≤ 0.5 最后一位单位	MPFR（79 位）	0.5	4,137.0
		Ziv（手写）	0.5	434.4
		Ziv（搜索得出）	0.5	184.8
	> 0.5最后一位单位	C	511.5	38.8
		Ziv（搜索得出）	449.5	31.2

LLVM-libc，Ziv的平均 0.82× 性能范围为 0.74× 至 0.96×。使用Ziv的高级抽象（DP1）取代现有实现中的函数专用代码，会带来适度的性能开销，但能大幅减小代码大小，具体量化结果如下。

我们报告了Ziv与原始基线实现的代码行数，见表2，结果表明Ziv 能够更简洁地表达现有库（RQ2）。我们的Ziv 重新实现的代码行数比例为 2.17:1（减少 53.8%），相较于 CORE-MATH 的原始实现；相较于 LLVM-libc 则为 3.59:1（减少 72.1%）。我们的重新实现表明，Ziv 能够表达现有的正确舍入库，并在提供更简洁实现的同时实现有竞争力的性能。此外，Ziv 代码比现有实现更易读，因为与§1和§2 （例如，图1和3；§2.1和§2.2）相比，它能更直接地表达高层次的数学意图。

7.2 使用Ziv（RQ3）编写新实现

我们使用 Ziv 编写新的实现，以考察它在开发新数学库实现方面的效果（RQ3）。我们重点关注三个函数：据我们所知，目前现有的定精度数学库中尚无这些函数的正确舍入实现：二重对数 [64], GELU [23],和双变量 softmax [22]。对于每个函数，我们评估Ziv 以及以正确舍入和宽松准确性为目标的基线实现，并通过实证评估正确舍入。

我们在表 3 中报告了新的 Ziv 实现及其基线实现的准确性和性能。对于以正确舍入（≤ 0.5 最后一位单位）为目标的实现，Ziv 的最大观测误差与使用 MPFR [18] 任意精度库的基线实现相当。由于我们没有找到正确舍入的定精度替代方案，因此使用 MPFR 基线实现。手写的 Ziv 在避免任意精度开销的同时，性能达到基线实现的 9.52× 至 359.7×。搜索得到的 Ziv 实现变体（§7.3）的性能达到手写 1.57× 至 2.35× 的 Ziv 实现。

对于表 > 0.5最后一位单位 中的宽松准确性目标（3），经过搜索的 Ziv 实现对于全部三个函数所达到的最大观测误差均低于 GNU 科学库（GSL） [20]和 C语言基线实现 [51] 。对于二重对数函数，Ziv 实现达到了 6.17× GSL 二重对数函数的性能；对于 GELU 和 Softmax，则分别达到了 0.96× 以及 1.24× C语言基线实现的性能。我们的新实现表明，Ziv 支持开发具有不同准确性目标的数学库函数，其性能与现有库相当或更优。

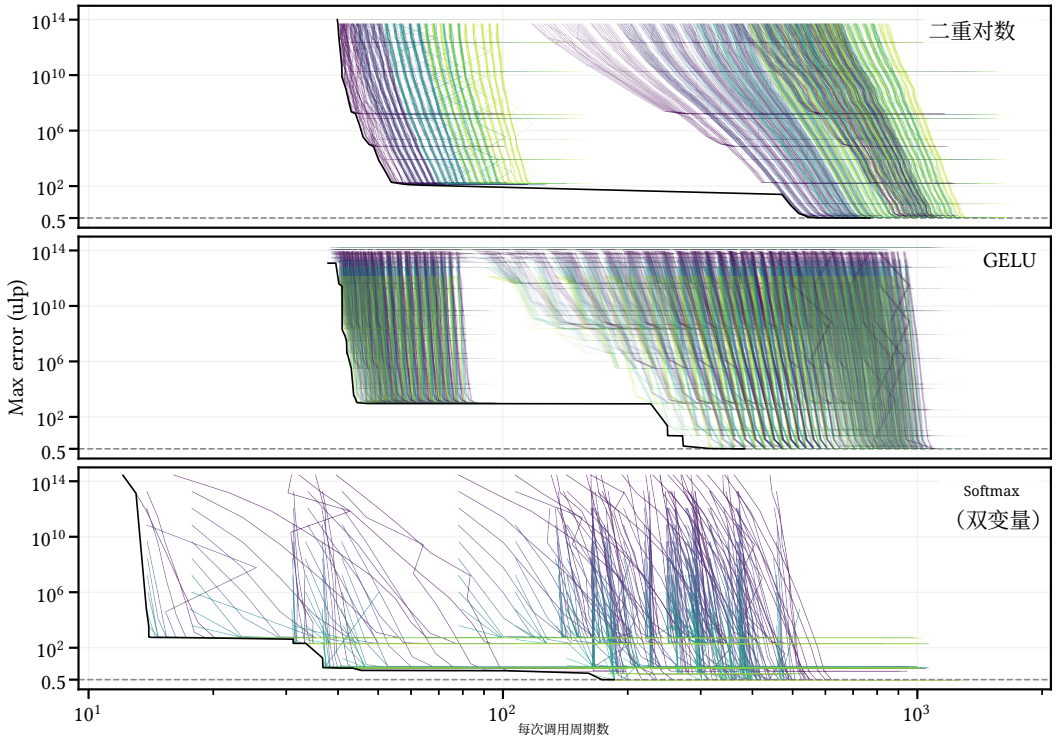


图 10. Zɪv生成的实现变体的观测到的最大 ulp 误差与每次调用周期数的关系。每条彩色曲线仅改变多项式次数，同时保持所有其他参数不变；曲线依次着色，以表示表格大小从 小 到 大。黑色曲线表示帕累托前沿。

7.3 探索Zɪv (RQ4)的实现变体

我们对 Zɪv 实现进行参数化、生成和测试，以评估 Zɪv 是否支持系统地探索实现变体（RQ4）。对于§7.2中的三个函数，我们根据工作精度、表大小和多项式次数对其Zɪv 代码进行参数化，并通过改变这些参数自动生成变体。为使探索保持可控，我们在 10^7 随机输入，以及 §7.2中的实验所发现的难以正确舍入的输入上测试每个变体的准确性。

我们在图 Zɪv 10中探讨生成的实现变体在准确性与性能之间的权衡。我们通过组合不同的工作精度、表大小和多项式次数，为二重对数生成了 72,000 个绘图变体，为 GELU 生成了 734,400 个，为 Softmax 生成了 17,856 个。由于每个函数的参数化方式不同，且各参数对准确性和性能的影响也各不相同，因此变体数量以及准确性–性能趋势也有所不同。当提高多项式次数不再改善准确性时，图中会出现水平平台；这是因为其他因素（例如工作精度不足）限制了准确性。在所探索的参数范围内，仅使用 64 位浮点数的变体都无法实现正确舍入，这表明需要 Zɪv 的扩展精度类型。

在我们的实验中，我们发现Zɪv的阶段化（DP2）非常适合通过参数化来生成和探索实现变体。例如，我们参数化的 GELU 实现具有简化的标签签名：`gelu(x : T, DEGREES : ⊥, TABLES : ⊥)`。Zɪv 编译器会针对 DEGREES和TABLES的编译时值专门化实现，运行时则仅将 x作为输入。遍历这些参数的不同取值，即可自动生成具有不同多项式次数和表大小的实现变体。

8 相关工作

正确舍入数学库。正确舍入数学库长期以来一直受到研究界和实践界的关注，*Brisebarre* 等人最近对此进行了综述 [9]。Ziv 为浮点算术中的成熟技术提供了抽象 [4, 47]，也为数学库实现提供了抽象 [2, 44, 46]。有多种库致力于实现正确舍入，包括 CR-LIBM [13], RLibm [41], CORE-MATH [57], 和 LLVM-libc [42]。我们在Ziv 与最后两者的对比见§7.1，因为它们在很大程度上代表了 64 位数学库的最新水平。除了定精度数学库之外，MPFR [18] 和 Arb [30] 等库还支持任意精度，这超出了我们的讨论范围。尽管数学库验证方面已有进展 [17, 21, 36, 37],，证明这些实现能够正确舍入仍需大量人工工作 [24, 65]，这也超出了我们的讨论范围。

数学库语言。 先前的研究提出了用于实现数学库的语言，但正确舍入并非其主要研究方向。Metalibm [35] 旨在根据高级规格生成数学函数，重点在于自动化程序合成。相比之下，Ziv 除了提供高级抽象，还支持对数值算法进行显式控制。Metalibm [10] 的后续版本引入了一种用于数学库代码生成的基于图的语言，但其重点是对汇编代码生成进行图级控制，而非语言设计。与之相反，Ziv 提供了专为正确舍入数学库定制的高级抽象。

最近，MegaLibm [7] 推出了一种用于数学库的高级语言，尽管其目标并非正确舍入。除了准确性目标之外，MegaLibm 和 Ziv 在设计原则上也大相径庭。MegaLibm 程序通过实数表达式表达意图，将浮点细节留给编译器处理。相比之下，Ziv程序指定浮点实现，并使用带有阶段化和正确舍入抽象的命令式语言。因此，Ziv 需要更多手动编程，但表达能力更强。 Ziv支持 MegaLibm 意图导向构造之外的技术，包括 Ziv 方法、制表、二进有理数和位运算，使其能够重新实现最先进的正确舍入库（§7.1）。

分阶段计算方法。 编程语言中的阶段化已得到广泛研究，现有综述中也提供了更广泛的背景 [31, 40]；本文重点介绍与 Ziv最相关的方法。轻量级模块化分阶段（LMS） [55] 使用类型来区分静态值与分阶段表达式，并构建动态计算的图。 Ziv 同样使用类型标签和追踪，但将最终的阶段化决策交给符号追踪，而不是类型系统（§5.2），这与 LMS 不同。类似地，机器学习框架使用符号追踪来记录程序 [1, 19]。

更为经典的是，Nielson 和 Nielson [48] 提出了一种方法，用于推断带类型的 λ演算中的静态和动态绑定时间注释。 Ziv 则使用标签来可靠地追踪哪些表达式能够保证在静态阶段求值（§5.2）。多阶段编程 [59] 通过显式的准引用表达式执行阶段化，而 Terra [15] 则使用类似的表达式级注释，实现一种嵌入 Lua 的阶段化语言。相比之下，Ziv的阶段化通过标签隐式完成，无需显式引用表达式即可进行简洁的编程。

9 结论

我们推出了 Ziv，这是一种用于开发正确舍入数学库的编程系统。Ziv通过扩展精度类型和丰富的标准库，支持高层次的数值抽象（DP1）；并通过对独立于输入的计算进行编译时求值，支持分阶段计算（DP2）。这些特性使高层次代码能够统一预计算与运行时计算。我们展示了 Ziv 能够有效地简洁表达现有实现、为新函数编写实用实现，并自动探索各种变体。

人工智能使用说明。 作者全面设计了 Ziv 语言、标准库和编译器。实现与评估工作绝大部分由作者完成，OpenAI Codex（GPT-6 Astra）仅提供了少量协助。实现过程中唯一直接使用 Codex 的情况，是协助移植多项式逼近例程，以及识别现有库中未记录的常数；所有由此产生的代码均由作者独立测试和验证。Codex 也用于构建部分 Lean构件，所有证明均由 Lean内核检查。

致谢

感谢浦项科技大学的 Hyeonjung Kim 协助重新实现 CORE-MATH 和 LLVM-libc。我们还感谢 CORE-MATH 项目的 Paul Zimmermann 和 Alexei Sibidanov，以及 LLVM-libc 项目的 Tue Ly，感谢他们就正确舍入数学库的设计进行讨论并提供指导。此外，我们感谢里昂高等师范学院 Pascaline 团队的 Guillaume Melquiond、Vincent Lefèvre、Claude- Pierre Jeannerod 和 Tom Hubrecht 提供宝贵的反馈。

本研究得到韩国国家研究基金会（NRF）RS-2026-25470496号资助，资助经费来自韩国科学与信息通信技术部（MSIT）。

参考文献

[1] Akshay Agrawal、Akshay Modi、Alexandre Passos、Allen Lavoie、Ashish Agarwal、Asim Shankar、Igor Ganichev、Josh Levenberg、Mingsheng Hong、Rajat Monga 和 Shanqing Cai。2019年。TensorFlow Eager：一种用于机器学习的多阶段、嵌入 Python 的领域特定语言。载于《机器学习与系统会议》（SysML）第2届会议论文集。https://mlsys.org/Conferences/2019/doc/2019/88.pdf [2] Nelson H. F. Beebe。2017年。《数学函数计算手册：使用 MathCW可移植软件库进行编程》。Springer。doi: 10 1007 978 3 319 64110 2 。 / - - - - Jeff Bezanson、Alan Edelman、Stefan Karpinski 和 Viral [3] B. Shah。2017年。Julia：数值计算的新方法。《SIAM评论》。 59, 1 2017 65 98 10 1137 141000671 （1月），–。doi:。 / Sylvie Boldo、Claude-Pierre Jeannerod、Guillaume Melquiond 和 Jean-Michel Muller。2023年。浮点算术。《Acta Numerica》32（2023年），203–290。doi:10.1017/S0962492922000101[5] Sylvie Boldo 和 Jean-Michel Muller。2005年。可通过融合乘加计算的一些函数。载于第17届IEEE计算机算术研讨会（ARITH）论文集。 52 58 10 1109 2005 39 –。 doi:。/ARITH.。Sylvie Boldo 和 Jean-Michel [6] Muller。。FMA的精确误差和近似误差。《IEEE计算机学报》60，2（ 2011 157 164 10 1109 2010 139 ），–。doi:。/TC.。Ian Briggs、Yash Lad 和 Pavel Panchekha。2024年。数学库函数的实现与合成。《ACM编程语言论文集》 8, 32 2024 10 1145 3632874 [7] POPL，文章（1月）。doi:。/ Nicolas Brisebarre 和 Sylvain Chevillard。。高效多项式近似。载于第18届IEEE计算机算术研讨会（ARITH）论文集。 169 176 10 1109 2007 17 –。doi:。/ARITH.。 Nicolas Brisebarre、[8] Guillaume Hanrot、Jean-Michel Muller 和 Paul Zimmermann。。函数的正确舍入求值：原因、方法及代价？《计算调查》 58, 1, 27 2026 10 1145 3747840 文章（1月）。doi:。/ [9] Nicolas Brunie、Florent de Dinechin、Olga Kupriianova 和 Christoph Lauter。。数学函数代码生成器。载于第22届IEEE计算机算术研讨会（ARITH）论文集。66–73。doi:10.1109/ARITH. 2015.22[11] Sylvain Chevillard、Mioara Joldes 和 Christoph Lauter。2010 年。Sollya：数值代码开发环境。载于第 3 届国际数学软件大会（ ICMS ）论文集。28–31。doi:10. 1007/978-3-642-15582-6 5_[12] William J. Cody, Jr. 和 William M. Waite。1980年。《初等函数软件手册》。Prentice-Hall。[13] Catherine Daramy-Loirat、David Defour、Florent de Dinechin、Matthieu Gallet、Nicolas Gast、Christoph Lauter 和 Jean-Michel Muller。2006年。CR-LIBM：双精度正确舍入初等函数库。研究报告。LIP。https://ens-lyon.hal.science/ensl-01529804 [14] 1971 Theodorus J. Dekker。。扩展可用精度的浮点技术。《Numerische Mathematik》18，3（6月 1971 224 242 10 1007 01397083 ），–。doi:。/BF Zachary DeVito、James Hegarty、Alex Aiken、Pat Hanrahan [15] 和 Jan Vitek。2013年。Terra：一种用于高性能计算的多阶段语言。载于第34届ACM SIGPLAN编程语言设计与实现会议（PLDI）论文集。 105 116 10 1145 2491956 2462166 –。doi:。/。Gerald Estrin。。计算机系统的组织：固定加可变结构 [16] 计算机。载于1960年西部联合IRE-AIEE-ACM计算机会议论文集。33–40。doi:10.1145/1460361.1460365

[17] Florian Faissole、Paul Geneau de Lamarlière 和 Guillaume Melquiond。2024年。快速且准确浮点近似的端到端形式化验证。载于第15届国际交互式定理证明会议（ITP）论文集。
14 1 14 18 10 4230 2024 14 : – :。doi:。/LIPIcs.ITP.。Laurent Fousse、[18] Guillaume Hanrot、Vincent Lefèvre、Patrick Pélissier 和 Paul Zimmermann。2007年。MPFR：一种支持正确舍入的多精度二进制浮点库。《ACM数学软件汇刊》33，2，文章13（2007年6月）。doi:10.1145/1236463.1236468[19] Roy Frostig、Matthew James Johnson 和 Chris Leary。2018年。通过高级追踪编译机器学习程序。载于第1届机器学习与系统会议（SysML）论文集。https://mlsys.org/Conferences/doc/2018/146.pdf [20] Mark Galassi、Jim Davies、James Theiler、Brian Gough、Gerard Jungman、Patrick Alken、Michael Booth 和 Fabrice Rossi。2009年。《GNU科学计算库参考手册》（第3版）。Network Theory Ltd. https://www.gnu.org/software/gsl/ [21] Paul Geneau de Lamarlière、Guillaume Melquiond 和 Florian Faissole。2023年。数学库的精简形式化证明。载于第30届IEEE计算机算术研讨会（ARITH）论文集。32–35。doi:10.1109/ARITH58626. 2023.00026[22] Ian Goodfellow、Yoshua Bengio 和 Aaron Courville。2016年。《深度学习》。麻省理工学院出版社。https://www.deeplearningbook.org [23] Dan Hendrycks 和 Kevin Gimpel。2023年。高斯误差线性单元（GELU）。https://arxiv.org/abs/1606.08415v5 arXiv:1606.08415v5 [cs.LG].[24] Tom Hubrecht、Claude-Pierre Jeannerod、Paul Zimmermann、Laurence Rideau 和 Laurent Théry。2024年。迈向binary64算术中正确舍入且快速的幂函数。https://inria.hal.science/hal-04159652v2 ARITH 2023论文的扩展版本。[25] IBM。2001年。IBM精确可移植数学库。https://github.com/dreal-deps/mathlib 原始库的非官方副本。
[26] 2019 *754 2019* 10 1109 2019 8766229 IEEE。。
[27] 2026 *754 3 23* *2026* IEEE Std -：IEEE浮点算术标准。doi:。/IEEESTD.。IEEE。。P /D。：IEEE浮点算术标准草案（11月）。https://listserv.ieee.org/cgi-bin/wa?A2=STDS-754;802b7116.26&S=[28] ISO/IEC。2024年。ISO/IEC 9899:2024：信息技术—编程语言—C。https://www.iso.org/standard/82075.html[29] Claude-Pierre Jeannerod、Mioara Joldes、Nicolas Louvet 和 Jean-Michel Muller。2026年。参数化双字重叠下的扩展精度FMA：紧致误差界与示例。载于第33届IEEE计算机算术研讨会（ARITH）论文集。
17 24 10 1109 69512 2026 00012 –。doi:。/ARITH。。Fredrik Johansson。。[30] 2017 Arb：高效的任意精度中点半径区间算术。《IEEE计算机学报》66，8（2017年），1281–1292。doi:10.1109/TC.2017.2690633[31] Neil D. Jones、Carsten K. Gomard 和 Peter Sestoft。1993年。《部分求值与自动程序生成》。Prentice Hall。https://raspi.itu.dk/people/sestoft/pebook/pebook.html [32] William Kahan。1997年。关于IEEE二进制浮点算术标准754现状的讲义。https://people.eecs.berkeley.edu/~wkahan/ieee754status/IEEE754.PDF 访问日期：2026-11-21。[33] Donald E. Knuth。1997年。《计算机程序设计艺术》（第3版）。第2卷：半数值算法。Addison-Wesley。https://cs.stanford.edu/~knuth/taocp.html [34] Toru Koizumi、Tomoaki Tsumura、Hidetsugu Irie 和 Shuichi Sakai。2023年。完美精度双精度指数函数的快速实现。IPSJ SIG技术报告2023-HPC-192，第21号。日本信息处理学会。https://ipsj.ixsq.nii.ac.jp/records/231099 日文。[35] Olga Kupriianova 和 Christoph Lauter。2014年。Metalibm：数学函数代码生成器。载于第
4 713 717 10 1007 978 3 662 44199 2_106 届国际数学软件大会（ICMS）论文集。–。doi:。/ - - - - [36] 2016 Wonyeol Lee、Rahul Sharma 和 Alex Aiken。。浮点数位操作的验证。载于第37届ACM SIGPLAN编程语言设计与实现会议（PLDI）论文集。70–84。doi:10.1145/2908080.2908107[37] Wonyeol Lee、Rahul Sharma 和 Alex Aiken。2018年。自动证明math.h实现的正确性。《ACM编程语言论文集》
2, 47 2018 10 1145 3158135 POPL，文章（1月）。doi:。/ [38] 2026 Vincent Lefèvre、Tue Ly 和 Paul Zimmermann。。计算双精度下sin、cos、tan的难以舍入的情形。载于第33届IEEE计算机算术研讨会（ARITH）论文集。113–120。doi:10.1109/ARITH69512. 2026.00024[39] Vincent Lefèvre 和 Jean-Michel Muller。2001年。双精度初等函数正确舍入的最坏情形。载于第15届IEEE计算机算术研讨会（ARITH）论文集。111–118。doi:10.1109/ARITH.2001.930110[40] Yannis Lilis 和 Anthony Savidis。2019年。元编程语言综述。《计算调查》52，6，文章113（2019年10月）。doi:10.1145/3354584

[41] Jay P. Lim 和 Santosh Nagarakatte。2021年。面向32位浮点表示的高性能正确舍入数学库。载于第42届 ACM SIGPLAN编程语言设计与实现会议（PLDI）论文集。 359 374 10 1145 3453483 3454049 – 。 doi: 。/ 。[42] LLVM开发者组。 。2026。LLVM C库：数学函数。https://libc.llvm.org/math/index.html 访问日期：2026-11-21。[43] Sandra Loosemore、Richard M. Stallman、Roland McGrath、Andrew Oram 和 Ulrich Drepper。2026年。GNU C库参考手册：数学。自由软件基金会。https://sourceware.org/glibc/manual/2.44/html_node/Mathematics.html 适用于2.44版。[44] Peter Markstein。2000年。IA-64与初等函数：速度与精度。Prentice Hall。[45] Ole Møller。1965年。浮点加法中的准双精度。BIT 5，1（1965年3月），37–50。doi:10.1007/BF01975722[46] Jean-Michel Muller。2016年。初等函数：算法与实现（第三版）。Birkhäuser。doi:10.1007/978-1-4899-7983-4[47] Jean-Michel Muller、Nicolas Brunie、Florent de Dinechin、Claude-Pierre Jeannerod、Mioara Joldes、Vincent Lefèvre、Guillaume Melquiond、Nathalie Revol 和 Serge Torres。2018年。浮点算术手册（第二版）。Birkhäuser。doi:10.1007/978-3-319-76526-6[48] Hanne R. Nielson 和 Flemming Nielson。1988年。类型化 λ-演算的自动绑定时间分析。载于第15届 ACM SIGPLAN-SIGACT程序设计语言原理研讨会（POPL）论文集。98–106。doi:10.1145/73560.73569[49] Yves Nievergelt。2003年。标量融合乘加指令可生成精确到倒数第二位的浮点矩阵算术。ACM数学软件汇刊 29, 1 2003 27 48 10 1145 641876 641878 （3月），– 。doi: 。/ 。[50] Oracle。 。2026。Java SE API 规范：Math 类。https://docs.oracle.com/en/java/javase/27/docs/api/java.base/ java/lang/Math.html 适用于第27版。[51] Adam Paszke、Sam Gross、Francisco Massa、Adam Lerer、James Bradbury、Gregory Chanan、Trevor Killeen、Zeming Lin、Natalia Gimelshein、Luca Antiga、Alban Desmaison、Andreas Köpf、Edward Yang、Zachary DeVito、Martin Raison、Alykhan Tejani、Sasank Chilamkurthy、Benoit Steiner、Lu Fang、Junjie Bai 和 Soumith Chintala。2019年。PyTorch：一种命令式风格的高性能深度学习库。载于第33届神经信息处理系统年会（NeurIPS）论文集。https://proceedings.neurips.cc/paper_files/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html[52] Mary H. Payne 和 Robert N. Hanek。1983年。三角函数的弧度约化。ACM SIGNUM Newslett. 18，1（1983年1月），19–24。doi:10.1145/1057600.1057602[53] Python软件基金会。2026年。Python文档：math——数学函数。https://docs.python.org/ 3.15/library/math.html 适用于3.15版。[54] Evgeny Y. Remez。1934年。关于一种用于确定逼近多项式的收敛逐次逼近方法。C. R. Acad. Sci. Paris 198（1934），2063–2065。https://gallica.bnf.fr/ark:/12148/bpt6k31506/f2063.item 法文。[55] Tiark Rompf 和 Martin Odersky。2010年。轻量级模块化分阶段：运行时代码生成与编译型领域特定语言的实用方法。载于第9届生成式编程与组件工程国际会议（GPCE）论文集。 127 136 10 1145 1868294 1868314 – 。 doi: 。/ 。[56] Jonathan R. Shewchuk。 。1997。自适应精度浮点算术与快速稳健几何谓词。离散与计算几何 18, 3 1997 305 363 10 1007 00009321 （10月），– 。doi: 。/PL[57] Alexei Sibidanov、Paul Zimmermann 和 Stéphane Glondu。 。2022。CORE-MATH项目。载于第29届IEEE计算机算术研讨会（ARITH）论文集。 26 34 10 1109 54963 2022 00014 – 。doi: 。/ARITH 。 。[58] Sun Microsystems。 。2004。libmcr：正确舍入的 libm。https://github.com/simonbyrne/libmcr 原始库的非官方副本。[59] Walid Taha 和 Tim Sheard。1997年。带显式注释的多阶段编程。载于1997年 ACM SIGPLAN部分求值与基于语义的程序操作研讨会（PEPM）论文集。203–217。doi:10.1145/258993.259019[60] Ping Tak Peter Tang。1989年。IEEE浮点算术中指数函数的表驱动实现。ACM数学软件汇刊 15, 2 1989 144 157 10 1145 63522 214389 （6月），– 。doi: 。/ 。[61] Ping Tak Peter Tang。 。1991。初等函数的查表算法及其误差分析。载于第 10 232 236 10 1109 1991 145565 届IEEE计算机算术研讨会（ARITH）论文集。– 。doi: 。/ARITH. 。[62] Julia项目。 。2026。Julia语言：数学运算与初等函数。https://docs.julialang. org/en/v1.13/manual/mathematical-operations/ 适用于1.13版。[63] Sage开发者。2026年。SageMath，Sage数学软件系统。https://www.sagemath.org/ [64] Don Zagier。2007年。双对数函数。载于《数论、物理与几何前沿 II》，Pierre Cartier、Pierre Moussa、Bernard Julia 和 Pierre Vanhove（编）。Springer，3–65。doi:10.1007/978-3-540-30308-4_1

[65] Paul Zimmermann。2026年。GNU libc 的 atanh 正确舍入。载于第*33*届*IEEE*计算机算术研讨会（*ARITH*）论文集。 97 104 10 1109 69512 2026 00022 – 。doi:。/ARITH 。 。Abraham Ziv。 。正确舍

[66] 1991入末位的初等数学函数快速求值。*ACM*数学软件汇刊17，3（1991年9月），410–423。doi:10.1145/114697.116813

Ziv 语言的完整定义

在本节　　中，我们定义 Ziv 语言的语法、类型系统和操作语义。

A.1 语法

Ziv 的抽象语法Zɪv如图11所示。

记号。 我们使用以下记号。

- $n \in \mathbb{Z}$是一个整数。
- $\ell \in$ 标签集 是一个标签，其中 标签集 $= \{\bot, \top\}$ 按照 $\bot \sqsubseteq \bot$、 $\bot \sqsubseteq \top$ 和 $\top \sqsubseteq \top$ 排序。
- $x \in$ 变量 是一个变量，其中 变量 是一个可数的变量名集合。
- $f \in$ 函数 是一个函数，其中 函数 是一个可数的函数名集合。
- $v \in$ 值Zɪv 是一个值，其中 值Zɪv 是 Ziv 中值的集合。集合 值Zɪv 由所有函数名和所有非函数值组成：

$$Val_{\text{Zɪv}} \coloneqq Fun \cup \bigcup_{\substack{T \in \text{Zɪv} \\ T \text{ is not a function type}}} Val_{erase(T)}.$$

这里，$erase(T)$ 是 T的擦除类型（即不含标签的类型），而 Val_θ是擦除类型 $\theta = erase(T)$ 的值的集合，其定义如下：

$$\begin{aligned}
Val_{\text{Bool}} &\coloneqq \{\text{val}_{\text{Bool}}(c) \mid c \in \{\text{true}, \text{false}\}\},\\
Val_{\text{Int}_n} &\coloneqq \{\text{val}_{\text{Int}_n}(c) \mid c \in \mathbb{Z} \wedge c \in [-2^{n-1}, 2^{n-1})\},\\
Val_{\text{UInt}_n} &\coloneqq \{\text{val}_{\text{UInt}_n}(c) \mid c \in \mathbb{Z} \wedge c \in [0, 2^n)\},\\
Val_{\text{Float}_n} &\coloneqq \{\text{val}_{\text{Float}_n}(c) \mid c \in \{0,1\}^n\} \text{ is the set of } n\text{-bit IEEE-754 floats},\\
Val_{\text{Real}} &\coloneqq \{\text{val}_{\text{Real}}(c) \mid c \in \mathbb{R}\},\\
Val_{\text{Dyadic}[\tau_1,\tau_2,n]} &\coloneqq \{\text{val}_{\text{Dyadic}[\tau_1,\tau_2,n]}(c) \mid c \in Val_{\tau_1} \times Val_{\tau_2}\},\\
Val_{\text{Dyadic}[\tau_1,\tau_2,\tau_3]} &\coloneqq \{\text{val}_{\text{Dyadic}[\tau_1,\tau_2,\tau_3]}(c) \mid c \in Val_{\tau_1} \times Val_{\tau_2} \times Val_{\tau_3}\},\\
Val_{\text{MultiWord}[\theta_1,\ldots,\theta_n]} &\coloneqq \{\text{val}_{\text{MultiWord}[\theta_1,\ldots,\theta_n]}(c) \mid c \in Val_{\theta_1} \times \cdots \times Val_{\theta_n}\},\\
Val_{\text{Tuple}[\theta_1,\ldots,\theta_n]} &\coloneqq \{\text{val}_{\text{Tuple}[\theta_1,\ldots,\theta_n]}(c) \mid c \in Val_{\theta_1} \times \cdots \times Val_{\theta_n}\},\\
Val_{\text{Table}[\theta,(n_1,\ldots,n_k)]} &\coloneqq \{\text{val}_{\text{Table}[\theta,(n_1,\ldots,n_k)]}(c) \mid c \in (Val_\theta)^{n_1 \times \cdots \times n_k}\},\\
Val_{\bar{\theta} \to \theta'} &\coloneqq \varnothing.
\end{aligned}$$

如上所示，每个非函数值 $\text{val}_\theta(c)$ 都带有其擦除类型 θ。

- $T^\bot$ 是一个所有标签均为 $\bot$ 的类型。
- 横线是一个有限序列，可以为空。例如：$\bar{}- E$是 $E_1, \ldots, E_n$，其中 $n \geq 0$ 取某个值。$\bar{}-\bar{x}\colon T$是 $x_1\colon T_1, \ldots, x_n\colon T_n$，其中 $n \geq 0$ 取某个值。

语法限制。 我们对 Ziv 的语法作如下限制。

- 类型 Int_n、UInt_n 和 Float_n: $n \geq 1$。
- 二进有理数类型$[\tau_1^{\ell_1}, \tau_2^{\ell_2}, n]$: τ_1是布尔型， τ_2是无符号整数$_k$，且 $n \in \mathbb{Z}$。
- 二进有理数类型$[\tau_1^{\ell_1}, \tau_2^{\ell_2}, \tau_3^{\ell_3}]$: τ_1是布尔型， τ_2是无符号整数$_k$，且 τ_3是 $\text{Int}_{k'}$。
- 多字类型$[\bar{T}]$: $|\bar{T}| \geq 1$和 T_i可以是浮点数$_n^\ell$、二进有理数$[\tau_1^{\ell_1}, \tau_2^{\ell_2}, n]$或二进有理数$[\tau_1^{\ell_1}, \tau_2^{\ell_2}, \tau_3^{\ell_3}]$。
- 元组类型$[\bar{T}]$: $|\bar{T}| \geq 1$和 T_i可以是 τ^ℓ、二进有理数$[\tau_1^{\ell_1}, \tau_2^{\ell_2}, n]$、二进有理数$[\tau_1^{\ell_1}, \tau_2^{\ell_2}, \tau_3^{\ell_3}]$或多字$[\bar{T}\prime]$。
- 表类型$[T^\bot, (n_1, \ldots, n_k)]$: $k \geq 1$、 $n_1, \ldots, n_k \geq 1$，且 T可以是 τ^ℓ、二进有理数$[\tau_1^{\ell_1}, \tau_2^{\ell_2}, n]$、二进有理数$[\tau_1^{\ell_1}, \tau_2^{\ell_2}, \tau_3^{\ell_3}]$、多字$[\bar{T}\prime]$,或元组$[\bar{T}\prime]$。
- 语句 while $E\,\{S\}$: S没有返回语句。

- 程序（def $f_i(\bar{x}_i\colon \bar{T}_i)\colon T_i\{S_i\})^{i\in[n]}$：$f_1, \ldots, f_n$互不相同，且 $x_{i,1}, \ldots, x_{i,|\bar{x}_i|}$互不相同。
- 带类型标注的运算符 $op^{\ell}[\bar{\tau}\to\tau']$：$op[\bar{\tau}\to\tau']\in Op_{\text{ZIv}}$和 $(\ell=\bot)\Longleftrightarrow(\exists i.\ \tau_i\notin \text{ZIvIR})$，其中 Op_{ZIv} 是 ZIv 中的可数运算符集合，而ZIvIR 中的类型定义见附录 B.1。Op_{ZIv} 的示例如下：

$$\begin{aligned} Op_{\text{ZIv}} = \{&\&[(\text{Bool},\text{Bool})\to\text{Bool}], ==[(\text{Bool},\text{Bool})\to\text{Bool}],\ldots,\\ &+[(\text{Int}_n,\text{Int}_n)\to\text{Int}_n], \le[(\text{Int}_n,\text{Int}_n)\to\text{Bool}],\ldots,\\ &+[(\text{UInt}_n,\text{UInt}_n)\to\text{UInt}_n], \le[(\text{UInt}_n,\text{UInt}_n)\to\text{Bool}],\ldots,\\ &+[(\text{Float}_n,\text{Float}_n)\to\text{Float}_n], \le[(\text{Float}_n,\text{Float}_n)\to\text{Bool}],\ldots,\\ &+[(\text{Real},\text{Real})\to\text{Real}], -[(\text{Real},\text{Real})\to\text{Real}],\ldots,\\ &\text{cast}[\text{UInt}_n\to\text{Int}_n], \text{cast}[\text{Float}_n\to\text{Real}],\ldots,\\ &\text{reint}[\text{Float}_n\to\text{UInt}_n], \text{reint}[\text{UInt}_n\to\text{Float}_n],\ldots\}. \end{aligned}$$

A.2 类型系统

ZIv的类型系统如图14所示。

记号。 我们使用以下记号。

- 对于 $n\in\mathbb{Z}$，$[n]$和 $[n]_0$分别是集合 {1、……、 n} 和 {0、……、 $n-1$}。
- 对于擦除类型 θ，$\theta^{\perp}$是仅用 $\perp$标注 θ后得到的类型。
- 对于 ℓ，$\ell'\in\{\perp,\top\}$，$\ell\sqcup\ell'$是 ℓ和 ℓ'的最小上界（或并）。
- 对于类型 T和 $\ell\in$标签集，$T\sqcup\ell$是将 T中的每个标签 ℓ'替换为 $\ell'\sqcup\ell$后得到的类型。若得到的类型无效（例如，实数$^{\top}$），或者 T是函数类型且 $\ell=\top$，则未定义。
- 对于类型 T和 T'，$T\sqcup T'$是通过对 T和 T'中的对应标签取并而得到的类型，条件是 $erase(T)=erase(T')$。若 $erase(T)\neq erase(T')$，或者 T 或 T'是函数类型且 $T\neq T'$，则未定义。

- 对于类型 T和 T'，子类型关系 $T\sqsubseteq T'$定义如下：

$$\begin{aligned} T\sqsubseteq T' \Longleftrightarrow{}& \text{擦除}(T)=\text{擦除}(T')\ \wedge(\ell\sqsubseteq\ell' \text{对于 } T\text{和 } T'\text{ 中所有对应的标签 } \ell\text{和 } \ell')\\ &\wedge (T \text{ or } T' \text{ is a function type} \implies T=T'). \end{aligned}$$

- 对于类型上下文 Γ 和 Γ'，细化关系 $\Gamma\preceq\Gamma'$ 定义如下：

$$\Gamma\preceq\Gamma' \iff \text{dom}(\Gamma)\supseteq\text{dom}(\Gamma')\wedge(\Gamma(x)\sqsubseteq\Gamma'(x) \text{ for all } x\in\text{dom}(\Gamma')).$$

辅助函数。 我们定义了图 14中使用的以下辅助函数。

- 对于语句 S，函数 返回(S) 定义如下：

$$ret(S) \coloneqq \begin{cases} true & \text{if } S \text{ has a return statement}\\ false & \text{otherwise.}\end{cases}$$

- 对于类型 T，函数 字段(T) 定义如下：

$$\begin{aligned} fields(\tau^{\ell}) &\coloneqq \text{undefined}, & fields(\bar{T}\to T') &\coloneqq \text{undefined},\\ fields(\text{Dyadic}[\tau_1^{\ell_1},\tau_2^{\ell_2},n]) &\coloneqq (\tau_1^{\ell_1},\tau_2^{\ell_2}), & fields(\text{Tuple}[\bar{T}]) &\coloneqq (\bar{T}),\\ fields(\text{Dyadic}[\tau_1^{\ell_1},\tau_2^{\ell_2},\tau_3^{\ell_3}]) &\coloneqq (\tau_1^{\ell_1},\tau_2^{\ell_2},\tau_3^{\ell_3}), & fields(\text{Table}[T^{\perp},\bar{n}]) &\coloneqq (\underbrace{T^{\perp},\ldots,T^{\perp}}_{n_1\times\cdots\times n_{|\bar{n}|}}),\\ fields(\text{MultiWord}[\bar{T}]) &\coloneqq (\bar{T}). \end{aligned}$$

- 对于类型判断结果 Ω，Ω、集合 W W、$\subseteq$ 变量以及 $\ell \in$ 标签集，函数 合并$(\ell,\ (\Omega,\ W),\ (\Omega,$ $W))$ 定义如下：

$$merge\big(\ell, (\Omega, W), (\Omega', W')\big) \coloneqq \begin{cases} \big((\Gamma \cap \Gamma') \sqcup_{W \cup W'} \ell,\ W \cup W'\big) & \text{if } (\Omega, \Omega') = (\Gamma, \Gamma') \\ (\Gamma' \sqcup_{W'} \ell,\ W') & \text{if } (\Omega, \Omega') = (\text{ret}, \Gamma') \\ (\Gamma \sqcup_{W} \ell,\ W) & \text{if } (\Omega, \Omega') = (\Gamma, \text{ret}) \\ (\text{ret}, \varnothing) & \text{if } (\Omega, \Omega') = (\text{ret}, \text{ret}). \end{cases}$$

此处，上下文交集 $\Gamma \cap \Gamma'$ 和上下文更新 $\Gamma \sqcup_W \ell$ 定义如下 ：

$$(\Gamma \cap \Gamma')(x) \coloneqq \begin{cases} \Gamma(x) \sqcup \Gamma'(x) & \text{if } x \in \text{dom}(\Gamma) \cap \text{dom}(\Gamma') \\ \text{undefined} & \text{otherwise,} \end{cases}$$

$$(\Gamma \sqcup_W \ell)(x) \coloneqq \begin{cases} \big(x \in W \ ?\ \Gamma(x) \sqcup \ell : \Gamma(x)\big) & \text{if } x \in \text{dom}(\Gamma) \\ \text{undefined} & \text{otherwise.} \end{cases}$$

A.3 操作语义

Ziv 的操作语义如图 所示15。

我们使用以下记号。

- 对于有限序列 $\bar{a}$，$|\bar{a}|$ 是 $\bar{a}$的长度。
- 对于$op[(\tau_1,\ \ldots\ldots,\ \tau_n) \longrightarrow \tau\prime] \in Op_{\text{Ziv}}$，我们用全函数表示其语义：

$$[\![op[(\tau_1, \ldots, \tau_n) \longrightarrow \tau']]\!] : Val_{\tau_1} \times \cdots \times Val_{\tau_n} \to Val_{\tau'}.$$

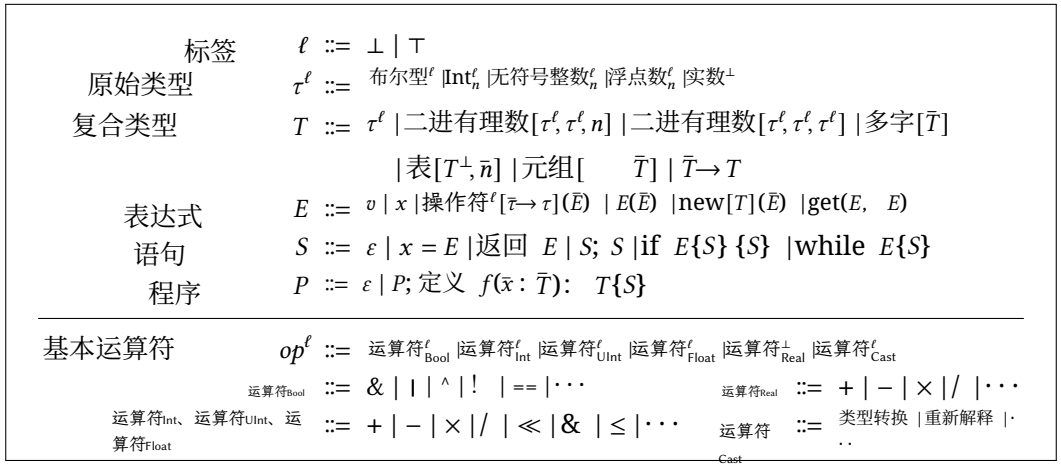


图 11. Ziv 的核心语法。

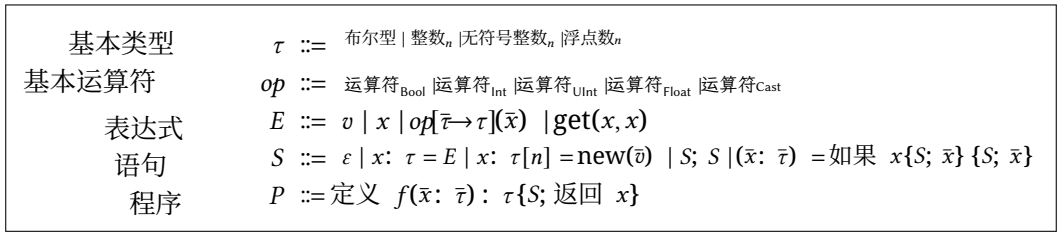


图 12. ZivIR 的核心语法。

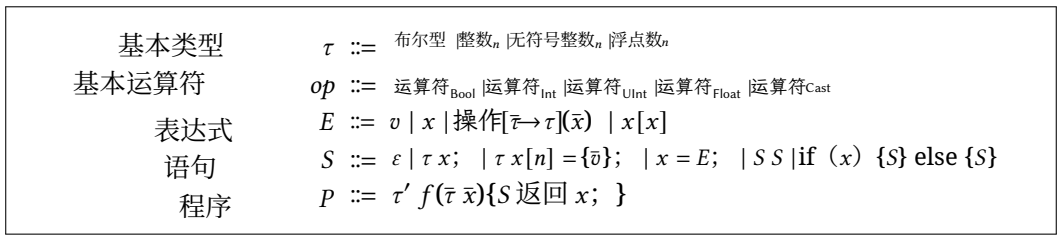


图 13. C语言的核心语法$_0$。

定义 （类型上下文、类型判断结果、写集合） $\boxed{\Gamma,\ \Omega,\ W}$

$$\Gamma ::= \varepsilon \mid \Gamma,\ x:\ T \qquad \Omega ::= \Gamma \mid \text{返回} \qquad W \subseteq \text{变量}$$

表达式 $\boxed{P;\Gamma \vdash E:\ T}$

$$\frac{v \in \text{值}_\theta}{P;\ \Gamma \vdash v:\ \theta^\perp}\ \text{T-常量} \qquad \frac{v \in \text{函数} \quad (\text{def}\ v(\bar{x}:\bar{T}):\ T'\ \{S\}) \in P}{P;\ \Gamma \vdash v:\ \bar{T} \to T'}\ \text{T-函数} \qquad \frac{}{P;\Gamma \vdash x:\ \Gamma(x)}\ \text{T-变量}$$

$$\frac{\forall i \in [n].\ P;\Gamma \vdash E_i:\ \tau_i^{\ell_i} \quad \forall i \in [n].\ \ell_i \sqsubseteq \ell \qquad \ell_1 \sqcup \cdots \sqcup \ell_n = \ell'}{P;\ \Gamma \vdash \text{操作}^{\ell}[\ (\tau_1,\ \ldots\ldots,\ \tau_n) \to \tau']\ (E_1,\ \ldots\ldots,\ E_n):\ (\tau')^{\ell'}}\ \text{T-Op}$$

$$\frac{P;\ \Gamma \vdash E_0:\ (T_1, \ldots, T_n) \to T' \quad \forall i \in [n].\ P;\Gamma \vdash E_i:\ T_i}{P;\ \Gamma \vdash E_0\ (E_1、\ \ldots\ldots、\ E_n):\ T'}\ \text{T-Call}$$

$$\frac{\text{字段}(T') = (T_1, \ldots, T_n) \wedge n \geq 1 \quad \forall i \in [n].\ P;\ \Gamma \vdash E_i:\ T_i}{P;\ \Gamma \vdash \text{new}[T_\prime]\ (E_1、\ \ldots\ldots、\ E_n):\ T_\prime}\ \text{T-New}$$

$$\frac{P;\ \Gamma \vdash E:\ T' \qquad \text{字段}(T') = (T_1, \ldots, T_n) \wedge i \in [n]_0}{P;\ \Gamma \vdash \text{get}(E, \text{值}^{\cup \text{Int}_k}(i)):\ T_{i+1}}\ \text{T-GetConsInd}$$

$$\frac{P;\ \Gamma \vdash E_1:\ \text{表}[T, \bar{n}] \quad P;\Gamma \vdash E_2:\ \text{无符号整数}_k^\ell}{P;\ \Gamma \vdash \text{get}(E_1,\ E_2):\ T \sqcup \ell}\ \text{T-GetExprInd} \qquad \frac{P;\Gamma \vdash E:\ T \qquad T \sqsubseteq T'}{P;\ \Gamma \vdash E:\ T'}\ \text{T-Sub}$$

语句 $\boxed{P;\Gamma \vdash_T S : (\Omega,\ W)}$

$$\frac{}{P;\ \Gamma \vdash_T \varepsilon:\ (\Gamma,\ \varnothing)}\ \text{T-Skip} \qquad \frac{P;\ \Gamma \vdash E:\ T'}{P;\ \Gamma \vdash_T x = E:\ ((\Gamma,\ x:\ T'),\ \{x\})}\ \text{T-Assn}$$

$$\frac{P;\ \Gamma \vdash E:\ T}{P;\ \Gamma \vdash_T \text{返回}\ E : (\text{返回}\ \varnothing)}\ \text{T-Ret} \qquad \frac{P;\ \Gamma \vdash_T S_1 : (\text{返回}\ \varnothing)}{P;\ \Gamma \vdash_T S_1; S_2 : (\text{返回}\ \varnothing)}\ \text{T-顺序1}$$

$$\frac{P;\Gamma \vdash_T S_1 : (\Gamma_1, W_1) \qquad P;\Gamma_1 \vdash_T S_2 : (\Omega_2, W_2)}{P;\ \Gamma \vdash_T S_1;\ S_2 : (\Omega_2, (\Omega_2 = \text{返回}\ ?\ \varnothing :\ W_1 \cup W_2))}\ \text{T-顺序2}$$

$$\frac{P;\Gamma \vdash E : \text{布尔型}^\ell \qquad \begin{array}{l} P;\Gamma \vdash_T S_1 : (\Omega_1, W_1) \\ P;\Gamma \vdash_T S_2 : (\Omega_2, W_2) \end{array} \qquad \begin{array}{l} \text{合并}(\ell, (\Omega_1, W_1), (\Omega_2, W_2)) = (\Omega', W\ ') \\ \text{返回}(S_1) \vee \text{返回}(S_2) \Longrightarrow T \sqcup \ell = T \end{array}}{P;\ \Gamma \vdash_T \text{如果}\ E\ \{S_1\}\ \{S_2\} :\ (\Omega', W\ ')}\ \text{T-If}$$

$$\frac{\neg \text{返回}(S) \qquad \begin{array}{l} P;\Gamma_I \vdash E:\ \text{布尔型}^\perp \\ P;\Gamma_I \vdash_T S : (\Gamma',\ W\ ') \end{array} \qquad \begin{array}{l} \Gamma \preceq \Gamma_I \\ \Gamma' \preceq \Gamma_I \end{array}}{P;\ \Gamma \vdash_T \text{while}\ E\{S\} :\ (\Gamma_I,\ W\ ')}\ \text{T-While}$$

程序 $\boxed{\vdash P : \text{ok}}$

$$\frac{P = (\text{def}\ f_j(\bar{x_j}:\bar{T_j}):\ T'_j\{S_j\})_{j \in [n]} \qquad \forall i \in [n].\ P;\ (\bar{x_i}:\bar{T_i}) \vdash_{T'_i} S_i : (\text{返回}\ \varnothing)}{\vdash P:\ \text{正常}}\ \text{T-程序}$$

图 14。Ziv 的类型系统。

定义（状态、求值结果） $\boxed{\sigma, \omega}$

$$\sigma ::= \varepsilon \mid \sigma,\ \ x \mapsto v \qquad\qquad \omega ::= \sigma \mid \text{返回}\ (v)$$

表达式 $\boxed{P;\ \ \sigma \vdash E \Downarrow v}$

$$\frac{v \in \text{值}_{\text{ZIV}}}{P; \sigma \vdash v \Downarrow v}\ \text{E-常量} \qquad \frac{\sigma(x) = v}{P; \sigma \vdash x \Downarrow v}\ \text{E-变量} \qquad \frac{\forall i \in [n].\ P; \sigma \vdash E_i \Downarrow v_i \quad [\![op[(\tau_1、\ \ldots、\ \tau_n) \to \tau']]\!](v_1、\ \ldots、\ v_n) = v'}{P; \sigma \vdash op^{\ell}[(\tau_1、\ \ldots、\ \tau_n) \to \tau'](E_1、\ \ldots、\ E_n) \Downarrow v'}\ \text{E-Op}$$

$$\frac{P; \sigma \vdash E_0 \Downarrow f \qquad (\text{def}\ f(x_1 \colon T_1、\ \ldots、\ x_n \colon T_n) \colon T'\ \{S\}) \in P \qquad \forall i \in [n].\ P; \sigma \vdash E_i \Downarrow v_i \qquad P;\ \ (x_1 \mapsto v_1, \ldots, x_n \mapsto v_n)\ \vdash S \Downarrow \text{返回}\ (v')}{P;\ \ \sigma \vdash E_0\ (E_1,\ \ldots\ldots,\ E_n)\ \Downarrow v'}\ \text{E-调用}$$

$$\frac{\text{字段}\ (T)\ = (T_1, \ldots, T_n)\ \wedge \text{擦除}\ (T)\ = \theta \qquad \forall i \in [n].\ P;\ \ \sigma \vdash E_i \Downarrow v_i \wedge v_i \in \text{值}_{erase(T_i)}}{P;\ \ \sigma \vdash \text{new}[T]\ (E_1, \ldots, E_n)\ \Downarrow \text{值}_{\theta}\ (v_1, \ldots, v_n)}\ \text{E-新建} \qquad \frac{P;\ \ \sigma \vdash E_1 \Downarrow \text{值}_{\theta}\ (v_1, \ldots, v_n) \qquad P; \sigma \vdash E_2 \Downarrow \text{值}_{\text{UInt}_k}(i)\ \wedge i \in [n]_0}{P; \sigma \vdash \text{get}(E_1,\ \ E_2)\ \Downarrow v_{i+1}}\ \text{E-获取}$$

语句 $\boxed{P; \sigma \vdash S \Downarrow \omega}$

$$\frac{}{P; \sigma \vdash \varepsilon \Downarrow \sigma}\ \text{E-跳过} \qquad \frac{P; \sigma \vdash E \Downarrow v}{P; \sigma \vdash x = E \Downarrow (\sigma,\ \ x \mapsto v)}\ \text{E-赋值} \qquad \frac{P; \sigma \vdash E \Downarrow v}{P; \sigma \vdash \text{返回}\ \ E \Downarrow \text{返回}(v)}\ \text{E-Ret}$$

$$\frac{P; \sigma \vdash S_1 \Downarrow \text{返回}(v)}{P;\ \ \sigma \vdash S_1; S_2 \Downarrow \text{返回}(v)}\ \text{E-Seq1} \qquad \frac{P; \sigma \vdash S_1 \Downarrow \sigma_1 \qquad P; \sigma_1 \vdash S_2 \Downarrow \omega_2}{P;\ \ \sigma \vdash S_1; S_2 \Downarrow \omega_2}\ \text{E-Seq2}$$

$$\frac{P; \sigma \vdash E \Downarrow \text{值}_{\text{Bool}}(\text{true}) \qquad P; \sigma \vdash S_1 \Downarrow \omega_1}{P; \sigma \vdash \text{if}\ \ E\ \{S_1\}\ \{S_2\}\ \ \Downarrow \omega_1}\ \text{E-IfTrue} \qquad \frac{P; \sigma \vdash E \Downarrow \text{值}_{\text{Bool}}(\text{false}) \qquad P; \sigma \vdash S_2 \Downarrow \omega_2}{P; \sigma \vdash \text{if}\ \ E\ \{S_1\}\ \{S_2\}\ \ \Downarrow \omega_2}\ \text{E-IfFalse}$$

$$\frac{P; \sigma \vdash E \Downarrow \text{值}_{\text{Bool}}(\text{false})}{P;\ \ \sigma \vdash \text{当}\ \ E\{S\}\ \ \Downarrow \sigma}\ \text{E-WhileFalse} \qquad \frac{P; \sigma \vdash E \Downarrow \text{值}_{\text{Bool}}(\text{true}) \qquad P; \sigma \vdash S \Downarrow \sigma' \qquad P;\ \ \sigma' \vdash \text{当}\ \ E\{S\}\ \ \Downarrow \sigma''}{P;\ \ \sigma \vdash \text{当}\ \ E\{S\}\ \ \Downarrow \sigma''}\ \text{E-WhileTrue}$$

程序 $\boxed{P \vdash f(\bar{v})\ \ \Downarrow v'}$

$$\frac{P = (\text{def}\ f_j(\bar{x}_j : \bar{T}_j) : T'_j\ \{S_j\})_{j \in [n]} \qquad i \in [n] \wedge |\bar{x}_i| = |\bar{T}_i| = |\bar{v}_i| \qquad P; (\bar{x}_i \mapsto \bar{v}_i) \vdash S_i \Downarrow \text{ret}(v')}{P \vdash f_i(\bar{v}_i) \Downarrow v'}\ \text{E-Prog}$$

图15. Ziv 的操作语义。

B 完整定义中间语言

本节定义三种中间语言：ZivNorm、ZivIR 和 C_0 的语法、类型系统和操作语义。

B.1 语法

的抽象语法ZivNorm 与Ziv （图11）相同，但ZivNorm 中的每个函数必须恰好包含一条返回语句，且该语句必须位于函数体末尾。ZivIR 和C_0 的抽象语法分别如图12 和13所示。

***No*记号。**我们使用与附录A.1相同的记号，但有以下区别 .

- 在 ZivIR 中， $v \in$ 值ZivIR 其中 值ZivIR 是 ZivIR 中值的集合。集合 值ZivIR 由所有原始类型和数组类型的值组成：

$$Val_{\text{ZivIR}} \coloneqq \bigcup_{\tau \in \text{ZivIR}} Val_\tau \cup \bigcup_{\tau \in \text{ZivIR},\, n \geq 1} Val_{\tau[n]}.$$

值$_\tau$ 定义见附录 A.1，而 值$_{\tau[n]}$ 定义如下：

$$Val_{\tau[n]} \coloneqq \{\text{val}_{\tau[n]}(v_1, \ldots, v_n) \mid v_1, \ldots, v_n \in Val_\tau\}.$$

- 在 C_0 中， $v \in$ 值$_{C_0}$ 其中 值$_{C_0}$ 是 C_0 中值的集合。集合 值$_{C_0}$ 由所有原始类型和数组类型的值组成：

$$Val_{C_0} \coloneqq \bigcup_{\tau \in C_0} Val_\tau \cup \bigcup_{\tau \in C_0,\, n \geq 1} Val_{\tau[n]}.$$

值$_\tau$, 值$_{\tau[n]}$ 如上所述。由于 ZivIR 和 C_0 具有相同的类型，值$_{C_0}$ = 值$_{\text{ZivIR}}$.

语法限制。 我们对 ZivIR 的语法作出以下限制：

- 类型 Int$_n$、UInt$_n$ 和 Float$_n$: $n \geq 1$。
- 语句 $(x\colon\ \tau[n] = \text{new}(\bar{v}))\colon\ n \geq 1$。
- 语句 $((\bar{x}\colon\ \bar{\tau})\ = \text{if}\ y\{S_1;\ \bar{z}_1\}\ \{S_2;\ \bar{z}_2\})\colon\ |\bar{x}| = |\bar{\tau}| = |\bar{z}_1| = |\bar{z}_2|$。
- 程序 (def $f(x_1\colon\ \tau_1, \ldots, x_n\colon\ \tau_n)\colon\ \tau\prime\{S;$ 返回 $y\}$): 变量 $x_1, \ldots, x_n$ 以及 S 中赋值语句左侧出现的所有变量两两不同。
- 带类型标注的运算符 $op[\bar{\tau} \to \tau\prime]$: $op[\bar{\tau} \to \tau\prime] \in Op_{\text{ZivIR}}$，其中 $Op_{\text{ZivIR}} \subseteq Op_{\text{Ziv}}$ 定义为

$$Op_{\text{ZivIR}} \coloneqq \{op[(\tau_1, \ldots, \tau_n) \to \tau_{n+1}] \in Op_{\text{Ziv}} \mid \tau_1, \ldots, \tau_{n+1} \in \text{ZivIR}\}.$$

我们对 C_0 的语法作如下限制：

- 类型 Int$_n$、无符号整数 UInt$_n$ 和浮点数 Float$_n$： $n \geq 1$。
- 语句 （$\tau\ x[n] = \{\bar{v}\};$ ） ： $n \geq 1$。
- 程序 （$\tau\prime\ f(x_1\colon\ \tau_1,\ \ldots\ldots,\ x_n\colon\ \tau_n)\{S;$ 返回 $y;\ \})\colon\ x_1,\ \ldots\ldots,\ x_n$ 各不相同。
- 带类型标注的运算符 $op[\bar{\tau} \to \tau\prime]$: $op[\bar{\tau} \to \tau\prime] \in Op_{\text{C语言}_0}$，其中$Op_{\text{C语言}_0} \subseteq Op_{\text{ZivIR}}$ 定义为 b y

$$Op_{C_0} \coloneqq \{op[(\tau_1, \ldots, \tau_n) \to \tau_{n+1}] \in Op_{\text{ZivIR}} \mid \tau_1, \ldots, \tau_{n+1} \in C_0\}.$$

由于 ZivIR 和 C语言$_0$ 具有相同的类型，$Op_{C_0} = Op_{\text{ZivIR}}$。

B.2 类型系统

的类型系统与 ZivNorm 的相同Ziv （图14）。ZivIR 和C语言$_0$ 的类型系统分别见图 16和 17。

定义（类型上下文） $\boxed{\Gamma}$

$$\Gamma ::= \varepsilon \mid \Gamma, x : \tau \mid \Gamma, x : \tau[n]$$

表达式 $\boxed{\Gamma \vdash E : \tau}$

$$\frac{v \in Val_\tau}{\Gamma \vdash v : \tau}\ \text{T-Const} \qquad \frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}\ \text{T-Var} \qquad \frac{\forall i \in [n].\ \Gamma(x_i) = \tau_i}{\Gamma \vdash op[(\tau_1, \ldots, \tau_n) \to \tau'](x_1, \ldots, x_n) : \tau'}\ \text{T-Op}$$

$$\frac{\Gamma(x) = \tau[n] \qquad \Gamma(y) = \mathsf{UInt}_k}{\Gamma \vdash \mathsf{get}(x, y) : \tau}\ \text{T-Get}$$

语句 $\boxed{\Gamma \vdash S : \Gamma'}$

$$\frac{}{\Gamma \vdash \varepsilon :\ \Gamma}\ \text{T-Skip} \qquad \frac{\Gamma \vdash E :\ \tau}{\Gamma \vdash (x :\ \tau = E) :\ (\Gamma,\ x :\ \tau)}\ \text{T-Assn}$$

$$\frac{n \geq 1 \qquad \forall i \in [n].\ v_i \in \text{值}_\tau}{\Gamma \vdash (x :\ \tau[n] = \mathsf{new}(v_1, \ldots, v_n)) :\ (\Gamma, x :\ \tau[n])}\ \text{T-New} \qquad \frac{\Gamma \vdash S_1 :\ \Gamma_1 \qquad \Gamma_1 \vdash S_2 :\ \Gamma_2}{\Gamma \vdash S_1;\ S_2 :\ \Gamma_2}\ \text{T-Seq}$$

$$\frac{\begin{array}{c}\Gamma \vdash S_1 :\ \Gamma_1 \qquad \forall i \in [n].\ \Gamma_1 \vdash z_{1,i} :\ \tau_i \\ \Gamma \vdash y :\ \text{布尔型} \qquad \Gamma \vdash S_2 :\ \Gamma_2 \qquad \forall i \in [n].\ \Gamma_2 \vdash z_{2,i} :\ \tau_i\end{array}}{\Gamma \vdash ((x_1 :\ \tau_1, \ldots, x_n :\ \tau_n) = \text{如果}\ y\{S_1;\ \bar{z}_1\}\{S_2;\ \bar{z}_2\}) :\ (\Gamma, x_1 :\ \tau_1, \ldots, x_n :\ \tau_n)}\ \text{T-If}$$

程序 $\boxed{\vdash P : \mathsf{ok}}$

$$\frac{(x_1 : \tau_1, \ldots, x_n : \tau_n) \vdash S : \Gamma \qquad \Gamma \vdash y : \tau'}{\vdash (\mathsf{def}\ f(x_1 : \tau_1, \ldots, x_n : \tau_n) : \tau'\ \{S; \mathsf{return}\ y\}) : \mathsf{ok}}\ \text{T-Prog}$$

图 16. ZivIR 的类型系统。

B.3 操作语义

ZivNorm 的操作语义与 ZivNorm 的操作语义相同Ziv (图 15）。ZivIR 和 C语言$_0$ 的操作语义分别见图 18 和19。

我们使用与附录 A.3中相同的记号，以及以下记号 .

- 对于 $f\colon X \to Y$和 $X' \subseteq X$, $f|_{X'}\colon X' \to Y$是 f对 X'的限制。
- 对于 $op[\bar{\tau} \to \tau'] \in Op_{\text{ZivIR}} \cup Op_{\text{C语言}_0}$，其语义是附录 $[\![op[\bar{\tau} \to \tau']]\!]$ 中定义的A.3 .

定义（类型上下文） $\boxed{\Gamma}$

$$\Gamma ::= \varepsilon \mid \Gamma, x : \tau \mid \Gamma, x : \tau[n]$$

表达式 $\boxed{\Gamma \vdash E : \tau}$

$$\frac{v \in Val_\tau}{\Gamma \vdash v : \tau}\ \text{T-Const} \qquad \frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}\ \text{T-Var} \qquad \frac{\forall i \in [n].\, \Gamma(x_i) = \tau_i}{\Gamma \vdash op[(\tau_1, \ldots, \tau_n) \to \tau'](x_1, \ldots, x_n) : \tau'}\ \text{T-Op}$$

$$\frac{\Gamma(x) = \tau[n] \qquad \Gamma(y) = \mathsf{UInt}_k}{\Gamma \vdash x[y] : \tau}\ \text{T-Get}$$

语句 $\boxed{\Gamma \vdash S : \Gamma'}$

$$\frac{}{\Gamma \vdash \varepsilon : \Gamma}\ \text{T-Skip} \qquad \frac{x \notin \mathrm{dom}(\Gamma)}{\Gamma \vdash (\tau\ x;) : (\Gamma, x : \tau)}\ \text{T-Decl} \qquad \frac{\Gamma \vdash x : \tau \qquad \Gamma \vdash E : \tau}{\Gamma \vdash (x = E;) : \Gamma}\ \text{T-Assn}$$

$$\frac{x \notin \mathrm{dom}(\Gamma) \qquad n \geq 1 \qquad \forall i \in [n].\, v_i \in \text{值}_\tau}{\Gamma \vdash (\tau\ x[n] = \{v_1, \ldots\ldots, v_n\};) : (\Gamma, x : \tau[n])}\ \text{T-New} \qquad \frac{\Gamma \vdash S_1 : \Gamma_1 \qquad \Gamma_1 \vdash S_2 : \Gamma_2}{\Gamma \vdash S_1\ S_2 : \Gamma_2}\ \text{T-Seq}$$

$$\frac{\Gamma \vdash x : \text{布尔型} \qquad \Gamma \vdash S_1 : \Gamma_1 \qquad \Gamma \vdash S_2 : \Gamma_2}{\Gamma \vdash \text{如果 } (x)\ \{S_1\}\ \text{否则 } \{S_2\} : \Gamma}\ \text{T-If}$$

程序 $\boxed{\vdash P : \mathsf{ok}}$

$$\frac{(x_1 : \tau_1, \ldots, x_n : \tau_n) \vdash S : \Gamma \qquad \Gamma \vdash y : \tau'}{\vdash \tau'\ f(\tau_1\ x_1, \ldots, \tau_n\ x_n)\{S\ \mathsf{return}\ y;\} : \mathsf{ok}}\ \text{T-Prog}$$

图 17。C语言$_0$的类型系统。

定义 (状态) $\boxed{\sigma}$

$$\sigma ::= \varepsilon \mid \sigma, x \mapsto v$$

表达式 $\boxed{\sigma \vdash E \Downarrow v}$

$$\frac{v \in \text{值}_\tau}{\sigma \vdash v \Downarrow v}\ \text{E-常量} \qquad \frac{\sigma(x) = v}{\sigma \vdash x \Downarrow v}\ \text{E-变量} \qquad \frac{\forall i \in [n].\ \sigma \vdash x_i \Downarrow v_i \quad [\![\text{操作}[(\tau_1, \ldots, \tau_n) \to \tau']]\!](v_1, \ldots, v_n) = v'}{\sigma \vdash \text{操作}[(\tau_1、\ \ldots\ldots、\ \tau_n) \to \tau'](x_1、\ \ldots\ldots、\ x_n) \Downarrow v'}\ \text{E-Op}$$

$$\frac{\sigma \vdash x \Downarrow \text{值}_{\tau[n]}(v_1, \ldots, v_n) \qquad \sigma \vdash y \Downarrow \text{值}_{\text{UInt}_k}(i) \qquad i \in [n]_0}{\sigma \vdash \text{get}(x,\ y) \Downarrow v_{i+1}}\ \text{E-get}$$

语句 $\boxed{\sigma \vdash S \Downarrow \sigma'}$

$$\frac{}{\sigma \vdash \varepsilon \Downarrow \sigma}\ \text{E-Skip} \qquad \frac{\sigma \vdash E \Downarrow v}{\sigma \vdash (x : \tau = E) \Downarrow (\sigma, x \mapsto v)}\ \text{E-Assn}$$

$$\frac{n \geq 1 \qquad \forall i \in [n].\ v_i \in \text{值}_\tau}{\sigma \vdash (x :\ \ \tau[n] = \text{new}\ (v_1, \ldots, v_n)) \Downarrow (\sigma, x \mapsto \text{值}_{\tau[n]}\ (v_1, \ldots, v_n))}\ \text{E-新建} \qquad \frac{\sigma \vdash S_1 \Downarrow \sigma_1 \quad \sigma_1 \vdash S_2 \Downarrow \sigma_2}{\sigma \vdash S_1; S_2 \Downarrow \sigma_2}\ \text{E-序列}$$

$$\frac{\sigma \vdash y \Downarrow \text{值}_{\text{Bool}}(\text{true}) \qquad \sigma \vdash S_1 \Downarrow \sigma_1 \qquad \forall i \in [n].\ \sigma_1 \vdash z_{1,\ i} \Downarrow v_i}{\sigma \vdash ((x_1 :\ \ \tau_1, \ldots, x_n :\ \ \tau_n) = \text{如果}\ y\{S_1;\ \ \bar{z}_1\}\ S_2;\ \ \bar{z}_2\}) \Downarrow (\sigma,\ \ x_1 \mapsto v_1, \ldots, x_n \mapsto v_n)}\ \text{E-IfTrue}$$

$$\frac{\sigma \vdash y \Downarrow \text{值}_{\text{Bool}}(\text{false}) \qquad \sigma \vdash S_2 \Downarrow \sigma_2 \qquad \forall i \in [n].\ \sigma_2 \vdash z_{2,i} \Downarrow v_i}{\sigma \vdash ((x_1 :\ \ \tau_1, \ldots, x_n :\ \ \tau_n) = \text{如果}\ y\{S_1;\ \ \bar{z}_1\}\{S_2;\ \ \bar{z}_2\}) \Downarrow (\sigma, x_1 \mapsto v_1, \ldots, x_n \mapsto v_n)}\ \text{E-IfFalse}$$

程序 $\boxed{P \vdash f(\bar{v}) \Downarrow v'}$

$$\frac{(x_1 \mapsto v_1, \ldots, x_n \mapsto v_n) \vdash S \Downarrow \sigma \qquad \sigma \vdash y \Downarrow v'}{(\text{def}\ f(x_1 : \tau_1, \ldots, x_n : \tau_n) : \tau'\ \{S; \text{return}\ y\}) \vdash f(v_1, \ldots, v_n) \Downarrow v'}\ \text{E-Prog}$$

图 18. ZivIR 的操作语义。

定义 （状态） $\boxed{\sigma}$

$$\sigma ::= \varepsilon \mid \sigma, x \mapsto v \mid \sigma, x \mapsto \mathsf{undef}_\tau$$

表达式 $\boxed{\sigma \vdash E \Downarrow v}$

$$\frac{v \in \text{值}_\tau}{\sigma \vdash v \Downarrow v}\ \text{E-常量} \qquad \frac{\sigma\ (x)\ = v}{\sigma \vdash x \Downarrow v}\ \text{E-变量} \qquad \frac{\forall i \in [n].\ \sigma \vdash x_i \Downarrow v_i \qquad [\![\text{运算符}[(\tau_1, \ldots, \tau_n) \to \tau']]\!](v_1, \ldots, v_n)\ = v'}{\sigma \vdash \text{操作符}[(\tau_1, \ldots, \tau_n) \to \tau'](x_1, \ldots, x_n)\ \Downarrow v'}\ \text{E-Op}$$

$$\frac{\sigma \vdash x \Downarrow \mathsf{val}_{\tau[n]}(v_1, \ldots, v_n) \qquad \sigma \vdash y \Downarrow \mathsf{val}_{\mathsf{UInt}_k}(i) \qquad i \in [n]_0}{\sigma \vdash x[y] \Downarrow v_{i+1}}\ \text{E-GET}$$

语句 $\boxed{\sigma \vdash S \Downarrow \sigma'}$

$$\frac{}{\sigma \vdash \varepsilon \Downarrow \sigma}\ \text{E-跳过} \qquad \frac{}{\sigma \vdash \tau\ x; \Downarrow (\sigma, x \mapsto \text{未定义}_\tau)}\ \text{E-声明} \qquad \frac{\sigma \vdash E \Downarrow v}{\sigma \vdash x = E; \Downarrow (\sigma,\ \ x \mapsto v)}\ \text{E-赋值}$$

$$\frac{n \geq 1 \qquad \forall i \in [n].\ v_i \in \text{值}_\tau}{\sigma \vdash \tau\ x[n] = \{v_1, \ldots, v_n\}; \Downarrow (\sigma, x \mapsto \text{值}_{\tau[n]}(v_1, \ldots, v_n))}\ \text{E-新建} \qquad \frac{\sigma \vdash S_1 \Downarrow \sigma_1 \qquad \sigma_1 \vdash S_2 \Downarrow \sigma_2}{\sigma \vdash S_1\ S_2 \Downarrow \sigma_2}\ \text{E-序列}$$

$$\frac{\sigma \vdash x \Downarrow \text{值}_{\mathsf{Bool}}(\mathsf{true}) \qquad \sigma \vdash S_1 \Downarrow \sigma_1}{\sigma \vdash \mathsf{if}\ (x)\ \{S_1\}\ \mathsf{else}\ \{S_2\}\ \Downarrow \sigma_1|_{\mathrm{dom}(\sigma)}}\ \text{E-IfTrue} \qquad \frac{\sigma \vdash x \Downarrow \text{值}_{\mathsf{Bool}}(\mathsf{false}) \qquad \sigma \vdash S_2 \Downarrow \sigma_2}{\sigma \vdash \text{如果}\ (x)\ \{S_1\}\ \text{否则}\ \{S_2\}\ \Downarrow \sigma_2|_{\mathrm{dom}(\sigma)}}\ \text{E-IfFalse}$$

程序 $\boxed{P \vdash f(\bar{v})\ \Downarrow v'}$

$$\frac{(x_1 \mapsto v_1, \ldots, x_n \mapsto v_n) \vdash S \Downarrow \sigma \qquad \sigma \vdash y \Downarrow v'}{\tau'\ f(\tau_1\ x_1, \ldots, \tau_n\ x_n)\{S\ \mathsf{return}\ y;\} \vdash f(v_1, \ldots, v_n) \Downarrow v'}\ \text{E-PROG}$$

图 19. C语言的操作语义[0∞]

C语言 Ziv 编译器的完整定义

本节将定义 Ziv 编译器的三个步骤：规范化、追踪和代码生成。我们使用背景色来区分不同的语言：黄色表示 Ziv 和 ZivNorm，蓝色表示 ZivIR，白色表

C.1 规范化（Ziv → ZivNorm）

图 20定义了 Ziv 编译器的规范化步骤。

Th函数 $\mathcal{T}^{y}_{\text{nor}}$ 会在多个情况均适用时选择第一个满足条件的情况。

C.2 追踪（ZivNorm → ZivIR）

Ziv 编译器的追踪步骤定义于图 22和 23中。

我们使用与附录 A和 B中相同的记号。注意，θ 表示 Ziv 中的擦除类型（即，对某个 θ = 类型 T 有 eras（T），中 T 属于 Ziv）

我们定义图 22 和 23中使用的三个辅助函数。

绑定函数。 我们定义绑定函数 $bind(\widehat{v})$，其中 $\widehat{v}$ 是一个追踪值：

$$bind(\widehat{v}) \coloneqq \begin{cases} (\varepsilon, \text{rtvar}(x:\tau)) & \text{if } \widehat{v} = \text{rtvar}(x:\tau) \\ (x:\tau = v, \text{rtvar}(x:\tau)) \text{ for fresh } x & \text{if } \widehat{v} = \text{ctval}(v) \text{ and } v \in Val_\tau \\ \text{undefined} & \text{otherwise.} \end{cases}$$

绑定函数接收一个追踪值 $\widehat{v}$并返回一个二元组 （S，rtvar（x： τ））其中 一个 ZivIR 语句，$\widehat{v}$绑定到 ZivIR 变量 x。

查找函数。 首先，令 θ 为 Ziv 中的一种擦除类型 我们对类型 θ 的追踪值集合进行归纳定义

$$\widehat{Val}_\theta \coloneqq \begin{cases} \{\text{ctval}(v) \mid v \in Val_\theta\} & \text{if } \theta = \tau \wedge \theta \notin \text{ZivIR} \\ \{\text{ctval}(v) \mid v \in Val_\theta\} \cup \{\text{rtvar}(x:\theta) \mid x \in \text{ZivIR}\} & \text{if } \theta = \tau \wedge \theta \in \text{ZivIR} \\ \{\text{ctval}(v) \mid v \in Val_\theta\} \cup \{\langle \widehat{v}_1, \ldots, \widehat{v}_n \rangle_\theta \mid \widehat{v}_i \in \widehat{Val}_{\theta_i}\} & \text{otherwise,} \end{cases}$$

其中，(=字段( ) 是最后一种情形中 θ 的字段类型 有序列表。注意，如果 θ 是函数类型， 值 $_\theta = \varnothing$，因为 值 $_\theta =$ 和 字段(θ) = 未定义。

接下来 令 θ 为 Ziv 中的一种非函数擦除 ，并令 原始字段 θ) = 为构成 θ 的原始类型有序列表；也就是说，原始字段( ) 是递归展平 的字段类型，直到只剩下原始类型后得到的结果。我们 下定义 θ 的解包函数：

$$unwrap_\theta : \widehat{Val}_\theta \to \prod_{j \in [m]} \widehat{Val}_{\tau_j},$$

$$unwrap_\theta(\widehat{v}) \coloneqq \begin{cases} (\widehat{v}) & \text{if } \theta = \tau \\ unwrap_{\theta_1}(\text{ctval}(v_1)) +\!\!+ \cdots +\!\!+ unwrap_{\theta_n}(\text{ctval}(v_n)) & \text{if } \widehat{v} = \text{ctval}(\text{val}_\theta(v_1, \ldots, v_n)) \\ unwrap_{\theta_1}(\widehat{v}_1) +\!\!+ \cdots +\!\!+ unwrap_{\theta_n}(\widehat{v}_n) & \text{if } \widehat{v} = \langle \widehat{v}_1, \ldots, \widehat{v}_n \rangle_\theta, \end{cases}$$

其中，$+\!\!+$ 是列表连接运算符，而在后两种情况下， =字段(解包函数将类型为 θ的追踪值 $\widehat{v}$转换为原始类型 $pfields(\theta)$ 的追踪值 $\widehat{v}_1, \ldots, \widehat{v}_m$。们将 θ的包装函数定义为逆操作。具体来说，我们将其 义为

使其具有以下签名并满足以下性质：

$$wrap_\theta : \prod_{j\in[m]} \widehat{Val}_{\tau_j} \to \widehat{Val}_\theta,$$

$$unwrap_\theta(wrap_\theta(\widehat{v}_1,\dots,\widehat{v}_m)) = (\widehat{v}_1,\dots,\widehat{v}_m) \quad \text{for all } (\widehat{v}_1,\dots,\widehat{v}_m) \in \textstyle\prod_{j\in[m]} \widehat{Val}_{\tau_j}.$$

现在，我们将查找函数 $lookup(\mathrm{val}_\theta(v_1,$ 下，其中 $\theta = \mathrm{Tabl}$ 中用于表的擦除类型 $v_1,\dots,v_n \in$ 示表项 Ziv 而 x 是表示表索引的ZIVIR 量。 $lds(\theta') = (\tau_1,\dots,\tau_m)$ 为构成 θ' 的基 类型 。设 $unwrap_{\theta'}(\mathrm{ctval}(v_i)) = (\mathrm{ctval}(v_{i,1}),\dots,\mathrm{ctval}(v$)) 为表项 v_i 包为基 值 $(i \in [n])$ 结果。那么， 找函数定义如下：

$$lookup(\mathrm{val}_\theta(v_1,\dots,v_n), x) \coloneqq \begin{cases} (S', \widehat{v}') & \text{if } \tau_j \in \text{ZIVIR for all } j \in [m] \\ \text{undefined} & \text{otherwise,} \end{cases}$$

其中， ′ 和 $\widehat{v}'$ 定义如下：

$$S' \coloneqq \begin{array}{ll} y_1 : \tau_1[n] & = \mathsf{new}(v_{1,1},\dots,v_{n,1}) \\ & \dots \\ y_m : \tau_m[n] & = \mathsf{new}(v_{1,m},\dots,v_{n,m}) \\ z_1 : \tau_1 & = \mathsf{get}(y_1, x) \\ & \dots \\ z_m : \tau_m & = \mathsf{get}(y_m, x) \end{array} \quad \text{for fresh } y_j \text{ and } z_j,$$

$$\widehat{v}' \coloneqq wrap_{\theta'}(\mathrm{rtvar}(z_1 : \tau_1),\dots,\mathrm{rtvar}(z_m : \tau_m)).$$

合并函数。设 b 是 字储务 句的条件值的 ZIVIR 变量，而 $(\ , \widehat{\sigma}_k)$ $(k = 1, 2)$ 是一个二元组，包含追踪条件语句的真分支或假分 得到的 ZIVIR 语句和跟踪状态。合并函数 $merge(\ , (\ , \widehat{\sigma}_1), (\ , \widehat{\sigma}_2))$ 定义如下。

首先，令 X 为两个分支末尾都定义的变量集合：

$$X \coloneqq \mathrm{dom}(\widehat{\sigma}_1) \cap \mathrm{dom}(\widehat{\sigma}_2).$$

我们考虑两个不相交的子集 $X_{\text{值}}$和 $X_{\text{类型}}$，它们属于 X，具体取决于两个分支是否为变量 $x \in X$赋予相同的追踪值（或擦除类型）：

$$X^{\mathrm{val}} \coloneqq \{x \in X \mid \widehat{\sigma}_1(x) = \widehat{\sigma}_2(x)\} \eqqcolon \{x_1^{\mathrm{val}},\dots,x_n^{\mathrm{val}}\},$$

$$X^{\mathrm{typ}} \coloneqq \left\{x \in X \setminus X^{\mathrm{val}} \;\middle|\; etype(\widehat{\sigma}_1(x)) = etype(\widehat{\sigma}_2(x)) \wedge \forall \tau \in pfields(etype(\widehat{\sigma}_1(x))).\ \tau \in \text{ZIVIR}\right\}$$
$$\eqqcolon \{x_1^{\mathrm{typ}},\dots,x_m^{\mathrm{typ}}\},$$

其中 x 、……互不相同，并且$etype(\widehat{v})$ 是由以下定义的 $\widehat{v}$的擦除类型

$$etype(\widehat{v}) \coloneqq \begin{cases} \theta & \text{if } \widehat{v} = \mathrm{ctval}(v) \text{ and } v \in Val_\theta \\ \tau & \text{if } \widehat{v} = \mathrm{rtvar}(x : \tau) \\ \theta & \text{if } \widehat{v} = \langle \widehat{v}_1,\dots,\widehat{v}_n \rangle_\theta \\ \text{undefined} & \text{otherwise.} \end{cases}$$

换言之，当且仅当 x 在两个分支结 时存储相同的追踪值， $x \in X^{\mathrm{val}}$ 才成立；而当且仅当 (x 在两个分支结束时 储不同的追踪值，(ii)

x 具有相同的非函数擦除类型 θ 在两个分支结束时，并且 (iii) θ 由 ZIVIR 中的基本类型构成。

现在，我们定义合并函数 $merge(b, (S_1, \widehat{\sigma}_1), (S_2, \widehat{\sigma}_2))$。其高层思路是：(i) 将 X^{val} 中的变量赋为它们的共同值；(ii) 将 X_{typ} 中的变量赋为新值，这些新值通过运行时使用 b 的条件表达式计算得出。令 $\theta_i := etype(\widehat{\sigma}_1(x_i^{\text{typ}}))$ 为 $x_i^{\text{typ}} \in X^{\text{typ}}$ 的共同擦除类型。对于每个 $k \in [2]$ 和 $i \in [m]$，定义

$$\overline{\tau_i} := pfields(\theta_i), \qquad \overline{z_i} := |\overline{\tau_i}| \text{ fresh variables in ZIVIR},$$
$$(\overline{S_{k,i}}, \text{rtvar}(\overline{y_{k,i} : \tau_i})) := bind\big(unwrap_{\theta_i}(\widehat{\sigma}_k(x_i^{\text{typ}}))\big),$$

其中，绑定函数扩展为追踪值列表，使其能够连接计算得到的ZIVIR 语句和追踪值。然后，合并定义如下：

$$merge\big(b, (S_1, \widehat{\sigma}_1), (S_2, \widehat{\sigma}_2)\big) := (S', \widehat{\sigma}'),$$

其中，S' 和 $\widehat{\sigma}'$ 定义为

$$S' := \begin{array}{l}(\overline{z_1} : \overline{\tau_1}, \ldots, \overline{z_m} : \overline{\tau_m}) = \text{if } b\ \{S_1; \overline{S_{1,1}}; \cdots; \overline{S_{1,m}}; (\overline{y_{1,1}}, \ldots, \overline{y_{1,m}})\} \\ \qquad\qquad\qquad\qquad\quad \{S_2; \overline{S_{2,1}}; \cdots; \overline{S_{2,m}}; (\overline{y_{2,1}}, \ldots, \overline{y_{2,m}})\}\end{array},$$
$$\widehat{\sigma}' := \Big(\big(x_i^{\text{val}} \mapsto \widehat{\sigma}_1(x_i^{\text{val}})\big)_{i \in [n]}, \big(x_i^{\text{typ}} \mapsto wrap_{\theta_i}(\text{rtvar}(\overline{z_i} : \overline{\tau_i}))\big)_{i \in [m]}\Big).$$

C.3 代码生成（ZivIR → C语言$_0$）

编译器的代码生成步骤定义于图 ZIV 中，见图 21。

语句

$$\mathcal{T}^{y}_{\text{norm}}(S, S') \coloneqq S; S' \qquad \text{if } \neg ret(S),$$
$$\mathcal{T}^{y}_{\text{norm}}(\text{return } E, S') \coloneqq y = E,$$
$$\mathcal{T}^{y}_{\text{norm}}(S_1; S_2, S') \coloneqq \mathcal{T}^{y}_{\text{norm}}(S_1, \mathcal{T}^{y}_{\text{norm}}(S_2, S')),$$
$$\mathcal{T}^{y}_{\text{norm}}(\text{if } E\ \{S_1\}\ \{S_2\}, S') \coloneqq \text{if } E\ \{\mathcal{T}^{y}_{\text{norm}}(S_1, S')\}\ \{\mathcal{T}^{y}_{\text{norm}}(S_2, S')\}.$$

程序

$$\mathcal{T}_{\text{norm}}(\varepsilon) \coloneqq \varepsilon,$$
$$\mathrm{T}_{\text{norm}}(\quad) \coloneqq$$

其中 是一个新变量。

图 20. 从 Ziv 到 ZivNor 的翻

表达式

$$\mathcal{T}_{\text{emit}}(v) \coloneqq v,$$
$$\mathcal{T}_{\text{emit}}(x) \coloneqq x,$$
$$\mathcal{T}_{\text{emit}}(op[\bar{\tau}\to\tau'](\bar{x})) \coloneqq op[\bar{\tau}\to\tau'](\bar{x}),$$
$$\mathcal{T}_{\text{emit}}(\text{get}(x, y)) \coloneqq x[y].$$

语句

$$\mathrm{T}_{\text{emit}}(\quad) \coloneqq \varepsilon,$$
$$\mathrm{T}_{\text{emit}}(x: \quad (= \tau\ x)) : x = ;\ x = \mathrm{T}_{\text{emit}}(\quad);,$$
$$\mathrm{T}_{\text{emit}}(\quad) \coloneqq \tau\ x[n] = \{\bar{v}\};,$$
$$\mathrm{T}_{\text{emit}}(\quad) \coloneqq \mathrm{T}_{\text{emit}}(\quad)\ \mathrm{T}_{\text{emit}}(\quad),$$
$$\mathrm{T}_{\text{emit}}((\bar{x}: \bar{\tau}) \quad \text{如果 } y\{S: \bar{\tau}\}\{S: \bar{\tau}\}) \coloneqq \bar{\tau}\ \bar{x};\ \text{如果 } (y)\ \{\mathrm{T}_{\text{emit}}(\quad)\ \bar{x} = \bar{z}_1;\}\ \text{否则 } \{\mathrm{T}_{\text{emit}}(\quad)\ \bar{x} = \bar{z}_2;\},$$

其中 $\bar{\tau}\ \bar{x};$ 以及 $\bar{x} = \bar{z}_i;$ 在声明和赋值之间使用分号。

程序

$$\mathrm{T}_{\text{emit}}(\text{定义 } f(\bar{x}: \bar{\tau}): \tau'\{S: \text{返回 } y\}) \coloneqq \tau'\ f(\bar{\tau}\ \bar{x})\{\mathrm{T}_{\text{emit}}(S)\ \text{返回 } y;\}。$$

图21. 从 ZivIR 到 C语言 择。

定义 （追踪值、追踪状态）

$\widehat{v}, \widehat{\sigma}$

$\widehat{v} ::= \mathrm{ctval}(v) \mid \mathrm{rtvar}(x, \langle \widehat{v}_1, \ldots, \widehat{v}_n \rangle)_\theta \qquad \widehat{\sigma} ::= \varepsilon \mid \widehat{\sigma}, \ell \mapsto \widehat{v}$

表达式

$P; \widehat{\sigma} \vdash e \rightsquigarrow (S, \widehat{v})$

$$\frac{}{P; \widehat{\sigma} \vdash v \rightsquigarrow (\varepsilon, \mathrm{ctval}(v))}\ \text{C-常量} \qquad \frac{\widehat{\sigma}(\ell) = \widehat{v}}{P; \widehat{\sigma} \vdash x \rightsquigarrow (\varepsilon, \widehat{v})}\ \text{C-变量}$$

$$\frac{\forall i \in [n].\ e_i; \widehat{\sigma} \vdash e_i \rightsquigarrow (S_i, \mathrm{ctval}(v_i)) \qquad [\![op]\!](v_1, \ldots, v_n) = v' \qquad P; \widehat{\sigma} \vdash op(e_1, \ldots, e_n) \rightsquigarrow (S_1; \cdots; S_n, \mathrm{ctval}(v'))}{}\ \text{常量操作规则}$$

$$\frac{\forall i \in [n].\ e_i; \widehat{\sigma} \vdash e_i \rightsquigarrow (S_i, \widehat{v}_i) \qquad \exists j \in [n].\ \forall \ell.\ \widehat{v}_j \neq \mathrm{ctval}(\ell) \qquad \forall i \in [n].\ \text{绑定}(\widehat{v}_i) = (\ell, \text{运行时变量}(\ldots)) \qquad S' := (S_1; \cdots;\ S_{2n};\ y' :\ \tau' = op[(\tau_1, \ldots, \tau_n) \to \tau'](x_1, \ldots, x_n)) \text{ 对于新鲜的 } \ell \qquad P; \widehat{\sigma} \vdash op(e_1, \ldots, e_n) \rightsquigarrow (\ell, \text{运行时变量}(\ldots, \tau'))}{}\ \text{运行时操作规则}$$

$$\frac{(\text{定义 } f(x_1 : T_1, \ldots, x_n : T_n) : T' \{S; \text{返回 } E\}) \in P \qquad P; (x_i \mapsto \widehat{v}_i)_{i \in [n]} \vdash \langle S, \widehat{\sigma} \rangle \rightsquigarrow (\varepsilon, \widehat{\sigma}') \qquad P; \widehat{\sigma}' \vdash E \rightsquigarrow (\ell, \widehat{v}') \qquad P; \widehat{\sigma} \vdash f(e_1, \ldots, e_n) \rightsquigarrow (\ell, \widehat{v}')}{}\ \text{C语言调用}$$

$$\frac{\forall i \in [n].\ e_i; \widehat{\sigma} \vdash e_i \rightsquigarrow (S_i, \mathrm{ctval}(v_i))}{P; \widehat{\sigma} \vdash \ldots \rightsquigarrow (\ell, \mathrm{ctval}\ldots)}\ \text{C语言-NewCt}$$

$$\frac{\forall i \in [n].\ e_i; \widehat{\sigma} \vdash e_i \rightsquigarrow (\ell, \widehat{v}_i) \qquad \exists j \in [n].\ \forall \ell.\ \widehat{v}_j \neq \mathrm{ctval}(\ell) \qquad P; \widehat{\sigma} \vdash \ldots \rightsquigarrow (\ldots, \langle \widehat{v}_1, \ldots, \widehat{v}_n \rangle \text{ 擦除}(\ell))}{}\ \text{C语言-new}$$

$$\frac{P; \widehat{\sigma} \vdash e \rightsquigarrow (\ell, \mathrm{ctval}(\ldots)) \qquad P; \widehat{\sigma} \vdash e \rightsquigarrow (\ell, \mathrm{ctval}(\ldots)) \qquad i \in [n]_0 \qquad P; \widehat{\sigma} \vdash \ldots \rightsquigarrow (S, \mathrm{ctval}\ldots)}{}\ \text{C语言-GetCtInd1}$$

$$\frac{P; \widehat{\sigma} \vdash e \rightsquigarrow (\ell, \langle \widehat{v}_1, \ldots, \widehat{v}_n \rangle) \qquad P; \widehat{\sigma} \vdash e \rightsquigarrow (\ell, \mathrm{ctval}\ldots)) \qquad i \in [n]_0 \qquad P; \widehat{\sigma} \vdash \ldots \rightsquigarrow (S_1; \ldots, \widehat{v}_{i+1})}{}\ \text{C-GetCtInd2}$$

$$\frac{P; \widehat{\sigma} \vdash e \rightsquigarrow (\ell, \mathrm{ctval}\ldots)) \text{ 适…} \qquad P; \widehat{\sigma} \vdash (\ell \rightsquigarrow \text{运行时变量}(x : \ldots)) \qquad \text{查找}(\text{值} \ldots) = (\ell, \widehat{v}) \qquad P; \widehat{\sigma} \vdash \ldots (\ldots; \ldots, \widehat{v})}{}\ \text{C-GetRtInd}$$

图 22. 从 ZivNorm 到 Zi…式的…。

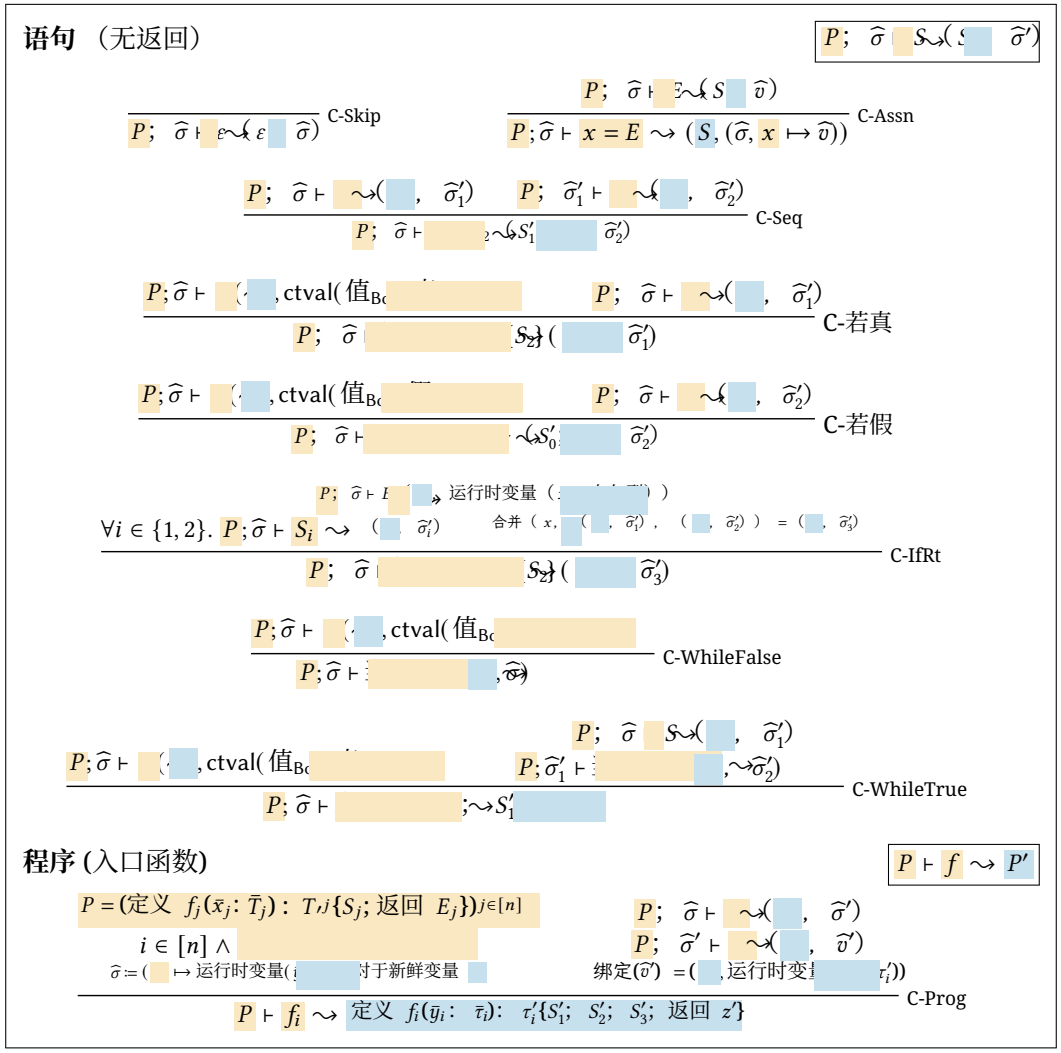


图 23. 从 ZivNorm 到 Zi 们与 翻译。

D 语言的可靠性

本节将给出 Ziv、ZivIR 和 C_0 的类型保持结果。所有结果均已在 Lean 4.35.0 中形式化，相关形式化内容收录于补充材料中。下文中的所有文件路径均相对于补充材料中的 lean/Ziv/ 目录。

D.1 Ziv

定理 D.1（类型保持；定理 3.1）。设 P 是一个类型正确的 Ziv 程序（即，$\vdash P : \mathrm{ok}$）。假设函数 f 的类型为 $(T_1, \ldots, T_n) \to T'$，属于 P（即，$f : (T_1, \ldots, T_n) \to T' \in P$）。则，对于所有值 $v_1, \ldots, v_n, v'$，

$$(\forall i \in [n].\ P; \varepsilon \vdash v_i : T_i) \quad \wedge \quad P \vdash f(v_1, \ldots, v_n) \Downarrow v' \quad \Longrightarrow \quad P; \varepsilon \vdash v' : T'.$$

证明。参见 Theorems.lean 中的 LangZiv.type_preservation。证明见 Proofs/LangZiv/TypePreservation.lean。□

D.2 ZivIR

定理 D.2（类型保持）。设 $P =$ 定义 $f(x_1 : \tau_1, \ldots, x_n : \tau_n) : \tau' \{ s; \text{返回 } y; \}$ 是一个类型良好的ZivIR程序（$\vdash P :$ 良好）。那么，对于所有值 $v_1, \ldots, v_n, v'$，

$$(\forall i \in [n].\ \varepsilon \vdash v_i : \tau_i) \quad \wedge \quad P \vdash f(v_1, \ldots, v_n) \Downarrow v' \quad \Longrightarrow \quad \varepsilon \vdash v' : \tau'.$$

证明。参见 Theorems.lean 中的 LangIR.type_preservation。证明见 Proofs/LangIR/TypePreservation.lean。□

D.3 C语言$_0$

定理 D.3（类型保持）。设 $P = \tau'\ f(x_1 : \tau_1, \ldots, x_n : \tau_n)\{S; \text{返回 } y; \}$ 是一个类型良好的C语言$_0$程序（即，$\vdash P :$ 良好）。那么，对于所有值 $v_1, \ldots, v_n, v'$，

$$(\forall i \in [n].\ \varepsilon \vdash v_i : \tau_i) \quad \wedge \quad P \vdash f(v_1, \ldots, v_n) \Downarrow v' \quad \Longrightarrow \quad \varepsilon \vdash v' : \tau'.$$

证明。参见 Theorems.lean 中的 LangC0.type_ 保持性。证明位于 Proofs/LangC0/TypePreservation.lean 中。□

E 编译器的可靠性

本节将介绍每个编译器步骤以及整个编译器的可靠性结果。所有结果均使用 Lean 4.35.0 形式化，形式化内容包含在补充材料中。以下所有文件路径均相对于补充材料中的 lean/Ziv/ 目录。

E.1 规范化（Ziv → ZivNorm）

定理 E.1（类型保持）。设 P 是一个 - P : 并且 = $\mathcal{T}_{\text{norm}}$ P）是对 P进行规范化后的 果。那 ЯM 程序：

$$\vdash P' : \text{ok}.$$

证明。参见 Theorems.lean 中的 CompNorm.type_preservation。证明见 Proofs/CompNorm/TypePreservation.lean。□

定理 *E.2*（语义保持）。设 P、P' 如定理 *E.1* 述，且 是一个类型为 $(T_1, \ldots, T_n) \to T'$ 的函数，其 : (... , 值 $v_1, \ldots, v_n$，若 v' 的类型均为 （即 $P; \varepsilon \vdash v_i : T_i$），则 , $_1, \ldots, v_n)$ 在 P'

$$P \vdash f(v_1, \ldots, v_n) \Downarrow v' \iff P' \vdash f(v_1, \ldots, v_n) \Downarrow v'.$$

证明：参见 Theorems.lean 中的 CompNorm.semantic_preservation。证明位于 Proofs/CompNorm/SemanticPreservation.lean 中。□

E.2 追踪（ZivNorm → ZivIR）

定理 E.3（类型保持）。令 P 为一个类型 ，并 定义于 P 中的函数。假设对 进 追踪会生成 。即 ⊢ 。那么，Q 是一个类型良好

$$\vdash Q : \text{ok}.$$

证明：参见 Theorems.lean 中的 CompTrace.type_preservation。证明位于 Proofs/CompTrace/TypePreservation.lean 中。□

定理 *E.4*（语义保持）。令 P、f 和 如定理 中所述，其中 在 P 中的类型为 $(T_1, \ldots, T_n) \to T$ $_1, \ldots, T_n) \to$ $\ldots, v_n$，v'，其中每个 v_i 的类型均为 （即，$P; \varepsilon \vdash v_i : T_i$），在 $\ldots, v_n)$ 在 Q 中

$$P \vdash f(v_1, \ldots, v_n) \Downarrow v' \iff Q \vdash f(v_1, \ldots, v_n) \Downarrow v'.$$

证明。参见CompTrace.semantic_preservation，见 Theorems.lean。证明位于 Proofs/CompTrace/SemanticPreservation.lean。□

类型兼容性。考虑一个 ZivNorm 程序 P。对于值和 态 $\widehat{v}$ 与 $\widehat{\sigma}$ 在追踪中，以及类型和类型上下文 T 与 Γ 于 ZivNorm中，我们定义关系 $\widehat{v} \vDash_P$ 与 $\widehat{\sigma} \vDash_P$ ，用以表示 $\widehat{v}$ 与 $\widehat{\sigma}$ 分别与 T 和 Γ 在 P 中相兼容：

$$\widehat{v} \vDash_P T \iff \begin{cases} \left(\widehat{v} = \text{ctval}(v) \wedge P; \varepsilon \vdash v : T\right) & \text{if } T = \overline{T}' \to T'' \\ \left(\widehat{v} = \text{ctval}(v) \wedge P; \varepsilon \vdash v : T\right) \vee \left(\widehat{v} = \text{rtvar}(x : \tau) \wedge \ell = \top\right) & \text{if } T = \tau^{\ell} \\ \left(\widehat{v} = \text{ctval}(v) \wedge P; \varepsilon \vdash v : T\right) & \text{otherwise} \\ \quad \vee \left(\widehat{v} = \langle \widehat{v}_1, \ldots, \widehat{v}_n \rangle_{erase(T)} \wedge fields(T) = (T_1, \ldots, T_n) \wedge \forall i \in [n].\ \widehat{v}_i \vDash_P T_i\right), \end{cases}$$

$\widehat{\sigma} \vDash_P \iff$ (定义域($\widehat{\sigma}$) ⊇定义域(Γ)) ∧ (∀ ∈定义域(Γ). $\widehat{\sigma}$(x) $\vDash_P \Gamma(x)$).

定理 E.5（类型兼容性）。设 P 是一个类[illegible]）。假[illegible]式 E 的类[illegible]Γ）。[illegible]与 Γ 兼容的跟踪状态（即 $\widehat{\sigma} \models_P$ [illegible]），[illegible]果在 $\widehat{\sigma}$ 下对 E 进行追踪[illegible]功，则所得值与 T 兼容：

$$P; \widehat{\sigma} \vdash E \rightsquigarrow (S, \widehat{v}) \quad \Longrightarrow \quad \widehat{v} \models_P T.$$

证明。参见 Theorems.lean 中的 CompTrace.type_compatibility。证明见 Proofs/CompTrace/TypeCompatibility.lean。□

定理 E.6（$\bot \Rightarrow$ ctval；定理 5.1）。设 P 是一[illegible]：ok[illegible]表达式 E[illegible]。对[illegible]容的追踪状态 Γ（即 $\widehat{\sigma} \models_P$ [illegible]），如果在 [illegible]下追踪 $\widehat{\sigma}$ 成功，则所得的[illegible]为 ctval：

$$P; \widehat{\sigma} \vdash E \rightsquigarrow (S, \widehat{v}) \quad \Longrightarrow \quad \exists v.\, \widehat{v} = \mathrm{ctval}(v) \wedge P; \varepsilon \vdash v : \tau^{\bot}.$$

证明。由定理 E.5 以及 $\models_P$ 的定义可得。参见 `CompTrace.ctval_的_bottom`，见 `Theorems.lean`。证明见 `Proofs/CompTrace/TypeCompatibility.lean`。□

E.3 代码生成（ZivIR $\to$ C语言$_0$）

定理 E.7（类型保持）。设 Q 是一个类[illegible]），且[illegible]$_t$([illegible])是由 [illegible]代码生成得到的结果。那么，R 是一个类型正确的C语言$_0$ 程序：

$$\vdash R : \mathrm{ok}.$$

证明。参见 Theorems.lean 中的 CompEmit.type_preservation。证明见 Proofs/Comp Emit/TypePreservation.lean。□

定理 *E.8*（语义保持）。设 [illegible]和 R 与定理 *E.7* 中相同，其中 $Q = def\ f(x_1 : \tau_1, \ldots,$ [illegible]$_1, \ldots, v_n, v'$，其中每个 v_i 的类型均为 τ_i（即$\varepsilon \vdash v_i : \tau_i$），[illegible]当且[illegible]$, .$ [illegible]$, \ldots, v_n)$ 在 Q 中求值为 v'：

$$Q \vdash f(v_1, \ldots, v_n) \Downarrow v' \quad \Longleftrightarrow \quad R \vdash f(v_1, \ldots, v_n) \Downarrow v'.$$

证明：参见 Theorems.lean 中的 CompEmit.semantic_preservation。证明位于 Proofs/CompEmit/SemanticPreservation.lean 中。□

E.4 完整编译器（Ziv $\to$ C$_0$）

定理 *E.9*（编译器可靠性；定理 *5.2*）。设 P 是一个类型良好[illegible] *Ziv* 程序（即，$\vdash P$：*ok*），其中包含[illegible]为（T_1，…，T_n）$\to T'$ 的函数 [illegible]（即[illegible]$_1$，…，[illegible]生成 R，即 T_{norm}（P）$\vdash$ [illegible]且 T_{emit}（[illegible]）$\vdash R$。那么，R 是一个类型良好的 *C*语言$_0$ 程序（即，$\vdash R$：*ok*），并且对于所有值 v_1、…、v_n、v'，其中每个 v_i 的类型均为 [illegible]（即，P；$\varepsilon \vdash v_i$：T_i），[illegible]

$$P \vdash f(v_1, \ldots, v_n) \Downarrow v' \quad \Longleftrightarrow \quad R \vdash f(v_1, \ldots, v_n) \Downarrow v'.$$

证明。 这可由定理立即得出 E.1–E.4、E.7 和E.8。参见 `CompAll.soundness`，位于 `Theorems.lean`中。证明见 `Proofs/CompAll/Soundness.lean`。 □