arXiv:2601.12813v1 [cs.PL] 19 Jan 2026

FIRRTL 中宽度推断的形式化验证过程*

Keyin Wang[1, 2, 3][0009−0004−1954−5171], Xiaomu Shi[1, 2, 3][0000−0001−6277−2813], Jiaxiang Liu[1, 2, 3][0000−0002−6725−8167], Zhilin Wu[1, 2, 3][0000−0003−0899−628X], FuSong[1, 2, 3, 4][0000−0002−0581−2679], Taolue Chen[5][0000−0002−5993−1665], 与 David N. Jansen[1, 2, 3][0000−0002−6636−3301]

[1] 系统软件重点实验室（中国科学院）[2] 中国科学院软件研究所[3] 中国科学院大学[4] 南京软件技术研究院[5] 伦敦大学伯贝克学院
{wangky,shixm,liujx,wuzl,songfu,dnjansen}@ios.ac.cn
t.chen@bbk.ac.uk

摘要。 FIRRTL 是一种用于寄存器传输级（RTL）硬件设计的中间表示语言。在 FIRRTL 程序中，许多组件的位宽并未显式指定，必须在编译过程中推断。在主流 FIRRTL 编译器（如官方编译器 firtool）中，位宽推断由名为 InferWidths 的编译遍执行，即使是简单的 FIRRTL 程序也可能导致该遍失败。本文深入研究 FIRRTL 程序的位宽推断问题。我们证明，如果从 FIRRTL 程序中得到的约束可满足，则存在唯一的最小解。基于这一结果，我们提出了一种求解位宽推断问题的完整过程。我们在交互式定理证明器 Rocq 中实现了该过程，并证明了其函数正确性。我们从 Rocq 实现中提取出 OCaml 实现，这是首个经过形式化验证的 InferWidths 遍实现。大量实验表明，我们的方法通常能以较高效率求解比 firtool 官方 InferWidths 遍更多的实例。

1 引言

Chisel（在 Scala 嵌入式语言中构造硬件 [2]）是一种开源硬件描述语言，旨在描述寄存器传输级（RTL）的数字电子学和电路。作为一种嵌入式领域专用语言，Chisel 由 Scala 驱动，将面向对象编程和函数式编程的强大能力引入类型安全的硬件设计与生成。它

* 本研究得到中国科学院战略性先导科技专项（项目编号：XDA0320101）的资助，并部分得到 NSFC-RGC 合作研究项目（项目编号：62561160151）的资助。D.N. Jansen 得到北京市自然科学基金项目（项目编号：IS25071）的资助。

还包含一套特殊类定义、预定义对象和使用约定库。Chisel 中的高层硬件生成器通过使用 Chisel 标准库中的电路组件以及 Scala 函数进行编程，并运用函数式和面向对象编程特性；其接口由 Scala 类型编码。这使得敏捷方法得以应用于高度参数化、模块化且可复用的硬件生成器，从而提升硬件设计的生产力和稳健性 [3]。Chisel 已成功用于开发 Rocket Chip [53], RISC-V BOOM [52], NutShell [48],和香山 [64,63]等 RISC-V 处理器。

FIRRTL（RTL 灵活中间表示 [23,57]）最初被提出作为一种用于编译 Chisel 程序的中间表示（IR）语言，其作用类似于 LLVM IR [38]在将 C/C++ 程序编译为低级代码时所发挥的作用。它原生支持向量类型、捆绑类型、条件语句、连接和模块等高级结构，因此可以轻松地将 Chisel 程序转换为 FIRRTL。FIRRTL 上的高层次综合通过一系列编译和优化遍来完成，最终得到一种限制最严格、接近可综合 RTL Verilog 的形式，通常称为低级 *FIRRTL*（简称 LoFIRRTL）。为清楚起见，我们将包含高级结构的 FIRRTL 称为高级 *FIRRTL*（简称 HiFIRRTL）。FIRRTL 最初虽是为编译 Chisel 程序而设计，但并不局限于 Chisel。其他硬件描述语言也可以编译为 FIRRTL，并复用 FIRRTL 编译器中的大部分编译和优化遍。例如，FIRRTL 已被集成到 CIRCT 项目 [13] 中，成为其核心方言之一。作为 CIRCT 项目的一部分，工具 fir-tool 是当前官方的 FIRRTL 编译器，并广泛用于 RISC-V 处理器设计。

FIRRTL 中的位宽推断。使用 FIRRTL 时，建议用户手动指定端口和寄存器的（位）宽度。未指定的宽度会在编译过程中推断得出。理想情况下，推断出的宽度应尽可能小，同时取决于所有输入连接的宽度。

考虑图 1 中的 FIRRTL 程序，该程序改编自 Chisel 手册 [55]。模块 `CombWhen` 有两个输入 `clock` 与 `cond`、一个输出`out`和一条导线 `w`。 `cond` 与 `out` 的宽度分别为 1 和 4，而 `w` 的宽度未指定。尽管 FIRRTL 采用“最后连接语义”，即将最近连接的表达式的值赋给 `w`，但位宽推断必须考虑所有连接，以确保它们合法。因此， `w` 的宽度必须不小于无符号整数`<1>("h0")` 和 无符号整数`<2>("h3")`的宽度，也就是至少为 2。当前官方 FIRRTL 编译器 firtool 中实现的 InferWidths 编译遍，能够正确推断出 w 的这一宽度。

这个例子相当直观，因为（电路）组件之间不存在循环依赖。请注意，如果存在连接语句`x <= exp` ，且组件 y 出现在表达式 exp 中，那么组件 x 就依赖于组件 y。组件之间的循环依赖会使位宽推断问题更具挑战性。例如，对于图 2 中的程序

```
1电路 CombWhen：2 模块
Co
输
ou
符
"h(
数
```

图 1：含有未指定宽度的 FIRRTL 程序

```
1电路 A：2  模块 A：3 输入 in：无符号
整
ou
数
1)
```

图 2：含有未指定宽度且存在循环依赖的 FIRRTL 程序

其中，x 和 out 的宽度未指定，且 x 依赖于自身，从而产生了（也许是最简单的）循环依赖。在这种情况下，firtool 的 InferWidths 遍无法推断 x 的宽度，并会抛出异常。

直观而言，图 2 中程序的宽度推断可以归结为求约束的最小解

$$w_{\mathtt{x}} \geq \max(w_{\mathtt{x}} - 1, 4) + 1 \ \wedge w_{\mathtt{out}} \geq w_{\mathtt{x}},$$

其中，$w_{\mathtt{x}}$ 和 $w_{\mathtt{out}}$ 分别是表示 x 和 out 位宽的非负整数变量。（根据 FIRRTL 规范，表达式低位截取`(e1, n)` 和加法`(e1, e2)` 的位宽分别为 $w_{\mathtt{e1}} - \mathrm{n}$ 和 $\max(w_{\mathtt{e1}}, w_{\mathtt{e2}}) + 1$。）不难看出，该约束存在最小解，即 $w_{\mathtt{x}} = w_{\mathtt{out}} = 5$。

这个例子表明，firtool 中 InferWidths 遍的当前实现并不完整。完整实现 InferWidths 遍至关重要：如果推断出的位宽过大，综合硬件就会浪费面积和功耗，并且可能因布线和时序复杂度增加而导致性能下降。反之，如果推断出的位宽过小，则可能发生算术溢出或数据丢失，导致结果不正确。此外，位宽推断发生在编译流水线的早期阶段；此阶段未检测到的错误可能会传播到后续遍，并很可能破坏功能保证。因此，面对循环依赖问题，有必要对位宽推断进行系统性研究。例如，唯一最小解是否存在，先验地并不明确。

我们的长期目标是开发一个经过形式化验证的 FIRRTL 编译器。本研究朝着这一目标迈出了重要一步。我们对 FIRRTL 中的位宽推断问题进行了形式化，探究了其理论性质，提出了解决该问题的完整过程，并开发了一个经过全面验证的 InferWidths 遍。

贡献。 本研究的主要贡献总结如下。

– 我们将 FIRRTL 语言的位宽推断问题形式化为求解 **FIR**RTL**W**idth **INE**quality（FIRWINE）约束的问题。特别地，我们证明了如果一个 FIRWINE 约束是可满足的，那么它具有唯一的最小解。

– 我们提出了一个完整过程，用于判定 FIRWINE 约束的可满足性，并计算可满足约束的最小解。– 我们在交互式定理证明器 Rocq[6] 中实现了该过程，并证明了其函数正确性。– 我们从 Rocq 实现中提取出 OCaml 实现，得到首个经过形式化验证的 InferWidths 遍。在包括三个真实世界 RISC-V 处理器 Chisel 设计（即 Rocket Chip [53], 、RISC-V BOOM [52], 和 NutShell [48]）在内的大量基准测试上进行的广泛实验表明，我们的 OCaml 实现能够比 firtool 中的 InferWidths 遍解决更多位宽推断实例，并且总体上具有更高的效率。

我们指出，位宽推断问题可以归约为整数线性规划（整数线性规划），并由整数线性规划求解器（例如 Gurobi [21]）求解。然而，目前通用的整数线性规划求解算法过于复杂，无法进行形式化验证；而我们的方法是专门设计的，便于形式化验证，这是迈向形式化验证的 FIRRTL 编译器的一项里程碑。

虽然本文仅针对 FIRRTL 中的问题，但其他硬件描述语言中也存在类似问题。例如，Verilog 和 VHDL 允许开发者实现具有可配置位宽的参数化模块，因此需要检查这些模块之间互连的一致性。我们的方法也可应用于此类场景，其中一致性检查可归约为位宽约束的可满足性检查 [54]。特别是，在判定过程中使用依赖图解决循环依赖，与具体语言无关，因此具有广泛的适用性。

概述。 第2节介绍 FIRRTL，对位宽推断问题进行形式化，并证明对于任意可满足的 FIRRTL 位宽不等式约束，始终存在唯一最小解。第3节介绍我们的完整过程。第4节描述 Rocq 实现及其形式化验证。在第5节中，我们提取 OCaml 实现并全面评估其性能。最后，我们在第6节讨论相关工作，并在第7节作出结论。

2 FIRRTL 与位宽推断问题

FIRRTL 电路 由模块列表构成，其中指定的顶层模块与电路具有相同的标识符。每个模块都表示一个可供实例化的硬件模块：顶层模块总是会被实例化，而其他模块则按需实例化。例如，图 2 所示的 FIRRTL 电路`A` 包含一个（顶层）模块`A`。（FIRRTL 语言的形式语法见附录 A。）

模块由一系列端口声明和语句组成。每个端口都有一个方向（输入或 输出）、一个标识符和一个数据类型。例如，图 2 中的模块 `A` 有两个输入端口，即 `in` 和`clock`，以及

[6] https://rocq-prover.org，此前称为 Coq。

一个输出端口，即 `out`，其类型分别为 无符号整数<4>、时钟 和无符号整数。每条语句定义一条导线、一个寄存器、一个节点、一个连接（<=）、一个条件语句（when）等。用于指定数据结构的类型可以是基元类型，例如 无符号整数<4>（位宽为 4 的无符号整数）或 时钟 （承载时钟信号）；也可以是聚合类型，包括向量类型，例如 无符号整数[3]（由三个无符号整数组成的向量），或束类型，例如 {enable: 无符号整数<1>, calc: 无符号整数<3>}，其中包含两个信号，其基元类型分别为 无符号整数<1> 和 无符号整数<3>。

（电路）组件是 FIRRTL 电路中声明的元素，可通过名称引用；它可以是端口、节点、导线、寄存器、存储器或模块实例。组件可以出现在连接语句中。连接必须遵循数据流规则，例如，输入端口和节点只能被读取。

2.1 FIRRTL 中的位宽推断问题

FIRRTL 提供了向量类型、束类型、条件语句、聚合连接和模块等高级结构，随后会通过一系列降级转换逐步移除这些结构。一个必要步骤是为所有位宽未指定的组件推断最小（位）宽度，同时确保所有输入连接均合法，如图 1 和图 2 所示。如果无法做到这一点，则应抛出异常，指明某个位宽无法推断。例如，若 x 是一根导线，且有两条连接语句：`x <= 无符号整数<5>(15)` 和 `x <= 无符号整数<1>(1)`，则推断 x 的位宽为 5，其中 `无符号整数<n>(m)` 表示位宽为 n 位的无符号整数 m。相反，若导线 x 通过以下语句连接：`x <= 加法(x, 无符号整数<1>(1))`，则 x 不存在可行的位宽，因为不等式 $w_x \geq \max(w_x + 1, 1)$ 不可满足。因此会抛出异常。

在本文中，令 $X = \{x_1, \ldots, x_n\}$ 表示一个非空变量集合，其变量取值范围为非负整数集 $\mathbb{N}$。此外，对于 $m \geq 1$，令 $[m]$ 表示集合 $\{1, \ldots, m\}$。形式化地说，对于给定的 FIRRTL 程序，位宽推断问题 是求解由以下不等式构成的集合的最小（非负）解：对每条连接语句 $w_x \geq w_e$，`x <= e`，其中 w_x是一个非负整数变量，表示 x未指定的位宽；w_e 表示表达式 e 的位宽，其定义遵循 FIRRTL 规范 [57]（形式化定义见附录 B）。例如，$w_{\texttt{mux(a,b,c)}} = \max(w_b, w_c)$ 和 $w_{\texttt{add(a,b)}} = \max(w_a, w_b) + 1$。根据 [57, 第 25] 节，FIRRTL 中的每个表达式 e w_e 都是符合以下语法的项

$$t \stackrel{\text{def}}{=} w_x \mid c \mid \min(t, t) \mid \max(t, t) \mid t + t,$$

其中，w_x 是一个非负整数变量，表示组件 x 未指定的位宽，且 $c \in \mathbb{Z}$。请注意，术语 $\min(w_x, w_y)$ 是为表达式 `rem(x, y)`（即 x 除以 y 的余数）引入的。此处我们跳过 FIRRTL 中的动态左移 操作，该操作会引入形如 2^{w_e} 的项。我们的方法可以扩展以处理这些项，不过具体细节留待未来工作。虽然我们不支持最一般形式的动态左移 动态左移，但我们基准测试中所有动态左移的出现形式都属于两种特定形式，可由我们的方法处理（参见第5.2节）。

如果形如 $w_{\mathsf{x}} \geq w_{\mathsf{e}}$ 的不等式的合取可满足，则称该位宽推断问题可满足 。位宽推断问题的解 是一个满足所有不等式的变量赋值。

2.2 唯一最小解的存在性

由于解空间没有自然的全序，只有当一组不等式可满足且存在唯一最小解时，位宽推断问题才有意义。我们将证明确实如此。（如果存在多个极小解，则生成的硬件将无法充分确定。）

位宽项由以下规则定义

$$t \stackrel{\text{def}}{=} x_j \mid c \mid t + t$$

使得 $x_j \in X$ 和 $c \in \mathbb{Z}$。注意，在位宽项中，不允许对变量进行减法。因此，我们可以假设每个位宽项的形式均为$a_0 + \sum_{j\in[n]} a_j x_j$ 使得 $a_0 \in \mathbb{Z}$ 且 $a_j \in \mathbb{N}$ 对于每个 $j \in [n]$。

定义 1（Φ_W-约束）。一个***FIRRTL WidthINEquality***（*FIRRTL* 位宽不等式）约束（简称 Φ_W-约束）定义在 X 上，是由以下形式的不等式构成的合取： $x_i \geq \min(t_{i,1}, \ldots, t_{i,k_i})$。每个不等式称为关于 $x_i \in X$ 的不等式，其中 $t_{i,1}, \ldots, t_{i,k_i}$ 是定义在 X 上的位宽项。

Φ_W约束 φ 的一个解是函数 η： $X \to \mathbb{N}$，使得 η下 φ 中的每个不等式都成立。为简洁起见，我们将解 η写作（$\eta(x_1)$.....， $\eta(x_n)$）。令 $\preceq_n$ 表示 $\mathbb{N}^n$ 上的偏序，使得对于每个 $i \in [n]$，若 $u_i \leq v_i$，则（u_1，.....， u_n） $\preceq_n$（v_1，.....， v_n）。如果对于 φ的每个解（v_1，.....， v_n），都有（u_1，.....， u_n） $\preceq_n$（v_1，.....， v_n），则称（u_1，.....， u_n）为 Φ_W约束 φ的最小解。

位宽推断问题中的每个不等式 $w_{\mathsf{x}} \geq w_{\mathsf{e}}$ 都可以改写为无 max 不等式的合取，因为最小值和加法对最大值（以及最小值）满足分配律，并且 $w_{\mathsf{x}} \geq \max(w_{\mathsf{a}}, w_{\mathsf{b}})$等价于 $w_{\mathsf{x}} \geq w_{\mathsf{a}} \wedge w_{\mathsf{x}} \geq w_{\mathsf{b}}$。形式化地，

命题 1. 位宽推断问题中表述的每个不等式 $w_{\mathsf{x}} \geq w_{\mathsf{e}}$ 都可以转换为等价的 Φ_W-约束。

例 *1.* 考虑 Φ_W-约束 $\varphi \equiv x_1 \geq 2x_2 - 3 \wedge x_2 \geq 2x_1 + 1.$。容易看出 $x_1 \geq 2x_2 - 3 \geq 2(2x_1 + 1) - 3 = 4x_1 - 1$。因此，我们可以推导出 $1 \geq 3x_1$，进而得到 $x_1 = 0$。此外，根据 $0 \geq 2x_2 - 3$ 和 $x_2 \geq 1$，我们可以得出 $x_2 = 1$。由此可知，(0,1) 是 φ 的唯一解，因此也是 φ 的最小解。

另一方面，如果 $\varphi \equiv x_1 \geq 2x_1 \wedge x_1 \geq 1$，那么 $0 \geq x_1 \geq 1$，这意味着φ不可满足。
□

命题 2. 可满足的 Φ_W-约束有唯一最小解。

证明. 设 φ 是一个可满足的 Φ_W-约束。只需证明:

如果 $\eta_1 = (u_1, \ldots, u_n)$ 和 $\eta_2 = (v_1, \ldots, v_n)$ 是 φ 的解，那么 $\eta = (\min(u_1, v_1) \ldots, \min(u_n, v_n))$ 也是 φ 的解。

下文中，我们用 $\eta(x_i)$ 表示 $\min(u_i, v_i)$，类似地，$\eta_1(x_i) = u_i, \eta_2(x_i) = v_i$；并将这些函数扩展到位宽项。令 I 表示满足 $u_i \le v_i$ 的索引集合 $i \in [n]$。那么，对于每个 $i \in [n] \setminus I$，都有 $u_i > v_i$。因此，对于每个 $i \in [n]$,

$$\eta(x_i) = \min(u_i, v_i) = \begin{cases} u_i, & \text{if } i \in I; \\ v_i, & \text{if } i \in [n] \setminus I. \end{cases}$$

对于某个 $i \in [n]$，设 $x_i \ge \min(t_{i,1}, \ldots, t_{i,k_i})$ 是 Φ_W-约束 φ 中变量 x_i 的一个不等式，其中对每个 $j \in [k_i]$，都有 $t_{i,j} = a_{i,j,0} + \sum_{l \in [n]} a_{i,j,l} x_l$。我们将证明 $\eta(x_i) \ge \min(\eta(t_{i,1}) \ldots, \eta(t_{i,k_i}))$。由此可直接得出，$\eta$ 是 φ 的一个解。

我们先来看 $u_i \le v_i$ 这一情况，即 $i \in I$。于是

$$\begin{aligned}
\eta(x_i) = u_i &\overset{*}{\ge} \min(\eta_1(t_{i,1}), \ldots, \eta_1(t_{i,k_i})) \\
&= \min\left\{ a_{i,j,0} + \sum_{l \in I} a_{i,j,l} \eta_1(x_l) + \sum_{l \in [n] \setminus I} a_{i,j,l} \eta_1(x_l) \;\middle|\; j \in [k_i] \right\} \\
&= \min\left\{ a_{i,j,0} + \sum_{l \in I} a_{i,j,l} u_l + \sum_{l \in [n] \setminus I} a_{i,j,l} u_l \;\middle|\; j \in [k_i] \right\} \\
&\overset{**}{\ge} \min\left\{ a_{i,j,0} + \sum_{l \in I} a_{i,j,l} u_l + \sum_{l \in [n] \setminus I} a_{i,j,l} v_l \;\middle|\; j \in [k_i] \right\} \\
&= \min\left\{ a_{i,j,0} + \sum_{l \in I} a_{i,j,l} \eta(x_l) + \sum_{l \in [n] \setminus I} a_{i,j,l} \eta(x_l) \;\middle|\; j \in [k_i] \right\} \\
&= \min(\eta(t_{i,1}), \ldots, \eta(t_{i,k_i})).
\end{aligned}$$

$*$上面的第一个不等式 $\ge$ 成立，是因为 η_1 是 φ 的一个解；而第二个 $**$不等式 $\ge$ 成立，是因为 $a_{i,j,l} \ge 0$，并且对于 $l \in [n] \setminus I$，有 $u_l > v_l$。

剩下的情况，即 $u_i > v_i$，可用类似的方法证明，只需交换集合 I 和 $[n] \setminus I$。□

3 位宽推断的完整过程

注意到 $x_i \ge \min(t_1, \ldots\ldots, t_k)$ 等价于 $\bigvee_{j \in [k]} x_i \ge t_j$，因此可以将 Φ_W-约束改写为 $\Phi_{\mathrm{nf}} := \bigvee_{j_1 \in [\ell]} \bigwedge_{j_2 \in [m_{j_1}]} z_{j_1,j_2} \ge t_{j_1,j_2}$ 使得Φ_{nf} 可满足，当且仅当对于某个 $j_1 \in [\ell]$，$\bigwedge_{j_2 \in [m_{j_1}]} z_{j_1,j_2} \ge t_{j_1,j_2}$ 可满足，其中$z_{j_1,j_2} \in X$，且 t_{j_1,j_2} 是一个位宽项。此外，Φ_{nf} 的最小解是各析取项最小解中的最小者。因此，只需考虑形如 $x_i \ge a_0 + \sum_{j \in [n]} a_j x_j$ 的不等式合取。

该过程利用了依赖图这一概念。

定义 2（依赖图）。令 φ 为关于 X 的 Φ_W-约束。假设所有变量 x_i 的不等式总数至多为 $L \in \mathbb{N}$。这使得能够为 x_i 的每个不等式分配唯一标签 $l \in [L]$。

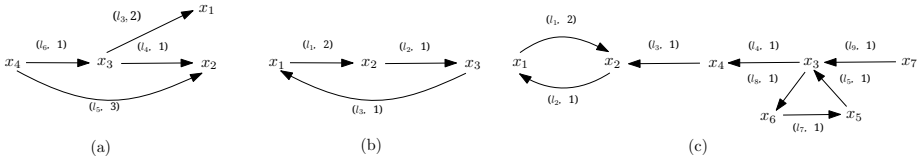

图 3: φ_1、φ_2、φ_3的依赖图

φ 的依赖图 G_φ 被定义为有向图 (V_φ, E_φ)，其中 $V_\varphi = X$，而 $E_\varphi \subseteq V_\varphi \times [L] \times \mathbb{N} \times V_\varphi$ 包含满足以下条件的边 (x_i, l, a_j, x_j): 包含不等式：$x_i \geq a_0 + \sum_{j\in[n]} a_j x_j$，且满足 $a_j > 0$。

例 *2.* 给定带标签的 Φ_W-约束 φ_1、φ_2 与 φ_3:

$$\begin{aligned}
\varphi_1 \equiv\ & l_1 : x_1 \geq 5 \quad \wedge \quad l_2 : x_2 \geq 2 \quad \wedge \quad l_3 : x_3 \geq 2x_1 - 3 \quad \wedge \\
& l_4 : x_3 \geq x_2 + 2 \quad \wedge \quad l_5 : x_4 \geq 3x_2 \quad \wedge \quad l_6 : x_4 \geq x_3 + 4, \\
\varphi_2 \equiv\ & l_1 : x_1 \geq 2x_2 - 4 \quad \wedge \quad l_2 : x_2 \geq x_3 - 2 \quad \wedge \quad l_3 : x_3 \geq x_1 + 1, \\
\varphi_3 \equiv\ & l_1 : x_1 \geq 2x_2 - 4 \quad \wedge \quad l_2 : x_2 \geq x_1 + 1 \quad \wedge \quad l_3 : x_4 \geq x_2 \quad \wedge \\
& l_4 : x_3 \geq x_4 \quad \wedge \quad l_5 : x_5 \geq x_3 + 1 \quad \wedge \quad l_6 : x_5 \geq 2 \quad \wedge \\
& l_7 : x_6 \geq x_5 - 1 \quad \wedge \quad l_8 : x_3 \geq x_6 \quad \wedge \quad l_9 : x_7 \geq x_3.
\end{aligned}$$

这些 Φ_W 约束的依赖图分别为 G_{φ_1}、G_{φ_2} 和 G_{φ_3}，如图 3(a)–(c) 所示。显然，G_{φ_1} 是无环的，G_{φ_2} 是强连通的，而 G_{φ_3} 由四个强连通分量组成，其中两个是非平凡的（至少包含一条边）。□

3.1 整体过程

令 $\mathcal{S}_{G_\varphi}$ 为 G_φ 的强连通分量图。我们首先对 $\mathcal{S}_{G_\varphi}$ 进行拓扑排序，得到 C_1、.....、C_k；也就是说，从 C_i 中的顶点到 C_j 中的顶点的任意一条边都意味着 $i \leq j$。我们用 $V(C)$ 表示 C 中的变量集合。例如，在图 3(c) 中，我们有 $V(C_1) = \{x_7\}$、$V(C_2) = \{x_3, x_5, x_6\}$、$V(C_3) = \{x_4\}$ 和 $V(C_4) = \{x_1, x_2\}$。随后，我们从 $j = k$ 到 $j = 1$ 迭代执行以下过程:

假设已计算出每个 $x_i \in V(C_{j+1}) \cup \cdots \cup V(C_k)$ 的最小宽度 $u_i \in \mathbb{N}$。我们首先将 $x_{i'} \in V(C_j)$ 的不等式中每个 $x_i \in V(C_{j+1}) \cup \cdots \cup V(C_k)$ 都替换为最小宽度 u_i，从而得到关于 $x_{i'} \in V(C_j)$ 的不等式合取 ψ_{C_j}。

– 如果 C_j 是平凡的，即仅包含 $x_{i'}$，那么我们将 $u_{i'}$ 设为 0 与 ψ_{C_j} 的不等式 $x_{i'} \geq a$ 中常数 a 的最大值。–否则，C_j 是强连通的，我们计算 ψ_{C_j} 的最小解 $(u_{i'})_{x_{i'} \in C_j}$；或者 ψ_{C_j} 不可满足，此时过程终止。

现在我们将重点讨论强连通依赖图，中最小解的计算，这是我们过程的核心。我们首先通过扩张性识别出一类强连通依赖图。

3.2 强连通依赖图的刻画

“扩张性”这一概念旨在为一类依赖图提供结构化刻画。如果依赖图 G_φ 包含一条边 $(x,\ l,\ a,\ x')$，满足 $a > 1$，或者从某个顶点引出两条具有相同标签的不同边，则称其为扩张的；否则称其为非扩张。扩张依赖图具有以下性质。

命题 3. 假设 G_φ 是强连通的，且 φ 可满足。G_φ 是扩张的，当且仅当存在常数 $B \in \mathbb{N}$，使得 φ 的每个解中变量的值都有上界 B。

证明。注意，G_φ 中的任意一条边 $(z_{r-1},\ l_r,\ a_r,\ z_r)$ 都源自一个不等式 $l_r : z_{r-1} \geq a_{r,0} + \sum_{z \in X} a_{r,z} z$，其中 $a_{r,z_r} = a_r > 0$。我们可以将这个不等式弱化为 $z_{r-1} \geq a_{r,0} + a_r z_r$，并将这个弱化不等式记作 $\mathsf{Ineq}(z_{r-1},\ l_r,\ a_r,\ z_r)$。类似的弱化不等式可以通过归纳法定义在路径上，具体如下。给定从 z_0 到 z_0的空路径 $\pi_{z_0 \to z_0}$，令

$$\mathsf{Ineq}(\pi_{z_0 \to z_0}) \equiv z_0 \geq z_0.$$

对于从 z_0 经由z_{m-1} 到 z_m 的非空路径 $\pi_{z_0 \to z_m} = \pi'_{z_0 \to z_{m-1}} \cdot (z_{m-1}、\ l_m、\ a_m、\ z_m)$ 以及不等式

$$\begin{aligned} \mathsf{Ineq}(\pi'_{z_0 \to z_{m-1}}) &\equiv \quad z_0 \geq a_{m-1} + b_{m-1} z_{m-1} \\ \mathsf{Ineq}(z_{m-1}, l_m, a_m, z_m) &\equiv z_{m-1} \geq a_{m,0} + a_m z_m, \end{aligned}$$

let

$$\begin{aligned} \mathsf{Ineq}(\pi_{z_0 \to z_m}) \equiv \quad & z_0 \geq a_{m-1} + b_{m-1}(a_{m,0} + a_m z_m) \\ & = a_{m-1} + b_{m-1} a_{m,0} + b_{m-1} a_m z_m. \end{aligned}$$

φ 的任意解对于 G_φ中的所有路径 π 均满足此类 $\mathsf{Ineq}(\pi)$。

– 如果 G_φ 是扩张的，则解有上界：假设 G_φ 是扩张的。我们证明，任意变量 x_k都存在一个上界。存在两种情况：

G_φ包含一条边（$x_i,\ l,\ a,\ x_j$），其 $a > 1$：由于 G_φ是强连通的，因此存在从 x_k 到 x_i 的路径 π_{ki}，以及从 x_j 到 x_k的路径π_{jk}。复合路径 $\mathsf{Ineq}(\pi_{ki} \cdot (x_i,\ l,\ a,\ x_j) \cdot \pi_{jk})$ 的不等式形式为$x_k \geq a_{k,0} + a_{kk} x_k$。注意，由于 $a_{kk} > 1$ 是 $a > 1$的倍数，我们必有 $a_{kk} > 1$。这意味着 φ 的任何解都满足$x_k \leq -a_{k,0}/(a_{kk} - 1)$。$G_\varphi$包含两条不同的边（$x_i,\ l,\ 1,\ x_{j_1}$）和（$x_i,\ l,\ 1,\ x_{j_2}$）：我们可以假设 G_φ 中的所有边（$x',\ l',\ a',\ x''$）都具有 $a' = 1$；否则结论可由第一种情况立即得到。由这两条边，我们可以对不等式 l进行类似的弱化，即$x_i \geq a_{i,0} + x_{j_1} + x_{j_2}$。存在以下路径及其不等式：

$$\begin{aligned}
&\pi_{ki} \text{ from } x_k \text{ to } x_i && \mathsf{Ineq}(\pi_{ki}) \equiv\ x_k \geq a_{ki} + x_i \\
&\pi_{j_1k} \text{ from } x_{j_1} \text{ to } x_k && \mathsf{Ineq}(\pi_{j_1k}) \equiv x_{j_1} \geq a_{j_1k} + x_k \\
&\pi_{j_2k} \text{ from } x_{j_2} \text{ to } x_k && \mathsf{Ineq}(\pi_{j_2k}) \equiv x_{j_2} \geq a_{j_2k} + x_k.
\end{aligned}$$

将所有这些结合起来，我们得到

$$x_k \geq a_{ki} + x_i \geq a_{ki} + a_{i,0} + x_{j_1} + x_{j_2} \geq a_{ki} + a_{i,0} + a_{j_1k} + x_k + a_{j_2k} + x_k.$$

这意味着 φ 的任意解都满足 $x_k \leq -a_{ki} - a_{i,\,0} - a_{j_1k} - a_{j_2k}$。**– 如果 G_φ 是非扩张的，那么这些解没有上界：**假设 G_φ 不包含从某个顶点 x_i 出发的、标签相同的两条不同边，也不包含任何满足 $a > 1$ 的边 $(x_i,\ l,\ a,\ x_j)$。因此，φ 中的所有不等式都具有 $x_i \geq a'_{ij} + x_j$ 或 $x_i \geq a''_i$ 的形式，其中 a'_{ij}, $a''_i \in \mathbb{Z}$ 是常数。

由于 φ 可满足，令 $(u_1,\ u_2,\ \ldots,\ u_n)$ 为 φ 的一个解。显然，对于任意 $C \in \mathbb{N}$，$(u_1 + C,\ u_2 + C,\ \ldots,\ u_n + C)$ 也是 φ 的一个解。因此，φ 的解中的变量取值没有上界。□

因此，如果 G_φ 是扩张的，那么 φ 有限多个解，并且可以通过穷举搜索计算出最小解。因此，我们的过程根据 G_φ 是否扩张，分为不同情况处理（第 3.3 节或第 3.4 节）。

3.3 计　　算扩张　　的 G_φ 的最小解

在这种情况下，我们首先介绍一种计算上界的算法（基本遵循命题 3 的证明），然后说明如何利用这些上界，基于分支定界算法搜索最小解。

计算变量的上界。该算法区分扩张性概念的两种不同情况。

– 若存在一条边（x_i, l, a, x_j），且满足 $a > 1$，则我们通过以下过程计算上界。1. 找出一条从 x_j 到 x_i 的路径，例如，$\pi_{ji} = (y_0, l_1, a_1, y_1) \cdots (y_{r-1}, l_r, a_r, y_r)$（其中 $y_0 = x_j$ 与 $y_r = x_i$）。2. 按照命题 3 的证明，计算 $\mathsf{Ineq}((x_i, l, a, x_j) \cdot \pi_{ji})$，例如，$x_i \geq a_{i,0} + a_{ii} x_i$。3. 根据 $a > 1$，我们知道 $a_{ii} > 1$。因此，$x_i \leq \lfloor -a_{i,0}/(a_{ii} - 1) \rfloor$。由此我们得到了 x_i 的一个上界。4. 我们从 x_i 开始，应用广度优先搜索（BFS）来计算其他变量的上界。也就是说，假设我们知道 $x_{i'} \leq c$，且（$x_{i'}, l', a', x_{j'}$）是 G_φ 中的一条边，令 l'：$x_{i'} \geq b_0 + \sum_{k \in [n]} b_k x_k$（其中 $b_{j'} = a'$），则有 $a' x_{j'} \leq x_{i'} - b_0 \leq c - b_0$，因此 $x_{j'} \leq \lfloor (c - b_0)/a' \rfloor$。（由于 G_φ 是强连通的，应用广度优先搜索即可得到 G_φ 中所有变量的上界。）

– 若某个顶点 x_i 有两条不同的同标签出边，例如，$e_1 = (x_i, l, a_1, x_{j_1})$ 和 $e_2 = (x_i, l, a_2, x_{j_2})$，则我们通过以下过程计算上界。假设 G_φ 中的所有边的权重均为 1。否则，我们已经可以通过上述过程计算上界。

1. 分别寻找从 x_{j_1} 和 x_{j_2} 到 x_i 的两条简单路径，分别记为 π_1 和 π_2。然后，如同命题 3 的证明中那样，我们可以利用 Ineq($e_1 \cdot \pi_1$) 和 Ineq($e_2 \cdot \pi_2$) 计算 x_i 的一个上界，记为 $x_i \le B_i$。2. 与 G_φ 中存在一条权重大于 1 的边的情况类似，我们从 x_i 出发，应用广度优先搜索来计算 G_φ 中所有其他变量的上界。

例 *3.* 考虑例 2 中的 φ_2。如图 3(b) 所示的依赖图 G_{φ_2} 是强连通的，并且包含一条权重大于 1 的边 $(x_1, l_1, 2, x_2)$。然后我们按如下方式计算上界。

由于 $x_1 \ge 2x_2 - 4$，我们找到一条从 x_2 到 x_1 的简单路径，记为 $(x_2, l_2, 1, x_3) \cdot (x_3, l_3, 1, x_1)$，并利用该路径应用替换。于是我们得到

$$x_1 \ge 2x_2 - 4 \ge 2(x_3 - 2) - 4 = 2x_3 - 8 \ge 2(x_1 + 1) - 8 = 2x_1 - 6.$$

因此，$x_1 \le 6$。

最后，我们从 x_1 开始，并应用广度优先搜索来计算 x_2 和 x_3 的上界。首先，根据 $x_1 \ge 2x_2 - 4$，我们有 $2x_2 \le x_1 + 4 \le 6 + 4 = 10$，因此 $x_2 \le 5$。此外，根据 $x_2 \ge x_3 - 2$，我们有 $x_3 \le x_2 + 2 \le 7$。

总而言之，我们得到 (x_1, x_2, x_3) 的上界 $(6, 5, 7)$。

分支定界。令 $(ub_1, \ldots\ldots, ub_n)$ 表示 X 中各变量的上界。对于每个 $i \in [n]$，令 lb_i 表示 0 与 φ 的不等式 $x_i \ge a_0 + \sum_{j\in[n]} a_j x_j$ 中常数 a_0 的最大值。如果对于某个 $i \in [n]$，$lb_i > ub_i$，则报告 φ 不可满足，并终止算法。否则，算法迭代以下分支定界过程，其中上界 $(ub_1, \ldots\ldots, ub_n)$ 和下界 $(lb_1, \ldots\ldots, lb_n)$ 作为参数。（过程示意图见图 4。）

对于情形 $(ub_1, \ldots, ub_n) = (lb_1, \ldots, lb_n)$，我们检查 $(lb_1, \ldots, lb_n)$ 是否是 φ 的解（即，$\varphi[lb_1/x_1, \ldots, lb_n/x_n]$ 是否为真）。如果为真，则返回 $(lb_1, \ldots, lb_n)$ 作为最小解；否则，报告 φ 不可满足。

否则，$(ub_1, \ldots, ub_n) \ne (lb_1, \ldots, lb_n)$，然后我们计算 $(v_1, \ldots, v_n) = (\lfloor (ub_1 + lb_1)/2 \rfloor, \ldots, \lfloor (ub_n + lb_n)/2 \rfloor)$，并检查 $(v_1, \ldots, v_n)$ 是否为 φ 的解。如果它是一个解，则我们设置 $(ub'_1, \ldots, ub'_n) := (v_1, \ldots, v_n)$，并尝试求出相对于 $(ub'_1, \ldots, ub'_n)$ 和 $(lb_1, \ldots, lb_n)$ 的 φ 的最小解。否则，$(v_1, \ldots, v_n)$ 违反某个不等式 l：$x_i \ge a_0 + \sum_{j\in[n]} a_j x_j$，即 $v_i < a_0 + \sum_{j\in[n]} a_j v_j$。在这种情况下，必定存在 $j_0 \in [n]$，使得 $v_{j_0} > 0$ 且 $a_{j_0} > 0$，因为否则 $v_i < a_0$，这与 $v_i \ge lb_i \ge a_0$ 矛盾。我们通过分类讨论继续进行。

– There exists $j_0 \in [n]$ such that $v_{j_0} > 0$, $a_{j_0} > 0$ and $lb_{j_0} < v_{j_0}$. In this case, we try to satisfy the inequality by decreasing the right-hand side $a_0 +$

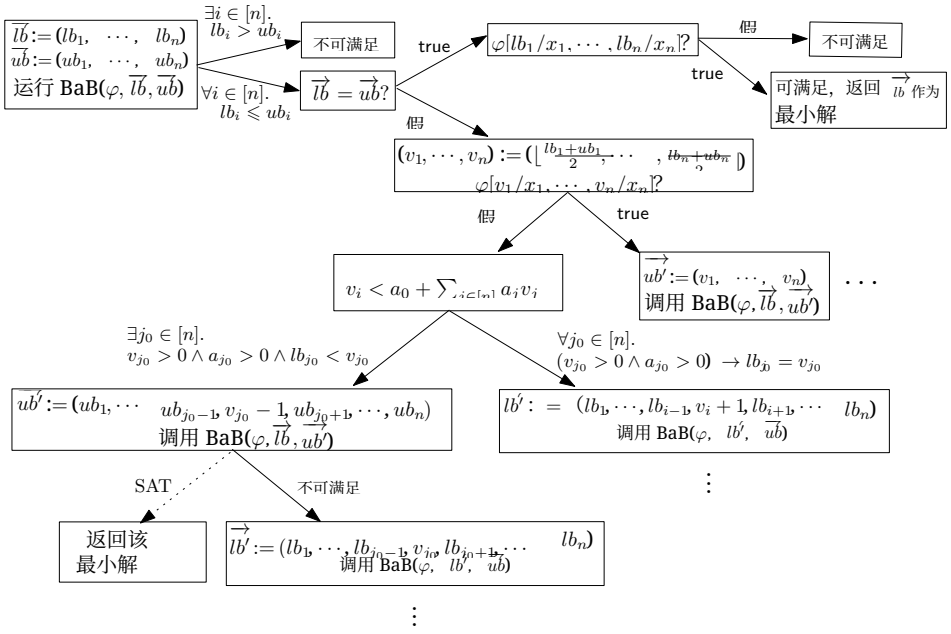

图 4：分支定界（BaB）过程示意图

$\sum_{j\in[n]} a_j x_j$，具体而言，通过减小 x_{j_0}。为此，我们设置 $ub'_{j_0} := v_{j_0} - 1$，并尝试求出相对于 $(ub_1,\ \ldots,\ ub_{j_0-1},\ ub'_{j_0},\ ub_{j_0+1},\ \ldots,\ ub_n)$ 和 $(lb_1,\ \ldots,\ lb_n)$ 的 φ 的最小解。

- 如果尝试成功，我们就计算出了相对于相同下界的 φ 最小解，并将其作为所需的最小解返回。
- 如果尝试失败，仍可能存在满足 $x_{j_0} \geq v_{j_0}$ 的解。因此，令 $lb'_{j_0} := v_{j_0}$，并返回 φ 相对于 $(ub_1, \ldots, ub_n)$ 和 $(lb_1, \ldots, lb_{j_0-1}, lb'_{j_0}, lb_{j_0+1}, \ldots, lb_n)$ 的最小解。如果 φ 相对于这些界不可满足，则报告 φ 不可满足。

– 对于每个满足 $v_{j_0} > 0$ 和 $a_{j_0} > 0$ 的 $j_0 \in [n]$，$lb_{j_0} = v_{j_0}$。在这种情况下，不可能减小 l 的右端：$x_i \geq a_0 + \sum_{j\in[n]} a_j x_j$，因此满足该不等式的唯一方法是增大 x_i。于是我们令 $lb'_i := v_i + 1$，并计算 φ 相对于 $(ub_1, \ldots, ub_n)$ 和 $(lb_1, \ldots, lb_{i-1}, lb'_i, lb_{i+1}, \ldots, lb_n)$ 的最小解。如果 φ 不可满足，则报告 φ 不可满足；否则，将最小解作为所需的最小解返回。

示例 *4*。考虑 Φ_W-约束

$$\varphi \equiv x_1 \geq 2x_2 - 4 \quad \wedge \quad x_2 \geq x_3 - 2 \quad \wedge \quad x_3 \geq x_1 + 1.$$

我们的过程得出变量的上界（$ub_1,\ ub_2,\ ub_3$）：=（6，6，8）。下界从 φ 中提取，分别为（$lb_1,\ lb_2,\ lb_3$）：=（0，0，1），

即不等式中的常数。随后，我们的过程使用上界（ub_1，ub_2，ub_3）和下界（lb_1，lb_2，lb_3）计算最小解，如下所示。

1. 首先，由于 $lb_i \leq ub_i$ for $1 \leq i \leq 3$ but $(lb_1, lb_2, lb_3) \neq (ub_1, ub_2, ub_3)$，我们根据当前上界和下界计算 (v_1, v_2, v_3):

$$(v_1, v_2, v_3) := (\lfloor (lb_1 + ub_1)/2 \rfloor, \lfloor (lb_2 + ub_2)/2 \rfloor, \lfloor (lb_3 + ub_3)/2 \rfloor) = (3, 3, 4).$$

由于 $3 \geq 2 \times 3 - 4 \wedge 3 \geq 4 - 2 \wedge 4 \geq 3 + 1$，$(v_1, v_2, v_3)$ 是 φ 的一个解。因此，我们将 $(ub'_1, ub'_2, ub'_3) := (v_1, v_2, v_3) = (3, 3, 4)$，并使用更新后的上界 (ub'_1, ub'_2, ub'_3) 和先前的下界 (lb_1, lb_2, lb_3) 计算最小解。

2. 使用更新后的上界 (ub'_1, ub'_2, ub'_3)，对于 $lb_i \leq ub'_i$，我们有 $1 \leq i \leq 3$，但 $(lb_1, lb_2, lb_3) \neq (ub'_1, ub'_2, ub'_3)$。因此，我们根据更新后的上界和先前的下界计算新的赋值 (v'_1, v'_2, v'_3):

$$(v'_1, v'_2, v'_3) := (\lfloor (lb_1 + ub'_1)/2 \rfloor, \lfloor (lb_2 + ub'_2)/2 \rfloor, \lfloor (lb_3 + ub'_3)/2 \rfloor) = (1, 1, 2),$$

它也满足 φ as $1 \geq 2 \times 1 - 4 \wedge 1 \geq 2 - 2 \wedge 2 \geq 1 + 1$。然后，上界更新为 $(ub''_1, ub''_2, ub''_3) := (v'_1, v'_2, v'_3) = (1, 1, 2)$，据此重复该过程以计算最小解。

3. 同样，由于 $lb_i \leq ub''_i$for $1 \leq i \leq 3$，但 $(lb_1, lb_2, lb_3) \neq (ub''_1, ub''_2, ub''_3)$，我们计算赋值

$$(v''_1, v''_2, v''_3) := (\lfloor (lb_1 + ub''_1)/2 \rfloor, \lfloor (lb_2 + ub''_2)/2 \rfloor, \lfloor (lb_3 + ub''_3)/2 \rfloor) = (0, 0, 1).$$

由于 (v''_1, v''_2, v''_3) 是 φ 的一个解，因此再次更新上界，即 $(ub'''_1, ub'''_2, ub'''_3) := (v''_1, v''_2, v''_3) = (0, 0, 1)$，并据此重复该过程。

4. 此时，我们有 $(lb_1, lb_2, lb_3) = (ub'''_1, ub'''_2, ub'''_3) = (0, 0, 1)$。此外，$(0, 0, 1)$ 是 φ 的一个解。因此，该过程返回 $(0, 0, 1)$ 作为最小解，并终止。□

3.4 计算非扩张 G_φ 的最小解

在这种情况下，G_φ 中的每条边 (x_i, l, a, x_j) 都满足 $a = 1$，并且从同一顶点出发的所有边的标签两两不同。因此，φ 中的每个不等式都具有 $x_i \geq c_{ij} + x_j$ 或 $x_i \geq c'_i$ 的形式，其中 $c_{ij}, c'_i \in \mathbb{Z}$。

我们构造另一个图 G'_φ，其中包含对应于不等式 $x_i \geq c_{ij} + x_j$ 的边 (x_i, c_{ij}, x_j)。求解 φ 的最小解，可归结为确定 G'_φ 中顶点之间的最大距离（即路径的最大权重）。

直观上，情况如下。对于每个 $i \in [n]$，如果 $x_i \geq c_{ij} + x_j$ 与 $x_j \geq c_{jk} + x_k$，由于 x_i, x_j, x_k 的取值范围为 $\mathbb{N}$，我们可以推导出 $x_i \geq c_{ij} + x_j \geq c_{ij}$与 $x_i \geq c_{ij} + x_j \geq c_{ij} + c_{jk} + x_k \geq c_{ij} + c_{jk}$。一般来说，对于每条从 x_i 开始的路径，例如 $(y_1, l_1, a_1, y_2) \ldots (y_r, l_r, a_r, y_{r+1})$（其中 $y_1 = x_i$），我们有 $x_i \geq$

max($a_1, a_1 + a_2, \ldots, \sum_{j\in[r]} a_j$)。更具体地说，最小解的计算方式如下。对于每个 $i \in [n]$，我们将 v_i 定义为 0 与不等式 $x_i \geq c'_i$ 中的常数 c'_i 的最大值。然后，最小解中 x_i 的值是 v_i 与这些数值 $w_{ij} + v_j$ 的最大值，其中 $j \in [n]$，且 w_{ij} 是 G'_φ 中从 x_i 到 x_j 的最大距离。

为了计算 G'_φ中顶点之间的最大距离，我们将用于计算顶点之间最小距离的著名 Floyd–Warshall 算法 [20,61]改编为计算最大距离的算法。此外，在计算过程中，如果发现了权重为正的环，则报告 φ 不可满足。我们将改编后的算法称为最大 *Floyd–Warshall* 算法。

示例 *5*。考虑

$$\varphi \equiv l_1 : x_1 \geq x_2 + 1 \quad \wedge \quad l_2 : x_1 \geq 2 \quad \wedge \quad l_3 : x_3 \geq x_1 - 1 \quad \wedge \quad l_4 : x_2 \geq x_3.$$

那么，G_φ是一个包含三条边的环，即（x_1，l_1，1，x_2）、（x_2，l_4，1，x_3）和（x_3，l_3，1，x_1）。显然，G_φ是非扩张的。为了解决位宽推断问题，我们构造了 G'_φ，其包含边（x_1，1，x_2）、（x_2，0，x_3）和（x_3，-1，x_1）。然后，我们使用最大 Floyd–Warshall 算法计算以下矩阵 W，该矩阵表示顶点之间的最大距离，

$$W = \begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix}.$$

由于 $w_{11} = w_{22} = w_{33} = 0$，也就是说不存在权重为正的环，因此 φ 是可满足的。因为 $x_1 \geq 2$，我们定义 (v_1, v_2, v_3) = (2,0,0)。因此，最小解为 (u_1, u_2, u_3) = (2, 1, 1)，因为

$$\begin{aligned} &- \ u_1 = \text{最大值}\{v_1,\ w_{12} + v_2,\ w_{13} + v_3\} = \text{最大值}\{2,\quad 1 + 0, 1 + 0\} = 2,- \\ &\quad u_2 = \text{最大值}\{v_2,\ w_{21} + v_1,\ w_{23} + v_3\} = \text{最大值}\{0,\ -1 + 2, 0 + 0\} = 1,- \\ &\quad u_3 = \text{最大值}\{v_3,\ w_{31} + v_1,\ w_{32} + v_2\} = \text{最大值}\{0,\ -1 + 2, 0 + 0\} = 1. \end{aligned}$$

□

3.5 一个完整的示例

我们给出一个完整示例，以展示解决位宽推断问题的整个过程。

例 *6.* 考虑例 2 中的 φ_3。如图 3(c) 所示，其依赖图 G_{φ_3} 包含四个强连通分量，分别为 $V(C_1) = \{x_7\}$、$V(C_2) = \{x_3, x_5, x_6\}$、$V(C_3) = \{x_4\}$ 与 $V(C_4) = \{x_1, x_2\}$。显然，$C_1 :: C_2 :: C_3 :: C_4$ 是 G_{φ_3}的强连通分量图的唯一拓扑排序。接下来，我们计算最小解。

1. 首先，由于 C_4（顶点为 x_1，x_2）包含一条边（x_1，l_1，2，x_2），我们计算 x_1 和 x_2的上界，得到 (2，3)。(x_1，x_2) 的下界为 (0，1)。接下来，我们应用分支定界算法来

计算最小解。具体而言，我们考虑 $((2+0)/2,\ (3+1)/2) = (1,\ 2)$。由于 $(1,\ 2)$ 满足 $x_1 \geq 2x_2 - 4$ 与 $x_2 \geq x_1 + 1$，我们将上界收紧为 $(1,\ 2)$。然后，我们根据上界 $(1,\ 2)$ 和下界 $(0,\ 1)$，计算 φ 的最小解。由于 $(0,\ 1)$ 同时满足 $x_1 \geq 2x_2 - 4$ 和 $x_2 \geq x_1 + 1$，我们将上界收紧为 $(0,\ 1)$。随后，由于下界与上界相等，我们返回 $(0,\ 1)$ 作为最小解。因此，$(u_1,\ u_2) = (0,\ 1)$。

2. 对于 C_3（顶点为 x_4），我们将 x_2 中的 $x_4 \geq x_2$ 替换为 $u_2 = 1$，得到 $x_4 \geq 1$。然后设定 $u_4 = 1$。
3. 对于 C_2（顶点为 $x_3,\ x_5,\ x_6$），我们将 x_4 中的 $x_3 \geq x_4$ 替换为 1，得到 $x_3 \geq 1$。此外，我们有关于 $x_3,\ x_5,\ x_6$ 的以下不等式：$x_5 \geq x_3 + 1$、$x_5 \geq 2$、$x_6 \geq x_5 - 1$ 与 $x_3 \geq x_6$。由于 C_3 是非扩张的，我们构造图 G'_{C_3}，它由边 $(x_5,\ 1,\ x_3)$、$(x_6,\ -1,\ x_5)$ 和 $(x_3,\ 0,\ x_6)$ 组成。然后，我们使用最大 Floyd–Warshall 算法计算顶点之间的最大距离，得到如下矩阵（顶点顺序为 $x_3,\ x_5,\ x_6$），

$$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}.$$

此外，我们计算 $(v_3,\ v_5,\ v_6) = (1,\ 2,\ 0)$。因此，$(u_3,\ u_5,\ u_6) = (\max(1,\ -1+1,\ 0+1),\ \max(2,\ 1+1,\ 1+0),\ \max(0,\ 0+1,\ -1+2)) = (1,\ 2,\ 1)$。4. 最后，对于 C_1 (带有顶点 x_7)，我们将 x_3 in $x_7 \geq x_3$ 替换为 u_3 并得到$x_7 \geq 1$。因此，$u_7 = 1$。

最终，我们得到最小解 $(0,\ 1,\ 1,\ 1,\ 2,\ 1,\ 1)$。 □

3.6 时间复杂度

我们的过程的总体最坏情况时间复杂度相对于依赖图的规模呈指数级，即相对于电路组件的数量呈指数级（由扩张强连通分量的分支定界主导）。各阶段的时间复杂度如下。

– 拓扑排序。我们使用 Tarjan 算法，其运行时间与依赖图的规模呈线性关系，即 $O(|V| + |E|)$.– 非扩张强连通分量。对于每个非扩张强连通分量，我们应用时间复杂度为 $O(n^3)$ 的最大 Floyd–Warshall 算法，其中 n 是该强连通分量中的节点数。– 扩张强连通分量。对于每个扩张强连通分量，我们应用分支定界（BaB）过程。首先，在多项式时间内计算命题 3 中的上界 B，计算时间取决于该强连通分量的规模。随后通过分支定界，在大小为 B^n 的空间中搜索解；最坏情况下（例如必须探索所有分支时），搜索时间关于 n 呈指数级。

然而，在实际应用中，我们的实验遇到的大多数强连通分量要么很小，要么不是扩张的。因此，我们的过程在实际应用中表现良好。

4 Rocq 实现与形式化验证

本节中，我们使用 Rocq 对第3节中的位宽推断过程进行形式化，规定其函数正确性，并说明如何证明 Rocq 实现满足正确性要求。形式化内容共包含 5,301 行 Rocq 代码；规范和证明共包含 6,992 行 Rocq 代码。

4.1 位宽推断过程的形式化

令 φ 为一个 Φ_W 约束。我们在 Rocq 中将 φ 表示为一个映射 cm，其中对于每个变量 x，$cm(x)$ 是一组关于 x 的不等式。我们引入一种数据结构来表示依赖图 G_φ，并在其上应用 Mathcomp 2.0 [14] 中经过形式化证明的 Tarjan 模块，以计算 G_φ 的强连通分量图的拓扑排序，该图表示为列表 $C_1 :: \cdots :: C_l$。

我们定义归纳函数 inferWidth($cm,\ ls,\ S,\ \eta$) 来实现位宽推断过程，其中 cm 表示 Φ_W 约束 φ，ls 表示待处理的强连通分量列表（初始时，它是 G_φ 的强连通分量图的拓扑排序；在递归调用中，它将是此拓扑排序的前缀），S 表示其位宽已推断出的变量集合，$\eta : S \to \mathbb{N}$ 表示推断出的位宽。如果约束不可满足，其结果为不可满足；如果约束可满足，结果则为 (可满足 θ)；此时，θ 表示 ls 中强连通分量内变量的最小解。函数 inferWidth($cm,\ ls,\ S,\ \eta$) 通过以下规则实现。

– 如果 $ls =$ nil，inferWidth(cm, ls, S, η) 返回 (SAT $\emptyset$)。– 否则，令 $ls = ls' :: C$，inferWidth(cm, ls, S, η) 会调用 inferSCC(C, cm'_C)，以推断 C 中变量的位宽，其中 cm'_C 表示对应于 C 的 Φ_W-约束，即 dom(cm'_C) $= V(C)$，并且对于每个$x \in V(C)$，$cm'_C(x) = cm(x)[(\eta(z)/z)_{z\in S}]$。

- 如果 inferSCC(C, cm'_C) 返回 (SAT θ)，则会递归调用 inferWidth(cm, ls', S', η')，其中 $S' = S \cup V(C)$ 且 $\eta' = \eta \cup \theta$。
- 否则，inferWidth(cm, ls, S, η) 返回不可满足。$ls =$ nil inferWidth: $=$nil inferWidth(cm, ls, S, η) $=$(可满足 $\emptyset$)$ls = ls' :: C$ $cm'_C = (x \to cm(x)[(\eta(z)/z)_{z\in S}])$ $_{x\in V(C)}$ inferSCC(C, cm'_C) $=$(可满足 θ) $S' = S \cup V(C)$ $\eta' = \eta \cup \theta$infer Width(cm, ls', S', η') $=$(可满足 η'') inferWidth: $\neq$nil-sat-1 inferWidth(cm, ls, S, η) $=$(可满足 $\theta \cup \eta''$)$ls = ls' :: C$ $cm'_C = (x \to cm(x)[(\eta(z)/z)_{z\in S}])_{x\in V(C)}$ i nferSCC(C, cm'_C) $=$(可满足 θ) $S' = S \cup V(C)$ $\eta' = \eta \cup \theta$inferWidth(cm, ls', S', η') $=$不可满足 inferWidth: $\neq$nil-unsat-1 inferWidth(cm, ls, S, η) $=$不可满足

$$\text{inferWidth: } \neq\text{nil-不可满足-2}\quad \frac{ls = ls' :: C \quad cm'_C = (x \to cm(x)[(\eta(z)/z)_{z\in S}])^{x\in V(C)} \quad \text{inferSCC}(C, cm'_C) = \text{不可满足}}{\text{inferWidth}(cm,\ ls,\ S,\ \eta) = \text{不可满足}}$$

函数 inferSCC(C, cm) 推断强连通分量 C 中变量的位宽，该分量需要满足 Φ_W-约束 cm。与上文类似，如果约束不可满足，其结果为不可满足；如果约束可满足，则结果为 (SAT θ)；此时 θ 表示最小解。该函数会区分 C 是否为平凡情况。

– 如果 C 是平凡情况，且 $V(C) = \{x\}$，inferSCC 会计算赋值 θ，使得 dom(θ) $= \{x\}$，并且 $\theta(x)$ 是不等式 $x \geq c \in cm(x)$ 中常数 c 与 0 的最大值。在这种情况下，inferSCC 返回 (SAT θ)。

– 如果 C 不是平凡的，我们还要进一步区分 C 是否为扩张的。• 如果 C 是扩张的，inferSCC 会计算上界 $\overrightarrow{ub}$ 和下界 $\overrightarrow{lb}$，并调用函数 BaB($cm, \overrightarrow{lb}, \overrightarrow{ub}$)。（参见规则 inferSCC：nontrivial-bab，其中函数 UB(C, cm) 会按照第3节所述计算上界 $\overrightarrow{ub}$，其详细定义从略。）

• 如果 C 是非扩张的，inferSCC 会调用 maxFW(C)，计算顶点之间的最大距离 md。如果发现正权重环，inferSCC 就会报告不可满足。否则，它会计算最小解θ。（参见规则 inferSCC：nontrivial-maxfw-unsat 和 inferSCC：nontrivial-maxfw-sat。）C是平凡的 $V(C) = \{x\}$ $\theta(x) =$最大值 $\{0, c \mid x \geq c \in cm(x)\}$ inferSCC：平凡 inferSCC(C, cm) =(可满足 θ)C是扩张的 $\overrightarrow{ub} =$ UB(C, cm)$\overrightarrow{lb} = \left(\left(\text{最大值 } \left\{0, c \mid x \geq c + \sum_{x_j \in C} a_j x_j \in cm(x)\right\}\right)_{x\in V(C)}\right)$ inferSCC：nontrivial-bab inferSCC(C, cm) = BaB($cm, \overrightarrow{lb}, \overrightarrow{ub}$)$C$是非扩张的 $md =$ maxFW(C) $\exists x \in V(C).\ md(x, x) > 0$ inferSCC：nontrivial-maxfw-unsat inferSCC(C, cm) =不可满足C是非扩张的 $md =$ maxFW(C) $\forall x \in V(C).\ md(x, x) \leq 0$ $\overrightarrow{v} = \left(\left(\{0, c \mid x \geq c \in cm\ x\ \}\right)_{x\in V\ C}\right)$ 最大值 ( ) ( ) 最大值 (,) ( ) (c)$\Big)$ $\theta = \Big(\big(\{0, md\ x\ x' + v\ x' \mid x' \in C\}\big)_{x\in V}$ inferSCC：nontrivial-maxfw-sat inferSCC(C, cm) =(可满足 θ)

函数 BaB(cm, $\overrightarrow{lb}$, $\overrightarrow{ub}$) 推断扩展型强连通分量 C 中变量的位宽。函数 BaB 的 Rocq 实现由以下内容规定：

BaB规则，它们与第3节中对分支定界过程的描述非常接近。请注意，在 BaB 规则中，为简化记法，强连通分量 C 并未作为 BaB 的参数引入，而是假设其为常量，且 $V(C) = \{x_1, \ldots, x_n\}$。此外，对于约束映射 cm，我们定义 $\psi_{cm} = \bigwedge_{x \in \mathsf{dom}(cm)} \bigwedge_{x \geq t \in cm(x)} x \geq t$。

$$\text{BaB: not lb} \leq \text{ub}\ \frac{\exists i \in [n].\ lb_i > ub_i}{\mathsf{BaB}(cm, \overrightarrow{lb}, \overrightarrow{ub}) = \mathsf{UNSAT}}$$

$$\text{BaB: eq-sat}\ \frac{\overrightarrow{lb} = \overrightarrow{ub} \quad \psi_{cm}[lb_1/x_1, \ldots, lb_n/x_n] \text{ is } \mathsf{true}}{\mathsf{BaB}(cm, \overrightarrow{lb}, \overrightarrow{ub}) = (\mathsf{SAT}, \overrightarrow{lb})}$$

$$\text{BaB: eq-unsat}\ \frac{\overrightarrow{lb} = \overrightarrow{ub} \quad \psi_{cm}[lb_1/x_1, \ldots, lb_n/x_n] \text{ is } \mathsf{false}}{\mathsf{BaB}(cm, \overrightarrow{lb}, \overrightarrow{ub}) = \mathsf{UNSAT}}$$

$$\text{BaB: neq-true}\ \frac{\begin{array}{c}\overrightarrow{lb} \lneqq \overrightarrow{ub} \quad \overrightarrow{v} = (\lfloor (ub_1 + lb_1)/2 \rfloor, \ldots, \lfloor (ub_n + lb_n)/2 \rfloor) \\ \psi_{cm}[v_1/x_1, \ldots, v_n/x_n] \text{ is } \mathsf{true} \quad \overrightarrow{ub'} = \overrightarrow{v}\end{array}}{\mathsf{BaB}(cm, \overrightarrow{lb}, \overrightarrow{ub}) = \mathsf{BaB}(cm, \overrightarrow{lb}, \overrightarrow{ub'})}$$

$$\text{BaB: neq-false-rhs-1}\ \frac{\begin{array}{c}\overrightarrow{lb} \lneqq \overrightarrow{ub} \quad \overrightarrow{v} = (\lfloor (ub_1 + lb_1)/2 \rfloor, \ldots, \lfloor (ub_n + lb_n)/2 \rfloor) \\ \psi_{cm}[v_1/x_1, \ldots, v_n/x_n] \text{ is } \mathsf{false} \\ x_i \geq a_0 + \sum_{j \in [n]} a_j x_j \in cm(x_i) \text{ and } v_i < a_0 + \sum_{j \in [n]} a_j v_j \\ j_0 \in [n] \quad a_{j_0} > 0 \quad lb_{j_0} < v_{j_0} \\ \overrightarrow{ub'} = (ub_1, \ldots, ub_{j_0 - 1}, v_{j_0} - 1, ub_{j_0 + 1}, \ldots, ub_n) \\ \mathsf{BaB}(cm, \overrightarrow{lb}, \overrightarrow{ub'}) = (\mathsf{SAT}, \theta)\end{array}}{\mathsf{BaB}(cm, \overrightarrow{lb}, \overrightarrow{ub}) = (\mathsf{SAT}, \theta)}$$

$$\text{BaB: neq-false-rhs-2}\ \frac{\begin{array}{c}\overrightarrow{lb} \lneqq \overrightarrow{ub} \quad \overrightarrow{v} = (\lfloor (ub_1 + lb_1)/2 \rfloor, \ldots, \lfloor (ub_n + lb_n)/2 \rfloor) \\ \psi_{cm}[v_1/x_1, \ldots, v_n/x_n] \text{ is } \mathsf{false} \\ x_i \geq a_0 + \sum_{j \in [n]} a_j x_j \in cm(x_i) \text{ and } v_i < a_0 + \sum_{j \in [n]} a_j v_j \\ j_0 \in [n] \quad a_{j_0} > 0 \quad lb_{j_0} < v_{j_0} \\ \overrightarrow{ub'} = (ub_1, \ldots, ub_{j_0 - 1}, v_{j_0} - 1, ub_{j_0 + 1}, \ldots, ub_n) \\ \mathsf{BaB}(cm, \overrightarrow{lb}, \overrightarrow{ub'}) = \mathsf{UNSAT} \\ \overrightarrow{lb'} = (lb_1, \ldots, lb_{j_0 - 1}, v_{j_0}, lb_{j_0 + 1}, \ldots, lb_n)\end{array}}{\mathsf{BaB}(cm, \overrightarrow{lb}, \overrightarrow{ub}) = \mathsf{BaB}(cm, \overrightarrow{lb'}, \overrightarrow{ub})}$$

$$\text{BaB: neq-false-lhs}\ \frac{\begin{array}{c}\overrightarrow{lb} \lneqq \overrightarrow{ub} \quad \overrightarrow{v} = (\lfloor (ub_1 + lb_1)/2 \rfloor, \ldots, \lfloor (ub_n + lb_n)/2 \rfloor) \\ \psi_{cm}[v_1/x_1, \ldots, v_n/x_n] \text{ is } \mathsf{false} \\ x_i \geq a_0 + \sum_{j \in [n]} a_j x_j \in cm(x_i) \text{ and } v_i < a_0 + \sum_{j \in [n]} a_j v_j \\ \forall j_0 \in [n].\ a_{j_0} > 0 \rightarrow lb_{j_0} = v_{j_0} \\ \overrightarrow{lb'} = (lb_1, \ldots, lb_{i-1}, v_i + 1, lb_{i+1}, \ldots, lb_n)\end{array}}{\mathsf{BaB}(cm, \overrightarrow{lb}, \overrightarrow{ub}) = \mathsf{BaB}(cm, \overrightarrow{lb'}, \overrightarrow{ub})}$$

4.2 位宽推断过程的规范

我们首先介绍一些记号。对于 $ls = C_1 :: \cdots :: C_k$，我们用 $V(ls)$ 表示 $\bigcup_{j\in[k]} V(C_j)$。此外，我们定义

$$\psi_{cm,ls,\eta} \stackrel{\text{def}}{=} \bigwedge_{x\in V(ls)} \bigwedge_{x\geq t\in cm(x)} x \geq t[(\eta(x')/x')_{x'\in S}].$$

注意，$S\cap V(ls) = \emptyset$ 与 $S\cup V(ls) = \mathsf{dom}(cm)$。如果 $ls = \mathsf{nil}$，那么按照约定，$\psi_{cm,ls,\eta} =$ 为 true。

inferWidth(cm, ls, S, η) 的正确性由 $P_{\mathsf{inferWidth}}$ 规定，该规定断言：如果 inferWidth 返回 (可满足 η')，那么 η' 是 $\psi_{cm,ls,\eta}$ 的最小解；否则（即 inferWidth 返回不可满足），$\psi_{cm,ls,\eta}$ 不可满足。

$$P_{\mathsf{inferWidth}} \stackrel{\text{def}}{=} \forall cm, ls, S, \eta, \eta'.\\ \left(\begin{array}{l}\left(\begin{array}{l}\mathsf{inferWidth}(cm, ls, S, \eta) = (\mathsf{SAT}, \eta') \rightarrow\\ \left(\begin{array}{l}\psi_{cm,ls,\eta}[(\eta'(x)/x)_{x\in V(ls)}] \wedge\\ \forall \eta''.\, (\psi_{cm,ls,\eta}[(\eta''(x)/x)_{x\in V(ls)}] \rightarrow \eta' \leq \eta'')\end{array}\right)\end{array}\right) \wedge\\ \mathsf{inferWidth}(cm, ls, S, \eta) = \mathsf{UNSAT} \rightarrow \forall \eta''.\, \neg\psi_{cm,ls,\eta}[(\eta''(x)/x)_{x\in V(ls)}]\end{array}\right).$$

注意，η, η', η'' 在 $P_{\mathsf{inferWidth}}$ 中是量化的（二阶）变量，表示赋值函数，而非具体赋值。

inferWidth(cm, ls, S, η) 满足 $P_{\mathsf{inferWidth}}$ 的证明通过对 G_φ 中的强连通分量列表 ls 进行归纳来完成。此外，该证明依赖于后文定义的 inferSCC 满足性质 P_{inferSCC} 的证明，

$$P_{\mathsf{inferSCC}} \stackrel{\text{def}}{=} \forall cm, \theta. \left(\begin{array}{l}\left(\begin{array}{l}\mathsf{inferSCC}(C, cm) = (\mathsf{SAT}, \theta) \rightarrow\\ \left(\begin{array}{l}\psi_{cm}[(\theta(x)/x)_{x\in\mathsf{dom}(cm)}] \wedge\\ \forall \theta'.\, (\psi_{cm}[(\theta'(x)/x)_{x\in\mathsf{dom}(cm)}] \rightarrow \theta \leq \theta')\end{array}\right)\end{array}\right) \wedge\\ \mathsf{inferSCC}(C, cm) = \mathsf{UNSAT} \rightarrow \forall \theta'.\, \neg\psi_{cm}[(\theta'(x)/x)_{x\in\mathsf{dom}(cm)}]\end{array}\right).$$

而 inferSCC 满足 P_{inferSCC} 的证明，又依赖于 BaB$(cm, \overrightarrow{lb}, \overrightarrow{ub})$ 和 maxFW(C) 分别满足正确性性质 P_{BaB} 和 P_{maxFW} 的证明。

$$P_{\mathsf{BaB}} \stackrel{\text{定义}}{=} \forall cm, \overrightarrow{lb}, \overrightarrow{ub}, \theta.\\ \left(\begin{array}{l}\left(\begin{array}{l}\mathsf{BaB}(cm, \overrightarrow{lb}, \overrightarrow{ub}) = (\mathsf{SAT}, \theta) \rightarrow\\ \left(\begin{array}{l}\psi_{cm}[(\theta(x)/x)_{x\in\mathsf{dom}(cm)}] \wedge \overrightarrow{lb} \leq \theta \leq \overrightarrow{ub} \wedge\\ \forall \theta'.(\psi_{cm}[(\theta'(x)/x)_{x\in\mathsf{dom}(cm)}] \wedge \overrightarrow{lb} \leq \theta' \leq \overrightarrow{ub} \rightarrow \theta \leq \theta')\end{array}\right)\end{array}\right) \wedge\\ \mathsf{BaB}(cm, \overrightarrow{lb}, \overrightarrow{ub}) = \mathsf{UNSAT} \rightarrow \forall \theta'.\, \neg(\psi_{cm}[(\theta'(x)/x)_{x\in\mathsf{dom}(cm)}] \wedge \overrightarrow{lb} \leq \theta' \leq \overrightarrow{ub})\end{array}\right).$$

$$P_{\mathsf{maxFW}} \stackrel{\text{def}}{=} \forall C, md.\ \mathsf{maxFW}(C) = md \ \rightarrow\\ \left(\begin{array}{l}\left(\begin{array}{l}(\forall x \in V(C).\ md(x, x) \leq 0) \ \rightarrow\\ \forall x, x' \in V(C).\ \forall \text{ path } \pi \text{ from } x \text{ to } x'.\ md(x, x') \geq weight(\pi)\end{array}\right) \wedge\\ \left(\begin{array}{l}(\exists x \in V(C).\ md(x, x) > 0) \ \rightarrow\\ \exists x \in V(C).\ \exists \text{ a simple cycle } \pi \text{ from } x \text{ to } x.\ weight(\pi) > 0\end{array}\right)\end{array}\right),$$

其中，$weight(\pi)$ 表示 π 的权重，即其中各边权重之和。

4.3 位宽推断过程的验证

inferWidth(cm, ls, S, η) 满足 $P_{\text{inferWidth}}$ 的归纳证明如下。

基础步骤 $ls = \text{nil}$。那么 inferWidth(cm, ls, S, η) =(可满足 $\emptyset$)。那么 $\psi_{cm,ls,\eta}$ = 为真。很容易验证，在这种情况下，$P_{\text{inferWidth}}$ 为真。
归纳步骤。假设对于长度小于或等于n的 ls'，结论成立。考虑长度为 $n+1$的 $ls = ls' :: C$。

– 如果 inferWidth(cm, ls, S, η) =(可满足 η')，那么根据规则 inferWidth: $\neq$ nil-sat，可知对于某些 θ和 η''，inferSCC(C, cm'_C) =(可满足 θ)，且 inferWidth($cm, ls', S \cup V(C), \eta \cup \theta$) =(可满足 η'')。根据归纳假设，η''是$\psi_{cm,ls',\eta\cup\theta}$的最小解。此外，根据 P_{inferSCC}，我们知道 θ 是 $\psi_{cm'_C}$的最小解。不难看出$\psi_{cm,ls,\eta} \models \psi_{cm'_C} \wedge \psi_{cm,ls',\eta\cup\theta}$。由 $\theta \cup \eta''$是$\psi_{cm'_C} \wedge \psi_{cm,ls',\eta\cup\theta}$的最小解这一事实，可推得 $\theta \cup \eta'' = \eta'$是$\psi_{cm,ls,\eta}$的最小解。
– 如果 inferWidth(cm, ls, S, η) =不可满足，则 inferSCC(C, cm'_C) =不可满足，或者 inferSCC(C, cm'_C) =(可满足 θ) 且 inferWidth($cm, ls', S \cup C, \eta \cup \theta$) =不可满足。

- 如果 inferSCC(C, cm'_C) =不可满足，那么根据 P_{inferSCC}，我们知道 $\psi_{cm'_C}$是不可满足的。根据 $\psi_{cm,ls,\eta} \models \psi_{cm'_C}$这一事实，我们推导出$\psi_{cm,ls,\eta}$ 同样是不可满足的。

- 如果 inferSCC(C, cm'_C) =(可满足 θ) 且 inferWidth($cm, ls', S \cup C, \eta \cup \theta$) =不可满足，那么根据归纳假设，$\psi_{cm,ls',\eta\cup\theta}$ 是不可满足的。此外，根据 P_{inferSCC}，θ是 $\psi_{cm'_C}$的最小解。不难看出，$\psi_{cm,ls,\eta} \models \psi_{cm'_C} \wedge \psi_{cm,ls',\eta\cup\theta}$。由此，我们推导出$\psi_{cm,ls,\eta}$ 是不可满足的。

inferSCC(C, cm) 满足 P_{inferSCC} 的证明通过分类讨论完成。

– 如果 C 是平凡的，那么 $cm(x)$ 中的每个不等式都具有 $x \geq c$ 的形式。因此，最小（非负）解为 max$\{0, c \mid x \geq c \in cm(x)\}$。
– 如果 C 是非平凡且扩张的，那么根据规则 inferSCC: nontrivial-bab，证明主要依赖于证明 BaB(cm, $\overrightarrow{lb}$, $\overrightarrow{ub}$) 满足 P_{BaB}；这可以通过遵循分支定界过程，并对 BaB 的递归调用进行归纳来完成。

– 如果 C 是非平凡且非扩张的，那么我们使用规则 inferSCC : nontrivial-maxfw-unsat 和 inferSCC : nontrivial-maxfw-sat，主要依赖于证明最大 Floyd–Warshall 算法 maxFW(C) 满足 P_{maxFW}；这基本上遵循了 Isabelle [62] 中 Floyd–Warshall 算法正确性的证明。

现在详细说明如何证明 BaB(cm, $\overrightarrow{lb}$, $\overrightarrow{ub}$) 满足 P_{BaB}。

– 若存在 $i \in [n]$ 使得 $lb_i > ub_i$，则根据规则 BaB：逻辑非 lb $\leq$ cm $\overrightarrow{lb}$ $\overrightarrow{ub}$ ψ_{cm} $\overrightarrow{x}$ $\wedge \overrightarrow{lb} \leq$ ub，BaB(、、) 返回 $\overrightarrow{x} \leq \overrightarrow{ub}$ P_{BaB} 不可满足。因此显然 ( ) 不可满足。因此，在此情况下，成立。下文中，假设 $\forall i \in [n].\ lb_i \leq ub_i$。

– 若 $\overrightarrow{lb} = \overrightarrow{ub}$，则 $\overrightarrow{lb}$ 是 $\psi_{cm}(\overrightarrow{x}) \wedge \overrightarrow{lb} \leq \overrightarrow{x} \leq \overrightarrow{ub}$ 的唯一可能解。

- 如果 $\psi_{cm}[lb_1/x_1, \ldots, lb_n/x_n]$ 为真，那么根据规则 BaB：eq-sat，BaB(cm，$\overrightarrow{lb}$，$\overrightarrow{ub}$) 返回 (SAT $\overrightarrow{lb}$)。显然，$\overrightarrow{lb}$ 是 $\psi_{cm}(\overrightarrow{x}) \wedge \overrightarrow{lb} \leq \overrightarrow{x} \leq \overrightarrow{ub}$ 的最小解，因此在这种情况下，P_{BaB} 成立。
- 否则，$\psi_{cm}[lb_1/x_1, \ldots, lb_n/x_n]$ 为假。根据规则 BaB：eq-unsat，我们知道 BaB(cm，$\overrightarrow{lb}$，$\overrightarrow{ub}$) 返回 UNSAT。在这种情况下，显然，$\psi_{cm}(\overrightarrow{x}) \wedge \overrightarrow{lb} \leq \overrightarrow{x} \leq \overrightarrow{ub}$ 不可满足。因此，在这种情况下，P_{BaB} 成立。

– 如果 $\overrightarrow{lb} \neq \overrightarrow{ub}$ 且 $\psi_{cm}[v_1/x_1, \ldots, v_n/x_n]$ 的求值结果为真时，根据规则 BaB: neq-true，BaB(cm，$\overrightarrow{lb}$，$\overrightarrow{ub}$) 会递归调用 BaB(cm，$\overrightarrow{lb}$，$\overrightarrow{v}$)，并将其输出作为结果。显然，$\overrightarrow{v} < \overrightarrow{ub}$。根据归纳假设，BaB($cm$，$\overrightarrow{lb}$，$\overrightarrow{v}$) 返回 $\psi_{cm}(\overrightarrow{x}) \wedge \overrightarrow{lb} \leq \overrightarrow{x} \leq \overrightarrow{v}$ 的最小解（因为已经存在一个解 $\overrightarrow{v}$）。根据 $\overrightarrow{v} < \overrightarrow{ub}$，$\psi_{cm}(\overrightarrow{x}) \wedge \overrightarrow{lb} \leq \overrightarrow{x} \leq \overrightarrow{v}$ 的最小解也是 $\psi_{cm}(\overrightarrow{x}) \wedge \overrightarrow{lb} \leq \overrightarrow{x} \leq \overrightarrow{ub}$ 的最小解。因此，在这种情况下，P_{BaB} 成立。

– 如果 $\overrightarrow{lb} \neq \overrightarrow{ub}$ 且 $\psi_{cm}[v_1/x_1, \ldots, v_n/x_n]$ 的求值结果为假，则存在某个 ψ_{cm} 中的 $i \in [n]$，使得不等式 $x_i \geq a_0 + \sum_{j \in [n]} a_j x_j \; v_i > a_0 + \sum_{j \in [n]} a_j v_j$ 成立。我们区分以下几种情况。

- 若 $\exists j_0 \in [n].\, v_{j_0} > 0 \wedge a_{j_0} > 0 \wedge lb_{j0} < v_{j_0}$，则根据规则 BaB：neq-false-rhs-1 和 BaB：neq-false-rhs-2，BaB($cm, \overrightarrow{lb}, \overrightarrow{ub}$) 会递归调用 BaB($cm, \overrightarrow{lb}, \overrightarrow{ub'}$)，其中 $\overrightarrow{ub'} = (ub_1, \ldots, ub_{j_0-1}, v_{j_0} - 1, ub_{j_0+1}, \ldots, ub_n)$。
  * 若 BaB($cm, \overrightarrow{lb}, \overrightarrow{ub'}$) $= \theta$（可满足 θ），则根据归纳假设，θ 是 $\psi_{cm}(\overrightarrow{x}) \wedge \overrightarrow{lb} \leq \overrightarrow{x} \leq \overrightarrow{ub'}$ 的最小解。由于 $\overrightarrow{ub'} \leq \overrightarrow{ub}$，可得 θ $\psi_{cm}\ \overrightarrow{x}\ \wedge \overrightarrow{lb} \leq \overrightarrow{x} \leq$ 也是 ( ) 的最 $\overrightarrow{ub}$ $cm\ \overrightarrow{lb}\ \overrightarrow{ub}\ =\ \theta$ 小解。根据规则 BaB：neq-false-rhs-1，BaB(,,) (可满足)。因此，P_{BaB} 在此情况下成立。
  * 否则，BaB(cm, $\rightarrow\ \overrightarrow{lb}$, ub') $=$ 不可满足。然后，根据归纳假设，$\psi_{cm}(\overrightarrow{x}) \wedge \overrightarrow{lb} \leq \overrightarrow{x} \leq \overrightarrow{ub'}$ 不可满足。因此，如果 $\psi_{cm}(\overrightarrow{x}) \wedge \overrightarrow{lb} \leq \overrightarrow{x} \leq \overrightarrow{ub}$ 可满足，那么我们必有 $x_{j_0} \geq v_{j_0}$。所以，$\psi_{cm}(\overrightarrow{x}) \wedge \overrightarrow{lb} \leq \overrightarrow{x} \leq \overrightarrow{ub}$ 的最小解（如果存在）就是 $\psi_{cm}(\overrightarrow{x}) \wedge \overrightarrow{lb'} \leq \overrightarrow{x} \leq \overrightarrow{ub}$ 的最小解，其中

$$\overrightarrow{lb'} = (lb_1, \ldots, lb_{j_0-1}, v_{j_0}, lb_{j_0+1}, \ldots, lb_n).$$

    根据规则 BaB：neq-false-rhs-2，BaB(ψ_{cm}，$\overrightarrow{lb}$，$\overrightarrow{ub}$) 会递归调用 BaB(ψ_{cm}，$\overrightarrow{lb'}$，$\overrightarrow{ub}$)。根据归纳假设，如果 BaB(ψ_{cm}，$\overrightarrow{lb'}$，$\overrightarrow{ub}$) 返回（可满足 θ θ $\psi_{cm}\ \overrightarrow{x}\ \wedge \overrightarrow{lb'} \leq \overrightarrow{x} \leq$），那 $\overrightarrow{ub}$ $\psi_{cm}\ \overrightarrow{x}\ \wedge \overrightarrow{lb'} \leq \overrightarrow{x} \leq \overrightarrow{ub}$ 么（ ）是最小解；否则，（ ）不可满足。因此，P_{BaB} 在此情况下成立。
- 如果 $\forall j_0 \in [n].\, a_{j_0} > 0 \rightarrow lb_{j0} = v_{j_0}$，那么就不可能通过选择比 $x_i \geq a_0 + \sum_{j \in [n]} a_j x_j$ 更小的值来减小右端

。因此，必须增大 x_i的值，也就是说，必须满足 $x_i \geq v_i + 1$ ψ_{cm} $\wedge$。所以，($\vec{x}$) $\vec{lb} \leq \vec{x} \leq \vec{ub}$ ψ_{cm} $\vec{x}$ $\triangle \vec{lb'} \leq \vec{x} \leq \vec{ub}$ 的最小解（如果存在）就是 ( ) 的最小解，其中 $\vec{lb'} = (lb_1, \ldots\ldots, lb_{i-1}, v_i{+}1, lb_{i+1}, \ldots\ldots, lb_n)$。根据规则 BaB: neq- false-lhs，在此情形下，BaB(cm, $\vec{lb}$, $\vec{ub}$) 会递归调用 BaB(cm, $\vec{lb'}$, $\vec{ub}$)。根据归纳假设，如果 BaB(cm, $\vec{lb'}$, $\vec{ub}$) 返回 (SAT θ)，那么 θ 就是 ψ_{cm} $\vec{x}$ $\wedge \vec{lb'} \leq \vec{x} \leq \vec{ub}$ ψ_{cm} $\vec{x}$ $\wedge \vec{lb'} \leq$ ( ) $\vec{x} \leq \vec{ub}$ P_{BaB} 的最小解；否则，( ) 不可满足。因此，在此情形下，成立。

5 OCaml 实现与评估

5.1 实现

基于第 4 节中归纳函数 inferWidth 的 Rocq 实现，我们利用 Rocq 证明助手中提供的提取机制 [35,36] 提取了 OCaml 代码。此外，由于 inferWidth 函数以 Φ_W-约束作为输入，我们实现了一个解析器，将 FIRRTL 程序转换为其抽象语法树（AST），再由约束提取器从中 Φ_W提取约束。该解析器使用 ocamlyacc [33],实现，其中指定了 FIRRTL 语言的上下文无关文法（附带语义动作）；约束提取器则由 Rocq 实现提取而来。inferWidth 函数的 OCaml 实现，连同解析器和约束提取器，构成了 FIRRTL 的 InferWidths 遍的 OCaml 实现（见图 5），我们称之为 BFWInferWidths（“BFW”是“分支定界 Floyd–Warshall”的缩写）。

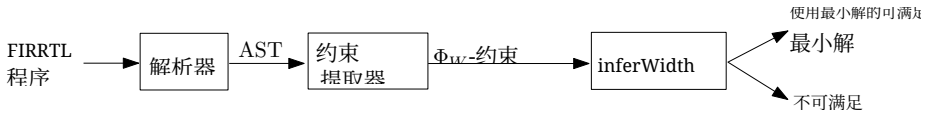

图 5：BFWInferWidths 的架构

在对由 RISC-V BOOM 处理器生成的位宽推断实例运行 BFWInferWidths 时，OCaml 标准库 Base 中函数 split、concat 和 flat map 的非尾递归实现导致了段错误和栈溢出。主要原因是这些实现无法扩展到大型真实实例，而此类实例可能包含超过 200,000 个变量和超过 130,000 个不等式。我们将这些非尾递归实现替换为尾递归版本。

5.2 评估

基准测试。为了评估 BFWInferWidths 的性能，我们收集了 75 个包含未指定宽度组件的 FIRRTL 程序：

– FIRRTL 规范 [57], 中的 4 个 FIRRTL 程序– 从 Chisel 手册 [55], 中的 Chisel 程序生成的 20 个 FIRRTL 程序– firtool 单元测试中使用的 11 个 FIRRTL 程序 [17],– 作为 firtool 问题报告的 26 个 FIRRTL 程序 [16],– 11 个（手动生成的）FIRRTL 程序，其中未指定宽度的组件之间存在循环依赖关系，– 用于生产级、基于 Chisel 的 RISC-V 处理器设计的 3 个 FIRRTL 程序: NutShell [48], 、Rocket Chip [53]与 RISC-V BOOM [52]。

这 75 个 FIRRTL 程序被划分为两个基准测试套件，即 REALWORLD（NutShell、Rocket Chip 和 RISC-V BOOM 的 3 个 FIRRTL 程序）和 MANUAL（所有其他 FIRRTL 程序）。

我们发现`dshl` 出现在三个基准测试中，即 NutShell（7 次）、Rocket Chip（3 次）和 RISC-V BOOM（3955 次）。所有出现情况均属于以下两种情形之一。

– 移位实际上不是动态的，而是由一个常数位数决定：例如，`y=dshl(x,3)`，其位宽约束为 $w_y \geq w_x + 2^3 - 1$，即 $w_y \geq w_x + 7$。– 不存在循环依赖：例如，`y=dshl(x, v)`，其位宽约束为 $w_y \geq w_x + 2^{w_v} - 1$，但v 的位宽不依赖于y 的位宽。因此，可以在推断 w_y 之前推断出 w_v （在对 inferSCC 的翻译过程进行单独调用时）。假设 $w_v = c2$，那么 $w_y \geq w_x + 2^{w_v} - 1$ 可化简为 $w_y \geq w_x + 2^{c2} - 1$，这是一个线性约束。

因此，基准测试中出现的所有动态左移都可以由我们的过程处理。不过，支持一般的动态左移动态左移 仍留待未来工作。

实验设置。 我们将 BFWIn- ferWidths 的正确性与效率和 firtool 中 I nferWidths 遍的实现进行比较。我们还将其与工业级整数线性规划求解器 Gurobi [21],进行比较。Gurobi 是一种基于分支定界的优化器，据 [37]所述，代表了当前最先进的水平。Gurobi 提供最小值函数，使我们能够对 FIRWINE 约束进行编码，并通过一个单目标函数求解，该函数对所有位宽变量求和。请回顾，如果问题有解，那么它必定有唯一的最小解（参见命题 2）。因此，使用单个目标函数就足够了，无需进行多目标优化（即为每个位宽变量分别设置一个目标函数）。

我们指出，对于不支持最小值函数的整数线性规划优化器，可以使用 Big-M（足够大的整数）来消除该函数。例如， $a \geq \min(y, z)$ 可以表示为 $a \geq x \wedge y \geq x \wedge z \geq x \wedge x \geq y - M * b \wedge x \geq -M + z + M * b$，其中 x 是一个辅助非负变量，且 b 是一个布尔变量。

请注意，尽管我们在第4节中对 InferWidths 的正确性进行了形式化证明，但 BFWInferWidths 仍包含未经验证的组件（解析器和约束提取器），以及未经验证的 OCaml 标准库 Base。因此，进行实证评估有助于确认 BFWInferWidths 整体的正确性。

表 1：BFWInferWidths 的正确性：与 firtool 和 Gurobi 的比较。

基准测试套件 # 程序数量		已求解实例数量		
		firtool	Gurobi	BFWInferWidths
MANUAL	72	60	72	72
真实世界	3	3	3	3

表 2：BFWInferWidths 与 firtool 和 Gurobi 的效率对比。（firtool 的耗时根据其能够求解的 60 个实例计算。）

基准测试		Avg. #cpnts	每个实例的耗时（毫秒）		
			firtool	Gurobi	BFWInferWidths
MANUAL（72）		119	7.49	12.07	**1.00**
真实世界	NutShell	7,152	190.70	194.55	**158.31**
	Rocket Chip	4,882	127.90	120.64	**22.24**
	RISC-V BOOM	205,608	8,338.30	**3,326.94**	3,467.80

所有实验均在一台配备 8 GB 内存的 3.2 GHz 八核 Apple M1 笔记本电脑上运行。对比实验中，我们使用 firtool v1.73 和 Gurobi v12.0.1，并启用了 8 线程并行优化。（请注意，虽然 Gurobi 使用 8 个线程，但 BFWInferWidths 和 firtool 仅使用单线程。）

实验结果。正确性评估结果见表 1。我们观察到，BFWInferWidths 的结果与 firtool 和 Gurobi 的结果一致，但 firtool 在 MANUAL 基准测试套件中的 12 个实例上运行失败。具体而言，firtool 在 24 小时内未能完成一个不含任何循环依赖的 FIRRTL 程序（产生了 192 个不等式），而 BFWInferWidths 和 Gurobi 均在不到 10 毫秒内解决了该问题。事实上，这是由于 firtool 中一个尚未解决的问题 [22]。firtool 因循环依赖，在其他 11 个 FIRRTL 程序上错误地抛出了异常。

效率评估结果见表 2，其中 Avg. #cpnts 列显示了声明为未指定宽度的组件的平均数量。我们观察到，在 MANUAL 基准测试套件中（声明为未指定宽度的组件数量较少），BFWInferWidths 的平均速度至少是 firtool（分别是 Gurobi）的 7 倍（分别是 10 倍）。在真实世界基准测试套件中，BFWInferWidths 的效率高于 firtool，且与 Gurobi 大致相当（BFWInferWidths 在 NutShell 和 Rocket Chip 上更快，但在 RISC-V BOOM 上略慢）。详细结果见附录 C。

我们还通过将位宽推断问题编码为最小化问题，对 Z3 的优化器 [43],进行了初步比较。在 BOOM 基准测试上，我们的方法比 Z3 快约 $26\times$。关键的是，与 Gurobi 一样，Z3 并未提供形式化验证的求解器，这正是我们应当强调的关键区别。

总而言之，作为从经过形式化验证的 Coq 实现中提取的 OCaml 程序，BFWInferWidths 展现出优于

FIRRTL 最先进编译器 firtool 的性能。其性能可与使用 8 个线程运行的工业级整数线性规划求解器 Gurobi 相媲美。

6 相关工作

编译器验证。编译器验证可追溯至 1967年，当时[41] 证明了从算术表达式到栈机代码的转换正确性。如今，这已发展为一个充满活力的领域，涵盖从单个编译遍到编译器中复杂代码优化的方方面面 [15]。

软件编程语言经认证编译方面的代表性工作包括：面向 C 语言的 CompCert [31,26,30,32,8,27]、面向 Lustre 的 Vélus [9], 、面向 LLVM IR 的 Ve- llvm [65,66] 和 Crellvm [25] ，以及面向 ML 的 CakeML [29,39,59,50], ，仅举数例。特别是，交互式定理证明器 Rocq 已被广泛用于形式化程序语义、编译和优化遍；在此基础上，可以证明编译正确性，并自动综合可执行解释器/编译器（参见 [1]）。例如， [5] 形式化并证明了一个编译器，它将简单函数式语言中的程序转换为理想化汇编语言的代码； [11,12] 形式化并证明了多个编译器，它们将简单类型或无类型 λ 演算中的程序转换为理想化汇编语言的代码；CompCert 是一个经过验证的编译器，可将 Clight（C 语言的一个大型子集）中的程序转换为 PowerPC 汇编语言代码。针对特定领域的系统， [42] 设计了一种面向安全的语言，用于底层代数库。其基于 CompCert 的编译器能够形式化地保持语义（在 Rocq 中验证），而该语言则支持对安全性属性进行演绎验证。 [51] 提出了一种混合符号方法，用于验证面向高层次综合的原始 C/C++ 程序与转换后程序之间的语义保持，并处理复杂的循环/缓冲区重构，从而将经认证的编译原则扩展至硬件综合流程。

约束求解器的验证。目前已有多项工作致力于开发形式化验证的约束求解器，其中大多数聚焦于 SAT 求解器，但采用的方法各不相同。例如，Lescuyer 和 Sylvain 使用 Rocq [34]；Shankar 和 Vaucher 使用 PVS [56]；Maric 在 Isabelle/HOL 中采用浅嵌入 [40,47]；Fleury 等人使用精化框架 [7,19,18]；Oe 等人利用经过验证的编程语言 Guru [49,58]。除 SAT 求解器外，还有针对有限域上的约束编程以及字符串约束的求解器 [10,24] ，分别使用 Rocq 和 Isabelle/HOL 进行开发与验证。相比之下，我们的工作聚焦于一类线性整数约束的经验证求解器，这类约束与先前工作的研究对象相去甚远。

求解线性约束。Presburger 算术及其片段在验证与优化中发挥着关键作用。本文研究的 FIRWINE 约束属于整数线性算术的一个片段，其求解算法已得到广泛研究 [28]。现代 SMT 求解器，如

Z3 [45,44] 和 CVC5 [4] ，实现了高度优化的无量词线性整数算术决策过程。它们的模理论优化（OMT）扩展 [6,46] 可针对整数线性算术约束优化目标函数。尽管带有 OMT 扩展的 SMT 求解器能够高效求解 FIRWINE 约束，但例如其内部启发式策略过于复杂，无法满足完整验证的需求。

ILP 通常是 NP 完全问题，因此其最坏情况时间复杂度呈指数级。尽管通用的整数线性规划求解器（例如 Gurobi 和 Z3）采用了各种启发式优化，但其实现未经验证且难以验证。相比之下，我们提供了一种专门用于求解 FIRRTL 位宽不等式的过程，其复杂度显著更低。此外，该过程基于经典的分支定界（BaB）算法（该算法广泛应用于组合优化）以及用于求解全源最短路径问题的经典 Floyd–Warshall 算法。我们在手工构造的基准测试和真实世界的基准测试上，将实现与商业整数线性规划求解器 Gurobi 进行了实证比较。结果表明，我们的实现优于 Gurobi，或达到了可比的性能。特别是，我们的求解算法拥有模块化的 Rocq 证明，这大幅推进了 FIRRTL 经过验证的编译器的发展前沿。

7 结论

本文研究了 FIRRTL 程序编译中的位宽推断问题。我们从 FIRRTL 程序中推导出一类线性整数约束，其中某些组件的位宽未被显式给出，并提供了求解这些约束的完整过程。此外，该过程已在 Rocq 中经过形式化验证，同时还派生出了一个具有竞争力性能的 OCaml 实现，作为 firtool 中官方的 InferWidths 遍。

本工作介绍了首个经过形式化验证的 InferWidths 编译遍，标志着向生产级、经过验证的 FIRRTL 编译器迈出了重要一步。总体而言，这项工作推动了经过形式化验证的硬件这一目标的实现，有助于构建一套经过验证的执行栈，确保从软件到硬件各个层面的正确性保证。

下一步，我们将扩展这一方法，以处理动态左移，因为它会在位宽约束中引入指数项。此外，将本文提出的过程用 C++ 实现也很有意义，这有望取代 firtool 中现有的 InferWidths 遍。

数据可用性声明

我们的过程的 Rocq 实现、其正确性证明、位宽推断工具的源代码以及基准测试可在 [60] 获取。

参考文献

1. Awesome Coq：精选的 Coq 库、插件、工具与资源列表。https://github.com/coq-community/awesome-coq（2023年） 2. Bachrach, J., Vo, H., Richards, B., Lee, Y., Waterman, A., Avižienis, R., Wawrzynek, J., Asanović, K.: Chisel：在 Scala 嵌入式语言中构造硬件。载于：设计自动化会议（DAC）论文集。第 1212–1221 页 2228584 3 2012年 10 1145 2228360。ACM（）。https://doi.org/。/。。Bao, Y., Carlson, T.E.：敏捷与开源硬件。IEEE Micro **40**（4），6–9（2020年 3002606 4 10 1109 2020）。https://doi.org/。/MM。。Barbosa, H., Barrett, C., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mohamed, A., Mohamed, M., Niemetz, A., Nötzli, A., Ozdemir, A., Preiner, M., Reynolds, A., Sheng, Y., Tinelli, C., Zohar, Y.：cvc5：一种多用途的工业级 SMT 求解器。载于：第 28 届系统构造与分析工具和算法国际会议（TACAS）论文集。第 415–442 页（2022年） 5. Benton, N., Tabareau, N.：将函数类型编译为低级命令式代码的关系规范。载于：ACM SIGPLAN 语言设计与实现中的类型国际研讨会（TLDI）论文集。第 3–14 页。ACM（ 2009 10 1145 1481861 1481864 ）。https://doi.org/。/。。6 Bjørner, N.S., Phan, A.D., Fleckenstein, L.：zν——一种优化型 SMT 求解器。载于：系统构造与分析工具和算法国际会议（TACAS）论文集。第 194–199 页（2015年） 7. Blanchette, J.C., Fleury, M., Lammich, P., Weidenbach, C.：一种支持学习、遗忘、重启与增量性的经验证 SAT 求解器框架。《自动化推理杂志》**61**（1-4），333–365（2018年） 8. Blazy, S., Dargaye, Z., Leroy, X.：C 编译器前端的形式化验证。载于：第 14 届形式化方法国际研讨会（FM）论文集。第 4085, 460 475 2006 10 1007 11813040_31 9 卷，第 – 页。Springer（）。https://doi.org/。/。。Bourke, T., Brun, L., Dagand, P.É., Leroy, X., Pouzet, M., Rieg, L.：一种经过形式化验证的 Lustre 编译器。载于：第 38 届 ACM SIGPLAN编程语言设计与实现大会（PLDI）论文集。第 586–601 页。ACM（2017年 3062358 10 10 1145 3062341）。https://doi.org/。/。。Carlier, M., Dubois, C., Gotlieb, A.：有限域上的经认证约束求解器。载于：第 18 届国际形式化方法研讨会（FM）论文集。第 116–131 页（2012年） 11. Chlipala, A.：从 λ 演算到汇编语言的经认证保持类型的编译器。载于：ACM SIGPLAN 2007 编程语言设计与实现大会（PLDI）论文集。第 54–65 页。ACM（2007年）。https: //doi.org/10 1250742 12 1145 1250734。/。。Chlipala, A.：一种用于非纯函数式语言的经过验证的编译器。载于：第 37 届 ACM SIGPLAN-SIGACT 编程语言原理研讨会（POPL）论文集。第 93–106 页。ACM（2010年）。https://doi.org/10.1145/1706299.1706312

13. CIRCT 项目。https://circt.llvm.org（2023年） 14. Cohen, C., Palmskog, K.: Tarjan 与 Kosaraju。https://github.com/ coq-community/tarjan（2025年） 15. Dave, M.A.：编译器验证：参考文献目录。ACM SIGSOFT 软件工程札记**28**966235 16 6 2 2003年 10 1145 966221 2025年（）， （）。https://doi.org/。/。。Firtool 问题。https://github.com/llvm/circt/issues（） 17. Firtool 1.74.0 版本。https://github.com/llvm/circt/releases/tag/ firtool-1.74.0（2025年） 18. Fleury, M.: 优化经验证的 SAT 求解器。载于：第 11 届 NASA 形式化方法国际研讨会（NFM）论文集。第 148–165 页（2019年） 19. Fleury, M., Blanchette, J.C., Lammich, P.：使用命令式 HOL 和监视文字的经验证 SAT 求解器。载于：第 7 届 ACM SIGPLAN 经认证程序与证明国际会议（CPP）论文集。第 158–171 页。ACM（2018年） 20. Floyd, R.W.：算法 97：最短路径。《ACM 通讯》**5**（6），345（1962年）。https://doi.org/10 368168 21 1145 367766。/。。Gurobi 优化。https://www.gurobi.com/（2025年） 22. 推断宽度的不可中断循环检查性能问题 #6742。https:// github.com/llvm/circt/issues/6742（2025年） 23. Izraelevitz, A.M., Koenig, J., Li, P., Lin, R., Wang, A., Magyar, A., Kim, D., Schmidt, C., Markley, C., Lawson, J., Bachrach, J.：可复用性是 FIRRTL 的根基：硬件构造语言、编译器框架与转换。载于：IEEE/ACM 计算机辅助设计国际会议（ICCAD）论文集。第 209–216 页（2017年）。https://doi.org/10.1109/ICCAD.2017。8203780 24. Kan, S., Lin, A.W., Rümmer, P., Schrader, M.：CertiStr：经认证的字符串求解器。载于：第 11 届 ACM SIGPLAN 经认证程序与证明国际会议（CPP）论文集。第 210–224 页。ACM（2022年） 25. Kang, J., Kim, Y., Song, Y., Lee, J., Park, S., Shin, M.D., Kim, Y., Cho, S., Choi, J., Hur, C.K., Yi, K.：Crellvm：面向 LLVM 的经验证可信编译。载于：第 39 届 ACM SIGPLAN编程语言设计与实现大会（PLDI）论文集。第 631–645 页。ACM（2018年）。https://doi.org/ 10.1145/3192366.3192377 26. Kästner, D., Barrho, J., Wünsche, U., Schlickling, M., Schommer, B., Schmidt, M., Ferdinand, C., Leroy, X., Blazy, S.：CompCert：集成和认证经过形式化验证的优化编译器的实践经验。载于：第 9 届欧洲嵌入式实时软件与系统大会（ERTS）论文集。第 1–9 页（2018年） 27. Kästner, D., Leroy, X., Blazy, S., Schommer, B., Schmidt, M., Ferdinand, C.：弥合差距——经过形式化验证的优化编译器 CompCert。载于：第二十五届安全关键系统研讨会（SSS）论文集。第 163–180 页（2017年） 28. Kroening, D., Strichman, O.：决策过程——算法视角（第二版）。理论计算机科学丛书。EATCS 系列，Springer（2016年） 29. Kumar, R., Myreen, M.O., Norrish, M., Owens, S.：CakeML：ML 的经验证实现。载于：第 41 届 ACM SIGPLAN-SIGACT 编程语言原理年度研讨会（POPL）论文集。第 179–191 页。ACM（2014年 2535841 30 10 1145 2535838）。https://doi.org/。/。。Leroy, X.：编译器后端的形式化认证，或：使用证明助手编写编译器。载于：第 33 届 ACM SIGPLAN-SIGACT

编程语言原理研讨会（POPL）。第42–54页。ACM（2006 1111042 31 10 1145 1111320）。https://doi.org/。/. . Leroy, X.：现实编译器的形式化验证。《ACM 通讯》**52**(7)，107–115 1538814 32 2009 10 1145 1538788（）。https://doi.org/。/. . Leroy, X., Blazy, S., Kästner, D., Schommer, B., Pister, M., Ferdinand, C.：CompCert：经过形式化验证的优化编译器。载于：第8届欧洲嵌入式实时软件与系统大会（ERTS）论文集（2016）33. Leroy, X., Doligez, D., Frisch, A., Garrigue, J., Rémy, D., Vouillon, J.：OCaml 系统 5.3 版：文档与用户手册。第17章：词法分析器与语法分析器生成器（ocamllex、ocamlyacc）。法国国家信息与自动化研究所（2025），https://ocaml.org/manual/5. 3/lexyacc.html 34. Lescuyer, S., Conchon, S.：在 Coq 中对 SAT 求解器进行反射式形式化。载于：第21届高阶逻辑定理证明国际会议论文集。第64页（2008）35. Letouzey, P.：Coq 的一种新提取方法。载于：第2届证明与程序类型国际研讨会（TYPES）论文集，精选论文。第200–219 2002 10 1007 3 540 39185 1_12 页。36 4 Springer（）。https://doi.org/。/- - -。Letouzey, P.：Coq 中的提取：概述。载于：第 th 届欧洲可计算性会议逻辑与算法理论会议（CiE）论文集。第5028卷，第359 369 2008 10 1007 978 3 540 69407 6_39 – 37 页。Springer（）。https://doi.org/。/- - - -。Lin, P., Zou, M., Chen, Z., Cai, S.：Parailp：一种具有协同进化机制的整数线性规划并行局部搜索框架。第三十三届国际人工智能联合会议论文集，第6949–6957页（2024）38. LLVM 编译器基础设施。https://llvm.org/（2023）39. Lööw, A., Kumar, R., Tan, Y.K., Myreen, M.O., Norrish, M., Abrahamsson, O., Fox, A.：在经过验证的处理器上验证编译。载于：第40届 ACM SIGPLAN 编程语言设计与实现会议（PLDI）论文集。第1041–1053页。ACM（2019）。https://doi.org/10.1145/3314221. 3314622 40. Maric, F.：通过浅嵌入 Isabelle/HOL，对现代 SAT 求解器进行形式化验证。《理论计算机科学》**411**(50)，4333–4356（2010）41. McCarthy, J., Painter, J.：算术表达式编译器的正确性。载于：《计算机科学的数学方面 1》。应用数学专题讨论会论文集，第19卷，第219–222页（1967）42. Melquiond, G., Moreau, J.：用于计算机代数的安全低级语言及其经过形式化验证的编译器。《ACM 程序设计语言》**8**(ICFP)（2024）。https: //doi.org/10 3674629 43 1145. / . 微软研究院：Z3 编程。https://theory.stanford.edu/~nikolaj/programmingz3.html#sec-optimization 44. de Moura, L., Bjørner, N.：SMT 求解器的高效 E匹配。载于：第21届自动演绎国际会议（CADE）论文集。第183–198页。Springer（2007）45. de Moura, L., Bjørner, N.：Z3：一种高效的 SMT 求解器。载于：第14届国际系统构造与分析工具和算法会议（TACAS）论文集。第337–340页。Springer（2008）46. Nieuwenhuis, R., Oliveras, A.：关于理论模糊SAT与优化问题。载于：第9届可满足性测试理论与应用国际会议（SAT）论文集。第156–169页。Springer（2006）。https://doi.org/ 10.1007/11814948_18

47. Nipkow, T., Paulson, L.C., Wenzel, M.：Isabelle/HOL——高阶逻辑证明助手。Springer（2002）48. NutShell RISC-V CPU。https://github.com/OSCPU/NutShell（2023）49. Oe, D., Stump, A., Oliver, C., Clancy, K.：versat：经过验证的现代 SAT 求解器。载于：第13届验证、模型检查与抽象解释国际会议（VMCAI）论文集。第363–378页（2012）50. Owens, S., Norrish, M., Kumar, R., Myreen, M.O., Tan, Y.K.：验证 CakeML 中高效的函数调用。《 ACM 程序设计语言》 **1**(ICFP)，18:1–18:27 （ 2017 ）。https://doi.org/10 3110262 51 1145. / . Pouchet, L.N., Tucker, E., Zhang, N., Chen, H., Pal, D., Rodríguez, G., Zhang, Z.：高层次综合源到源转换的形式化验证。载于：2024年 ACM/SIGDA 国际现场可编程门阵列研讨会（FPGA）论文集。第97–107页。ACM（2024）。https://doi.org/10.1145/3626202. 3637563 52. RISC-V BOOM：伯克利乱序 RISC-V 处理器。https://boom-core. org/（2023）53. Rocket chip RISC-V CPU 生成器。https://github.com/chipsalliance/ rocket-chip（2023）54. Salama, C., Malecha, G., Taha, W., Grundy, J., O'Leary, J.：Verilog 导线互连的静态一致性检查：使用依赖类型检查 Verilog 描述的合理性。载于：2009年 ACM SIGPLAN 部分求值与程序操纵研讨会论文集。第121–130页。PEPM '09，计算机协会（2009）。https://doi.org/10.1145/1480945.1480963，https://doi.org/10 1480963 55 1145 1480945 . / . . Schoeberl, M.：使用 Chisel 进行数字设计。Kindle Direct Publishing（2025），https: //www.imm.dtu.dk/~masca/chisel-book.html 56. Shankar, N., Vaucher, M.：对基于 DPLL 的可满足性求解器进行机械化验证。载于：第5届逻辑与语义框架及其应用研讨会（LSFA）论文集。第3–17页（2010）57. FIRRTL 语言规范（5.0.0 版）。https://github.com/chipsalliance/firrtl-spec/releases/download/v5.0.0/spec.pdf（2025）58. Stump, A., Deters, M., Petcher, A., Schiller, T., Simpson, T.W.：在 Guru 中进行经验证的程序设计。载于：第3届 ACM 编程语言与程序验证研讨会（PLPV）论文集。第49–58页。ACM（2009）59. Tan, Y.K., Myreen, M.O., Kumar, R., Fox, A., Owens, S., Norrish, M.：经过验证的 CakeML 编译器后端。《函数式编程期刊》**29**，e2（2019）。https://doi.org/ 10.1017/S0956796818000229 60. Wang, K., andJiaxiang Liu, X.S., Wu, Z., Song, F., Chen, T., Jansen, D.N.：BFWInferWidths：一种经过形式化验证的 FIRRTL 位宽推断工具。https: //github.com/wky17/VerinferWidth（2026），https://doi.org/10.5281/zenodo. 18194615 61. Warshall, S.：布尔矩阵定理。《AC M 期刊》**9**(1)，11–12（1962）。https: //doi.org/10 321107 62 1145 321105. / . . Wimmer, S., Lammich, P.：最短路径的 Floyd–Warshall 算法（2017），https://www.isa-afp.org/entries/Floyd_Warshall.html，Isabelle/HOL 中的形式化证明开发 63. 香山：开源高性能 RISC-V 处理器。https://github.com/OpenXiangShan/XiangShan（2023）64. Xu, Y., Yu, Z., Tang, D., Chen, G., Chen, L., Gou, L., Jin, Y., Li, Q., Li, X., Li, Z., Lin, J., Liu, T., Liu, Z., Tan, J., Wang, H., Wang, H., Wang, K., Zhang, C.,

Zhang, F., Zhang, L., Zhang, Z., Zhao, Y., Zhou, Y., Zhou, Y., Zou, J., Cai, Y., Huan, D., Li, Z., Zhao, J., Chen, Z., He, W., Quan, Q., Liu, X., Wang, S., Shi, K., Sun, N., Bao, Y.：使用敏捷方法开发高性能 RISC-V 处理器。载于：第55届IEEE/ACM国际微体系结构研讨会论文集。第1178–1199页。IEEE（2022）。https://doi.org/10.1109/MICRO56248.2022.00080 65. Zhao, J., Nagarakatte, S., Martin, M.M.K., Zdancewic, S.：为经验证的程序变换形式化 LLVM中间表示。载于：第39届ACM SIGPLAN-SIGACT编程语言原理研讨会（POPL）论文集。第427–440页。ACM（2012）。https://doi.org/10.1145/ 2103621.2103709 66. Zhao, J., Nagarakatte, S., Martin, M.M.K., Zdancewic, S.：LLVM 基于SSA的优化的形式化验证。载于：ACM SIGPLAN 编程语言设计与实现大会（PLDI）论文集。第175–186页。ACM（2013）。https://doi.org/10.1145/2499370.2462164

FIRRTL 的语法

$$\Phi_W := \bigwedge_{x_i \in X} (x_i \geq \min(t_{i,1}, \ldots, t_{i,k_i}))$$

$$\Phi_{\text{nf}} := \bigvee_{j_1 \in [\ell]} \bigwedge_{j_2 \in [m_{j_1}]} z_{j_1,j_2} \geq t_{j_1,j_2}$$

FIRRTL 的语法规则以扩展巴科斯-诺尔范式列于图 6 中，其中 $\{\cdots\}$ 表示重复（重复零次或多次），$[\cdots]$ 表示可选（包含或省略），终结符则用“和”括起来。此处给出的 FIRRTL 语法规则是对官方规范中规则的简化。[7] 例如，为便于阅读，省略了缩进、取消缩进和换行符终结符；此外，由于本文聚焦于数字电路，也省略了模拟信号的类型和结构。在语法中，存储器也被省略，因为其语义仍存在争议。[8]

根据语法，FIRRTL 具有向量类型、捆绑类型、条件语句、连接和模块等高级结构。这些高级结构随后通过一系列降级转换逐步移除。在每次降级转换中，电路都会被改写为使用更简单、层级更低的结构的等效电路。最终，电路会简化为限制最严格的形式，类似于结构化网表，从而便于将其转换为输出语言（例如 Verilog）。这种形式称为降级 FIRRTL（即 LoFIRRTL），是完整 FIRRTL 语言的严格子集。为便于区分，原始形式称为高级 FIRRTL（HiFIRRTL）。

如果 FIRRTL 电路遵守以下限制，则它是 LoFIRRTL 电路：

– 所有位宽均已显式定义。– 不使用条件语句（`when`）。– 所有电路组件均以基元类型声明。– 每个组件至多连接一次。

更准确地说，LoFIRRTL 电路是由图 7 中的语法规则定义的 FIRRTL 电路，并且满足每个模块中的每个组件至多连接一次。

请注意，在图 7 中，f、id、op 的规则与图 6 中的规则相同（因此省略）。简而言之，图 7 中的语法规则是在图 6 中规则的基础上，将 $[w]$替换为 w（从而使位宽成为必填项），从 s规则中删除`when` 语句，并删除聚合类型 ta的规则而得到的。每个组件最多只能连接一次这一要求，使 LoFIRRTL 带有类似于软件程序静态单赋值（SSA）形式的特征。SSA 形式的程序中，每个变量最多赋值一次，并且每个变量都在使用前定义。例如，图 8(a) 中的程序是一个 LoFIRRTL 程序，而

[7] https://github.com/chipsalliance/firrtl-spec/blob/main/spec.md[8] https://github.com/chipsalliance/chisel/issues/3112

c ::=“电路” id “：” $\{m\}m$ ::=“模块” $id\{p\}\{s\}p$ ::= (“输入” |“输出”) id “：” t
s ::= “导线” id “：” t |“寄存器” id “：” t e[“（带有：复位=>(”e “，” e“)）”]
“实例” id “的” id | “节点” id “=”id | r“<=”e | r “无效” | “当” e “：” $\{s\}$[“否则：”$\{s\}$] | “跳过”
e ::= | $[w]$ int | r | e e e | op :: (“无符号整数” “有符号整数”) “(” “)” “多路复用器(” “，” “，” “)” ::= id | r“.”id | r“[” “]”
w ::=“<”int“>”t ::= tg | $tatg$ ::= “时钟” |“复位” | “异步复位” | (“无符号整数” | “有符号整数”)$[w]ta$ ::= “{”$f\{f\}$“}” | t“[”int“]”f ::= [“翻转”] id “：” tid ::= (“_” | l)$\{$“_” | l | $d\}op$ ::= op_2e | op_1e | op_1e1i | op_1e2i
op_2e ::= op_2e_key“(”e“，”e“)”op_2e_key ::= “加法” |“减法” | “乘法” | “除法” | “取模” | “小于” | “小于等于” | “大于” | “大于等于”| “相等” | “不等” | “动态左移” | “动态右移” |“与” |“或” |“异或” | “拼接”op_1e ::= op_1e_key“(”e“)”op_1e_key ::= “转换为无符号整数” | “转换为有符号整数” | “转换为时钟” | “类型转换” |“取负”|“逻辑非” | “归约与” | “归约或” | “归约异或”
op_1e1i ::= op_1e1i_key“(”e“，”int“)”op_1e1i_key ::= “填充” | “左移” | “右移” | “高位截取” |“低位截取”op_1e2i ::= op_1e2i_key“(”e“，”int“，”int“)”op_1e2i_key ::=“位切片”int ::= [“-”]$d\{d\}$ |“\"b”[“-”]$d_b\{d_b\}$“\""|“\"o”[“-”]$d_o\{d_o\}$“\"” | “\"h”[“-”]$d_h\{d_h\}$“\""

d （分别表示十进制、二进制、八进制和十六进制）数字， l 表示字母

图 6：FIRRTL 的语法

s ::= “导线” id “:” t | “寄存器” id “:” t e[“(具有:{复位=>(”e“,”e“)})”]
|“实例” id “的” id |“节点” id“=”id | r“<=”e | r “无效” | “跳过”
e ::= (“无符号整数”|“有符号整数”) w“(”int“)” | r | “多路复用器(”e“,”e“,”e“)” | op
r ::= id | r“。”id | r“[”int“]”
w ::=“<”int“>”
t ::= tg
tg ::= “时钟” | “异步复位” |(“无符号整数” |“有符号整数”) w

图 7：LoFIRRTL 语法（与图 6 中相同的规则不再重复列出）

```
1模块 MyModule:
2     输入 a:  无符号整数 <1>
3     输入 b:  无符号整数 <1>
4     输出 myport1:  无符号整数 <1>
5     输出 myport2:  无符号整数 <1>
6     我的端口1  <=b
7     我的端口2  <=a
```

(a)

```
1模块 MyModule  :
2     输入 a: 无符号整数 <1>
3     输入 b: 无符号整数 <1>
4     输出 myport1 : 无符号整数 <1>
5     输出 myport2 : 无符号整数 <1>
6     myport1  <=a
7     myport1  <=b
8     myport2  <=a
```

(b)

图 8：用于说明 LoFIRRTL 语法限制的示例

图 8(b) 中的程序则不是，因为输出端口 `myport1` 分别与输入端口 `a` 和 `b`连接了两次。

B FIRRTL 规范中的表达式宽度

给定表达式 e，其宽度 w_{e} 在表3中通过归纳定义，其中$t_{\mathsf{e}} \in \{$无符号整数、有符号整数、复位、异步复位、时钟$\}$ 表示 e 的类型。我们注意到，通常情况下，在运算符多路复用器 中使用的引用 r 和表达式 e_i for $i = 1$、2 可能是聚合类型（即束类型和向量类型）的组件。类型检查确保从 r 或多路复用器 连接的组件具有兼容类型。因此，其连接语句可以按字段或元素逐一拆分。由此，只需要确定基元类型表达式的宽度。

表3：表达式 e 的位宽 w_{e}

操作	位宽	Note
e =无符号整数<n_1>(n_2)	$w_{\mathsf{e}} = n_1$	$n_1, n_2 \in \mathbb{N}$
e =有符号整数<n_1>(n_2)	$w_{\mathsf{e}} = n_1$	$n_1 \in \mathbb{N}, n_2 \in \mathbb{Z}$
$\mathsf{e} = r$	$w_{\mathsf{e}} = w_r$	r 是一个引用， $w_r \in$ {无符号整数， 有符号整数， 异步复位， 复位， 时钟}
e = 多路复用器(c,e_1,e_2)	$w_{\mathsf{e}} =$ 最大值($w_{\mathsf{e}_1}, w_{\mathsf{e}_2}$)	$t_{\mathsf{e}_1} = t_{\mathsf{e}_2} \in$ {无符号整数,有符号整数}
e = 加法(e_1,e_2) 或 e = 减法(e_1,e_2)	$w_{\mathsf{e}} =$ 最大值($w_{\mathsf{e}_1}, w_{\mathsf{e}_2}$) $+ 1$	$t_{\mathsf{e}_1} = t_{\mathsf{e}_2} \in$ {无符号整数,有符号整数}
e = 乘法(e_1,e_2) 或 e = 拼接(e_1,e_2)	$w_{\mathsf{e}} = w_{\mathsf{e}_1} + w_{\mathsf{e}_2}$	$t_{\mathsf{e}_1} = t_{\mathsf{e}_2} \in$ {无符号整数,有符号整数}
e = 除法(e_1,e_2)	$w_{\mathsf{e}} = w_{\mathsf{e}_1}$	$t_{\mathsf{e}_1} = t_{\mathsf{e}_2} =$ 无符号整数
e = 除法(e_1,e_2)	$w_{\mathsf{e}} = w_{\mathsf{e}_1} + 1$	$t_{\mathsf{e}_1} = t_{\mathsf{e}_2} =$ 有符号整数
e = 取余(e_1,e_2)	$w_{\mathsf{e}} =$ 最小值($w_{\mathsf{e}_1}, w_{\mathsf{e}_2}$)	$t_{\mathsf{e}_1} = t_{\mathsf{e}_2} \in$ {无符号整数,有符号整数}
e = 填充(e_1, n)	$w_{\mathsf{e}} =$ 最大值(w_{e_1}, n)	$t_{\mathsf{e}_1} \in$ {无符号整数， 有符号整数}, $n \geqslant 0$
e = 相等(e_1,e_2) 或 e = 不等(e_1,e_2) 或 e = 小于(e_1,e_2) 或 e = 小于等于(e_1,e_2) 或 e = 大于(e_1,e_2) 或 e = 大于等于(e_1,e_2)	$w_{\mathsf{e}} = 1$	$t_{\mathsf{e}_1} = t_{\mathsf{e}_2} \in$ {无符号整数,有符号整数}
e =左移(e_1,n)	$w_{\mathsf{e}} = w_{\mathsf{e}_1} + n$	$t_{\mathsf{e}_1} \in$ {无符号整数,有符号整数}, $n \geqslant 0$
e =右移(e_1， n)	$w_{\mathsf{e}} =$ 最大值($w_{\mathsf{e}_1} - n$,0)	$t_{\mathsf{e}_1} =$ 无符号整数， $n \geqslant 0$
e =右移(e_1， n)	$w_{\mathsf{e}} =$ 最大值($w_{\mathsf{e}_1} - n$,1)	$t_{\mathsf{e}_1} =$ 有符号整数， $n \geqslant 0$
e = 动态左移(e_1,e_2)	$w_{\mathsf{e}} = w_{\mathsf{e}_1} + 2^{w_{\mathsf{e}_2}} - 1$	$t_{\mathsf{e}_1} \in$ {无符号整数,有符号整数}, $t_{\mathsf{e}_2} =$ 无符号整数
e = 动态右移(e_1,e_2) 或 e =逻辑非(e_1)	$w_{\mathsf{e}} = w_{\mathsf{e}_1}$	$t_{\mathsf{e}_1} \in$ {无符号整数， 有符号整数}， $t_{\mathsf{e}_2} =$ 无符号整数
e =类型转换(e_1)	$w_{\mathsf{e}} = w_{\mathsf{e}_1} + 1$	$t_{\mathsf{e}_1} =$无符号整数
e =类型转换(e_1)	$w_{\mathsf{e}} = w_{\mathsf{e}_1}$	$t_{\mathsf{e}_1} =$有符号整数
e =取负(e_1)	$w_{\mathsf{e}} = w_{\mathsf{e}_1} + 1$	$t_{\mathsf{e}_1} \in$ {无符号整数， 有符号整数}
e = 与(e_1,e_2) 或 e = 或(e_1,e_2) 或 e = 异或(e_1,e_2)	$w_{\mathsf{e}} =$ 最大值($w_{\mathsf{e}_1}, w_{\mathsf{e}_2}$)	$t_{\mathsf{e}_1} \in$ {无符号整数， 有符号整数} $t_{\mathsf{e}_1} = t_{\mathsf{e}_2} \in$ {无符号整数， 有符号整数}
e = 归约与(e_1,e_2) 或 e = 归约或(e_1,e_2) 或 e = 归约异或(e_1,e_2)	$w_{\mathsf{e}} = 1$	$t_{\mathsf{e}_1} \in$ {无符号整数， 有符号整数}
e = 位切片(e_1, hi, lo)	$w_{\mathsf{e}} = hi - lo + 1$	$t_{\mathsf{e}_1} \in$ {无符号整数,有符号整数}, $0 \leqslant lo \leq hi < w_{\mathsf{e}_1}$
e = 高位截取(e_1, n)	$w_{\mathsf{e}} = n$	$t_{\mathsf{e}_1} \in$ {无符号整数,有符号整数}, $0 \leqslant n \leqslant w_{\mathsf{e}_1}$
e = 低位截取(e_1, n)	$w_{\mathsf{e}} = w_{\mathsf{e}_1} - n$	$t_{\mathsf{e}_1} \in$ {无符号整数,有符号整数}, $0 \leqslant n \leqslant w_{\mathsf{e}_1}$
e =转换为无符号整数(e_1) 或 e =转换为有符号整数(e_1)	$w_{\mathsf{e}} = w_{\mathsf{e}_1}$	$t_{\mathsf{e}_1} \in$ {无符号整数,有符号整数}
e =转换为无符号整数(e_1) 或 e =转换为有符号整数(e_1)	$w_{\mathsf{e}} = 1$	$t_{\mathsf{e}_1} \in$ {异步复位,复位,时钟}

C 更多实验结果

此处列出了每个实例的运行时间，其中最后三个基准测试属于真实世界基准测试套件，其余所有基准测试均属于 MANUAL 基准测试套件。

基准测试	#cpnts	每个实例的时间（毫秒）		
		firtool	Gurobi	BFWInferWidths
逻辑	25	2.00	31.44	**0.12**
AddNot	1	2.00	9.93	**0.01**
CombElseWhen	1	1.90	11.87	**0.01**
条件	3	2.20	13.78	**0.02**
DrawMux6	7	1.50	20.99	**0.03**
ParamFunc	1	1.10	13.05	**0.01**
计时器	5	0.90	10.08	**0.02**
宽度规范	14	2.60	12.06	**0.03**
CombOther	1	2.90	7.41	**0.01**
CombWhen	1	1.80	9.56	**0.01**
CombWireDefault	1	0.70	16.97	**0.01**
spec.md-code-example-019	1	1.50	5.31	**0.01**
spec.md-code-example-020	1	0.40	6.12	**0.01**
spec.md-code-example-104	5	0.30	4.78	**0.01**
spec.md-code-example-106	5	2.70	16.26	**0.02**
1088	1	0.30	5.08	**0.02**
1089	1	0.40	5.17	**0.01**
1090_1	1	0.30	9.26	**0.01**
1090_2	1	0.30	8.28	**0.01**
1091	1	0.40	4.87	**0.01**
1092	1	0.40	4.52	**0.02**
1093	1	0.40	4.39	**0.01**
1094	1	0.40	4.42	**0.01**
1095	1	0.40	4.24	**0.01**
1096	1	0.40	4.26	**0.01**
1097	1	0.40	4.55	**0.01**
1098	1	0.30	5.21	**0.01**
1099	1	0.40	4.36	**0.01**
1108	1	0.30	4.35	**0.01**
1110	1	1.00	49.53	**0.01**
1112	1	0.90	5.63	**0.01**
1113	1	0.40	3.60	**0.01**
1115	1	0.40	4.63	**0.01**
1116	1	0.30	4.93	**0.01**
1118_1	1	1.60	9.99	**0.01**
1118_2	1	0.30	10.68	**0.01**
1271	3	1.10	4.42	**0.05**

基准测试	# 组件	每个实例的时间（毫秒）		
		firtool	Gurobi	BFWInferWidths
1491	1	0.30	4.26	**0.01**
2538	2	0.10	4.29	**0.01**
5391	4	0.60	4.63	**0.02**
向下计数器	5	0.90	10.71	**0.01**
向下计时器	5	0.70	7.60	**0.05**
闪烁器	24	0.70	8.56	**0.12**
闪烁器2	22	3.80	10.99	**0.15**
GCD	10	1.30	11.85	**0.19**
溢出类型电路	12	1.80	8.00	**0.04**
寄存器	2	0.50	9.89	**0.01**
时序电路	4	1.80	10.57	**0.02**
ShouldBeBadUInt SubtractWithGrow	4	1.10	8.24	**0.02**
test	1	1.50	10.23	**0.01**
最大公约数 2	10	2.10	5.89	**0.18**
同步复位	5	1.10	7.61	**0.02**
MultiClockSubModuleTest	18	1.70	9.53	**0.05**
VcdAdder	1	1.30	8.03	**0.01**
转发存储器	9	1.80	8.46	**0.04**
真双端口存储器	10	3.20	9.39	**0.03**
lower-memories	6	3.00	9.35	**0.02**
Rob	2,775	259.20	98.41	**12.05**
CoreTester	1,984	41.90	89.32	**8.12**
CoreSoc	3,315	83.20	85.70	**44.89**
设计的	4	/	10.53	**0.04**
设计的2	4	/	7.24	**0.07**
设计的3	4	/	6.15	**0.09**
设计的4	3	/	4.47	**0.08**
设计的5	3	/	4.61	**0.07**
设计的6	3	/	4.64	**0.03**
设计的7	4	/	5.03	**0.06**
设计8	4	/	6.47	**0.09**
设计9	5	/	5.79	**3.98**
设计10	4	/	4.38	**0.06**
设计11	5	/	3.98	**0.10**
6742	192	/	8.32	**1.06**
NutShell	7,152	190.70	194.55	**158.31**
Rocket Chip	4,882	127.90	120.64	**22.24**
RISC-V BOOM	205,608	8,338.30	**3,326.94**	3,467.80
平均值	**2,976.43**	**144.54**	**59.41**	**48.96**