arXiv:2603.19851v1 [cs.LO] 20 Mar 2026

# 适用于有界时间 MTL 的动态可重编程运行时监视器

Chirantan Hebballi[1], Akash Poptani[1], AmruthaBenny[2], Rajshekar Kalayappan[1*], SandeepChandran[2],Ramchandra Phawade[1]

[1]印度理工学院达尔瓦德分校，卡纳塔克邦，580011，印度。[2]印度理工学院帕拉卡德分校，喀拉拉邦，678623，印度。

*通讯作者。电子邮件：rajshekar.k@iitdh.ac.in；贡献作者：EE20BT013.alum24@iitdh.ac.in；EE20BT005.alum24@iitdh.ac.in；112004003@smail.iitpkd.ac.in；sandeepchandran@iitpkd.ac.in；prb@iitdh.ac.in；

**摘要**

支持在线、全速验证可动态变化（在现场运行期间）的性质的运行时验证（RV）框架，将使众多应用受益。若干先进的 RV 框架提出在 FPGA 上实现监视器。尽管这种方法能够支持在现场运行期间更改被监测的性质，但面对使用高性能处理器的被验证系统时，它们难以跟上其运行速度。在这项工作中，我们提出了一种新颖的可重编程监视器，采用标准单元而非 FPGA 实现。这使监视器能够与被验证系统共置于同一芯片上，因此适合对性质进行全速监测。我们提出的设计由一个实现五种基本操作的可编程单元以及一组队列更新规则组成。我们证明，这些可编程单元的组合能够忠实实现离散时间有界 MTL。我们通过仿真演示，所提出的监视器在部署后可通过其输入/输出引脚重新编程。一个相当大型的监视器可支持包含多达**16**个原子命题的 MTL 公式，占用空间仅为 $\mathbf{0.55}mm^{\mathbf{2}}$，同时以 $\mathbf{1.25}GHz$的频率运行。

**关键词：** 运行时验证（RV）、MTL监视器、动态可重编程硬件监视器、集成电路（IC）

# 1 引言

运行时验证（RV）通过为被验证系统（SUV）配备监视器来实现，该监视器持续观察系统的运行，并标记任何偏离预期行为的情况。当监视器具有非侵入性并以全速（在线）方式运行时，RV 尤为有效；在这种情况下，监视器会并行验证 SUV 的行为，并在数个时钟周期内发出判定结果，而不会影响 SUV 的时序（和功能）行为。当 SUV 是复杂的安全关键系统时，若监视器能够在 SUV 运行于现场期间随时间改变待验证的性质，则更为理想 [1–3]。

先前有多项工作通过在 FPGA 上实现监视器，实现了以工作速度进行验证和灵活性这两个目标 [1, 4–7]。与相应的软件监视器实现相比，FPGA 在提供更佳性能的同时，也实现了相近的灵活性。这是可行的，因为 FPGA 虽然是一种芯片，却是由查找表（LUT）、触发器（FF）和多路复用器（MUX）构成的。LUT 是简单的存储元件（异步静态随机存取存储器），用于存储布尔函数的所有可能输出。输入（或布尔函数的参数）用于索引该表并获取相应的输出。在重新配置FPGA 时，会将新值写入 LUT，从而使其能够在部署后更改功能。由于 LUT 可以按需反复写入，因此，用于根据输入规约生成软件监视器的综合过程也可以重新用于生成不同的硬件监视器。

这种方法有三个缺点。首先，与采用标准单元实现的相同电路相比，FPGA 实现的效率较低；如果在集成电路（IC）中实现，该电路便会采用标准单元。标准单元是与非门等基本门电路的一种实现形式，针对特定工艺节点进行了优化。例如，标准单元库中的双输入与非门单元具有三维布局描述，晶圆厂可据此制造并排列实现与非门功能的晶体管。因此，与采用 FPGA 相比，使用标准单元的设计延迟更低、吞吐量更高（以更高的时钟频率运行——能够每秒观察来自 SUV 的更多事件），功耗更低，且占用的面积更小。第二个缺点是，重新配置 FPGA 耗时较长，因为通常需要重写数千到数十万个 LUT 的内容。最后，即使在 Zynq Ultrascale+这类将 FPGA 与处理器集成在同一芯片上的先进片上系统（SoC） [8]中，FPGA 与处理器核之间可用的通信带宽仍然有限。适用于标准单元实现的灵活、可重新编程监视器则没有这些限制，因此能够对处理器核上运行的软件进行细粒度（每条指令）且以工作速度进行监控。

先前的多项研究在不同程度上缓解了这些缺点。针对重新配置时间增加这一缺点，人们提出了多种技术。一种方法是将 MTL 算子实现为固定模块（为每个算子预先实现指定数量的实例），并适当地将输入流路由经过这些模块，以监视属性 [1]。另一种技术是利用输入属性之间共有的子表达式，以尽量减少

所需资源，从而也减少所需的重新配置量 [9]。最后，还可以使用部分重配置。当前最先进的现场可编程门阵列支持这种技术，只需重新配置少量查找表，而非全部查找表 [10]。通过将验证范围限制为总线事务，减少传输到监视器的数据量，可以缓解处理器与 FPGA 逻辑阵列之间通信延迟增加这一缺点。由于总线事务的数量至少比寄存器文件上的活动（每条指令）少一个数量级，因此这种做法能够减少数据量 [6]。然而，这需要在更粗粒度的（时间）尺度上重新定义属性。

在本工作中，我们通过综合可用标准单元实现的硬件监视器，克服了使用现场可编程门阵列的全部三项缺点。这些监视器可以与处理器集成，从而实现细粒度、全速率监测。这需要采用一种设计方法，通过动态可重编程性而非重新配置来实现灵活性 [11]。在这种方法下，电路是固定的，并通过输入指令（字节序列）在部署后配置监视器的运行方式。这种方法有诸多优点。运行频率或监测电路所占面积等性能参数不会随输入性质而改变。此外，由于监视器的电路设计独立于输入性质，且在设计时已知，因此我们可以根据需要重新设计电路（例如加入流水线），使其运行频率与 SUV 的频率相匹配。这确保监视器能够跟上 SUV 的运行速度（每个周期处理一个输入符号）。

我们首先设计一种抽象机 （AM），它可编程以执行五种简单操作之一：与、或、非、蕴含和导线（恒等函数），并根据所执行操作的结果更新自定义队列。随后，我们证明这些AM可编程为实现所有 度量时序逻辑（MTL） [12] 算子的求值机（EM）。最后，我们利用可编程互连将多个求值机组合起来，为给定的MTL公式构建监视器。求值机的这种组合由公式 [1, 3]的抽象语法树（AST）决定。向求值机输入事件流之前，会对每个抽象机要执行的具体操作以及互连的转发规则进行编程。我们提出的框架包含参数化硬件设计，以及用于生成编程位、以适当配置抽象机和互连的编译器。实验表明，即使是大型MTL监视器（支持多达16个原子命题，且MTL算子的最大时间界为256个时间步），占用的面积也仅为 $0.55mm^2$ ，使用保守的 $32nm$ 标准单元库实现时，时钟频率可达1.25 $GHz$ 。这意味着，所开发的监视器每秒能够观测来自SUV的 $1.25 \times 10^9$ 个事件，显著超过基于FPGA的监视器所能达到的水平。此外，对于所有可用16个原子命题、16个MTL算子以及最大时间界为256个时间步表示的MTL公式，该监视器均能实现这种高效率——高吞吐量和小面积。所提出的工具已在线发布 [13]。

本文其余部分安排如下。第2节讨论背景和相关工作。第3节介绍所提出的可编程MTL监视器的设计。第4 节概述一种适于使用标准单元进行综合的实现方案，并介绍监视器的编程细节。第5 节介绍所开发的工具 [13]以及对不同规模MTL监视器进行综合的结果。最后，我们将在第6节总结全文。

# 2 预备知识与相关工作

首先，我们介绍度量时序逻辑的语法和语义。

## 2.1 度量时序逻辑 (MTL)

在任意时间点对 SUV 进行的一次观测称为一个事件。一个事件由一组布尔变量的赋值组成。布尔变量集合构成了定义所需属性的原子命题（AP）集合。因此，我们可以将事件 $e \in 2^{AP}$正式表述为。SUV 会被定期观测，因此，SUV 的一次运行或执行是由事件构成的无限序列 $e_0, e_1, e_2, \ldots$，其中 $e_i$指在时间步 $i$进行的观测。换句话说，SUV 的一次运行可以表示为函数 $\mathbb{N}_0 \to 2^{AP}$，其中 $\mathbb{N}_0$ 是自然数集合 $\{0, 1, 2, \ldots\}$。

**语法**：如果 $\sigma$ 是任意原子命题，且 $\phi$和 $\psi$ 是有效的 MTL 公式，那么以下表达式也都是有效的 MTL 公式：

$$true \mid \sigma \mid \neg\phi \mid \phi \vee \psi \mid \phi \wedge \psi \mid \phi \to \psi \mid \mathcal{X}\phi \mid \phi\, \mathcal{U}_{[t_1,t_2]}\psi \mid \Box_{[t_1,t_2]}\phi \mid \Diamond_{[t_1,t_2]}\phi$$

时间界限以区间的形式指定：我们用 $[t_1, t_2]$ 表示集合$\{i \mid t_1 \leq i \leq t_2$，其中 $i$, $t_1$， $t_2 \in \mathbb{N}_0\}$。我们将讨论范围限制在有界语义内，也就是说， $t_2$ 不能为无穷大。离散时间有界语义非常适合表达许多实时系统规范 [1, 3, 14–16]。

**语义**：我们归纳定义 MTL公式 $\phi$ 在时间 $i$ 对执行$x$的满足关系，记作 $x^i \vDash \phi$，如下所示：

- $x^i \vDash true$ 为真，
- $x^i \vDash \sigma$ 当且仅当 $\sigma$ 在 $e_i$中成立，
- $x^i \vDash \neg\phi$ 当且仅当 $x^i \nvDash \phi$，
- $x^i \vDash \phi \vee \psi$ 当且仅当 $x^i \vDash \phi \vee x^i \vDash \psi$，
- $x^i \vDash \phi \wedge \psi$ 当且仅当 $x^i \vDash \phi \wedge x^i \vDash \psi$，
- $x^i \vDash \phi \to \psi$ 当且仅当 $x^i \nvDash \phi \vee x^i \vDash \psi$，
- $x^i \vDash \mathcal{X}\phi$ 当且仅当 $x^{i+1} \vDash \phi$，
- $x^i \vDash \phi\, \mathcal{U}_{[t_1,t_2]}\psi$当且仅当 $\exists j \in [i + t_1, i + t_2] : (x^j \vDash \psi \wedge (\forall k \in [i, j) : x^k \vDash \phi))$
- $x^i \vDash \Box_{[t_1,t_2]}\phi$ $\forall j \in [i + t_1, i + t_2]$ $x^j \vDash \phi$当且仅当：当且仅当：
- $x^i \vDash \Diamond_{[t_1,t_2]}\phi$ $\exists j \in [i + t_1, i + t_2]$ $x^j \vDash \phi$

**定义 1.**（运行时验证问题）。给定一个关于原子命题集合 $AP$ 的 MTL公式 $\phi$，以及一个关于 $2^{AP}$ 的词 $x$，运行时验证问题是构造一个关于 $\{true(\top)\ false(\bot)\}$ 的词 $x'$，使得 $x'(i) = \top$ 当且仅当 $x^i \vDash \phi$。

词语 $x$对应于来自 SUV 的观测流，而词语$x'$ 对应于判定流。

给定一个 MTL公式 $\phi$，其大小 $|\phi|$ 定义为其中算子出现的总次数。我们能得出 $\phi$ 的判定结果所需的最少时间步数称为其时域，记作 $H(\phi)$。其递归定义如下:

- $H(true) = H(\sigma) = 0, \forall \sigma \in AP$,
- $H(\neg\phi) = 1 + H(\phi)$,
- $H(\phi \vee \psi) = H(\phi \wedge \psi) = H(\phi \rightarrow \psi) = 1 + max(H(\phi)\ H(\psi))$,
- $H(\mathcal{X}\phi) = 2 + H(\phi)$,
- $H(\phi\ \mathcal{U}_{[t_1,t_2]}\psi) = t_2 + 1 + max(H(\phi)\ H(\psi))$
- $H(\Box_{[t_1,t_2]}\phi) = H(\diamond_{[t_1,t_2]}\phi) = t_2 + 1 + H(\phi)$

我们定义了一个从 MTL 算子到 $\mathbb{N}_0$的映射 $l$： $l(\neg) = l(\wedge) = l(\vee) = l(\rightarrow) = 1$； $l(\mathcal{X}) = 2$； $l(^{\diamond_{[t_1,t_2]}}) = l(\Box_{[t_1,t_2]}) = l(\ \mathcal{U}_{[t_1,t_2]}) = t_2 + 1$。
接下来，我们将介绍以往针对 MTL规约运行时验证（RV）的方法。

## 2.2 相关工作

以往研究构建度量时序逻辑监视器主要采用三种方法。第一种方法是构造一个包含 $H$（$\phi$）行和 $|\phi|$ 列的表格，用于表示规约 $\phi$ [2, 15, 17–19]。单元格（$i$, $j$）的内容给出子公式 $j$ 在时间 $k - i$的判定结果，该结果计算截至时间 $k$。截至时间 $k$计算出的这一判定结果不一定是最终判定结果，随着时间推移以及有关 SUV 的更多观测数据可用，它可能发生变化。在每个时间步，当监视器接收到新事件时，表格中最早的一行会被移除，表格内容向上移动一行，为新接收的事件腾出位置。随后，利用新获得的原子命题赋值，以自底向上的方式求解输入公式，并更新表中不同单元格的值。最早一行中的子公式保证具有最终判定结果（由 $H$（$\phi$）可知）。一些研究通过以下方式进一步改进了这一方法：(i) 不再重新处理之前已经处理过的值；(ii) 在单元格值无法利用现有观测数据计算时，重写公式，使单元格值的计算仅依赖未来的值。该方法通常需要 $O$（$H \times |\phi|$）空间，并且延迟为 $O$（$H \times |\phi|$）。吞吐量通常为$_{H\times|\phi|}^{1}$，因为对于每个新观测，都需要花费 $O$（$H \times |\phi|$）时间对其进行处理。这种方法在软件监视器中较为常见。——

构建 MTL 监视器的另一种方法是构建输入公式的抽象语法树（AST），并利用由此生成的树结构，推导出以流式方式运行的高吞吐量监视器（输入值仅读取一次；不进行回溯） [1, 3, 20, 21]。这些工作会为树中的每个节点（对应输入规约中的一个 MTL 算子）实例化一个求值器。求值器模块 $c$ 产生的结果会传递给求值器模块 $p$，该模块

表示 AST 中 $c$ 的父节点。在每个时间 $i$，在时间 $i$ 接收到的事件会从叶节点进入树，并在随后的时间步中向根节点传播。时间 $i$ 对应事件的判定结果会在时间 $i+O(H)$ 到达根节点。吞吐量为 $\frac{1}{l_{max}}$，其中 $l_{max}$ 是树中最复杂的求值器模块的延迟。我们的工作也采用了类似方法，其中 AST 中的每个节点都对应一个可编程抽象机。早期工作已采用此方法在 FPGA 上实现 MTL 监视器 [1, 3]。这些工作还强调了在现场支持输入规约变更（即可编程性）的重要性 [1]。然而，与我们的方法不同，这些工作会为每个 MTL 算子创建单独的求值器模块，在 FPGA 上实例化多个求值器模块，并重新配置这些求值器之间的连接，以实现针对不同 MTL 公式的监视器。相比之下，我们的求值器模块（或抽象机）能够求值所有 MTL 算子，因此，只需启用适当的操作和/或选择适当的结果，即可实现针对不同 MTL 公式的监视器。

第三种方法是采用定时自动机来描述度量时序逻辑规约 [22,23]。然而，据我们所知，目前尚无可编程的定时自动机处理器。因此，只能通过软件或 FPGA 实现可编程性，但这些实现存在第1节中讨论的缺点。

一些研究探索了采用存内计算和神经形态计算等新型计算范式来实现高效监视器 [20, 24]。这些研究与我们提出的工作相互独立，因为商用片上系统 (SoC) 尚未将此类加速器集成到同一芯片上，而这对于我们实现全速监控至关重要。

一些研究人员探讨了度量时序逻辑定量语义的运行时监控 [25–27]。这不仅涉及判断给定性质是否得到满足，还涉及确定 SUV 执行的鲁棒性，即 SUV 距离违反该性质有多近。在这项工作中，我们仅关注第2.1节所述的定性语义。

# 3 用于 MTL 公式的抽象机

我们提出的 MTL 监视器的基本构件是一个抽象机（AM），相关描述见第3.1节。大小为 1 的 MTL 公式的运行时验证监视器称为求值机（EM）。在第3.2节中，我们将说明如何通过编程一个或多个抽象机来实现每个 MTL 算子的 EM。随后在第3.3节中，我们将说明如何根据公式的抽象语法树，将公式中各个算子的 EM 组合起来，构造大小大于 1 的 MTL 公式的运行时验证监视器。

## 3.1 AM 及其操作

在每个时间步，抽象机消耗一个或两个输入操作数，并产生一个输出判定。输入操作数和输出属于 $\{true$（$\top$） $false$（$\bot$）$\}$。

抽象机会修改一个有限队列，该队列用于存放生成判定结果时产生的部分结果。我们将这种结构称为队列。队列中的每个单元格可以存储：（i）真（$\top$）、（ii）假（$\bot$）或（iii）可能（$\mathcal{M}$）。令 $\Gamma = \{\top, \bot, \mathcal{M}\}$ 表示该队列

的字母表。令 $Q^i$ 表示 $Q$ 在时间实例 $i$的内容。令 $Q^i[k]$ 表示 $Q$在时间实例 $i$ 时第 $k$个单元格的内容（$Q[k]$ 表示当前时间实例中第 $k$个单元格的内容）。令 $Q[0]$ 表示 $Q$的尾部。除了加法和删除这两种基本操作外， $Q$ 还支持条件 $modify$ 操作。下面定义这些操作。令区间表示$\mathbb{N}_0$上的所有区间组成的集合。

队列 $Q$ 上的基本操作 $QOps$ 描述如下：

- $add(Q)$将队列内容推向索引较大的单元，并插入元素 $\mathcal{M}$，即 $Q[0] = \mathcal{M}$。
- $del(Q, Head)$ 从队列中删除元素 $Q[Head]$ 。
- $modify(Q, \mathcal{I}, \top_{\mathcal{M}})$ 有条件地修改队列内容：对于区间 $\mathcal{I}$中的所有 $t$，如果$Q[t] = \mathcal{M}$，则 $Q[t] = \top$
- $modify(Q, \mathcal{I}, \bot_{\mathcal{M}})$ 有条件地修改队列内容：对于区间 $\mathcal{I}$中的所有 $t$，如果$Q[t] = \mathcal{M}$，则 $Q[t] = \bot$

抽象机是一个单一状态机，因此为简洁起见，我们在描述中省略对状态的提及。机器的输入可分为两类：程序输入和操作数输入。

抽象机的程序输入为：

- $opcode$ 是 $OPCODES = \{not, and, or, implies, wire\}$的一个元素
- $Q$ 是一个队列， $Head$是 $Q$的索引
- $\mathcal{I}_\top$ 和 $\mathcal{I}_\bot$ 是 $Intervals$的元素
- $Mod_\top$和 $Mod_\bot$是 $\{\top, \bot\}$的元素

抽象机的程序输入只在编程时发生变化。在其他所有时间步中，它们都保持不变。

操作数输入 $op_0$， $op_1$ 是 $\{\top, \bot\}$ 的元素，并且每一步都可能发生变化。

抽象机在每一步的操作如下。机器首先计算 $res$：

$$res = \begin{cases} \neg op_0 & \text{if } opcode = not \\ op_0 \wedge op_1 & \text{if } opcode = and \\ op_0 \vee op_1 & \text{if } opcode = or \\ op_0 \to op_1 & \text{if } opcode = implies \\ op_0 & \text{if } opcode = wire \end{cases}$$

然后，它对 $Q$ 执行一系列操作：

- 如果 $res = \top$与 $Mod_\top = \top$那么 $add(Q)$; $modify(Q, \mathcal{I}_\top, \top_{\mathcal{M}})$; $del(Q, Head)$;
- 如果 $res = \bot$与 $Mod_\bot = \top$那么 $add(Q)$; $modify(Q, \mathcal{I}_\bot, \bot_{\mathcal{M}})$; $del(Q, Head)$;
- 如果 $res = \top$与 $Mod_\top = \bot$那么 $add(Q)$; $del(Q, Head)$;
- 若 $res = \bot$与 $Mod_\bot = \bot$，则 $add(Q)$; $del(Q, Head)$;

## 3.2 MTL 运算符求值机的实现

用于 MTL算子的求值机（EM）也是大小为一的 MTL 公式的监视器。设公式为 $\phi$，且其中的单算子为 $\odot$。EM 有一个或两个输入流以及一个输出判定流。在任意给定的时间步，输入操作数的值为 $\alpha_0$ 与 $\alpha_1$。一个 EM 由一个或多个 AM 实例和一个 Que 组成。从相关 Que 中删除的值构成监测判定流。在任意时刻 $i$，第 $k$ 个队列单元的内容给出对应于时间 $(i-k)$ 的判定结果。如果 $k \geq l(\odot)$，则判定结果是稳定的（注意，由于 $l(\odot) = H(\phi)$，因为 $|\phi| = 1$）。否则，该判定结果可能在未来的时间步发生变化。因此，AM 实例中 $Head$ 的值始终选择为大于或等于 $l(\odot)$。

**表1**：使用抽象机实现与不同 MTL 运算符对应的监视器

| | **AM 编程** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **MTL 算子** | **操作码** $op_0$ | | $op_1$ | **Que ID** | **最小值 头部的** | $\mathcal{I}_\top$ | $\mathcal{I}_\bot$ | $Mod_\top$ | $Mod_\bot$ |
| $\neg\alpha_0$ | not | $\alpha_0$ | - | $Q$ | 1 | $[0,0]$ | $[0,0]$ | $\top$ | $\top$ |
| $\alpha_0 \vee \alpha_1$ | or | $\alpha_0$ | $\alpha_1$ | $Q$ | 1 | $[0,0]$ | $[0,0]$ | $\top$ | $\top$ |
| $\alpha_0 \wedge \alpha_1$ | and | $\alpha_0$ | $\alpha_1$ | $Q$ | 1 | $[0,0]$ | $[0,0]$ | $\top$ | $\top$ |
| $\alpha_0 \rightarrow \alpha_1$ | 蕴含 | $\alpha_0$ | $\alpha_1$ | $Q$ | 1 | $[0,0]$ | $[0,0]$ | $\top$ | $\top$ |
| $\mathcal{X}\alpha_0$ | wire | $\alpha_0$ | - | $Q$ | 2 | $[1,1]$ | $[1,1]$ | $\top$ | $\top$ |
| $\Box_{[t_1,t_2]}\alpha_0$ | wire | $\alpha_0$ | - | $Q$ | $t_2+1$ | $[t_2,t_2]$ | $[t_1,t_2]$ | $\top$ | $\top$ |
| $\Diamond_{[t_1,t_2]}\alpha_0$ | wire | $\alpha_0$ | - | $Q$ | $t_2+1$ | $[t_1,t_2]$ | $[t_2,t_2]$ | $\top$ | $\top$ |
| $\alpha_0\ \mathcal{U}_{[t_1,t_2]}\alpha_1$ | wire<br>wire<br>or | $\alpha_0$<br>$\alpha_1$<br>$\alpha_0$ | -<br>-<br>$\alpha_1$ | $Q$<br>$Q$<br>$Q$ | $t_2+1$<br>$t_2+1$<br>$t_2+1$ | $[0,0]$<br>$[t_1,t_2]$<br>$[0,0]$ | $[0,t_1-1]$<br>$[t_2,t_2]$<br>$[t_1,t_2-1]$ | $\bot$<br>$\top$<br>$\bot$ | $\top$<br>$\top$<br>$\top$ |
| $\alpha_0\ \mathcal{U}_{[0,t_2]}\alpha_1$ | or<br>wire | $\alpha_0$<br>$\alpha_1$ | $\alpha_1$<br>- | $Q$<br>$Q$ | $t_2+1$<br>$t_2+1$ | $[0,0]$<br>$[0,t_2]$ | $[0,t_2-1]$<br>$[t_2,t_2]$ | $\bot$<br>$\top$ | $\top$<br>$\top$ |

表1列出了如何通过 AM 实例实现不同 MTL 算子的 EM。我们设计的 EM 能够对给定的每个输入操作数仅处理一次，从而构建出完全流式的监视器。当前时间步的输入操作数对过去时间步对应判定结果的影响，会被适当地记录在结果队列中。

我们通过两个示例展示 EM 的实现及其工作方式。

**示例1**。 $EM$针对 $\neg$是一个 AM $M = (opcode = \neg, Q, Head = 1, \mathcal{I}_\top = [0,0], \mathcal{I}_\bot = [0,0], Mod_\top = \top, Mod_\bot = \top)$，如表 1所示。该机器的操作遵循抽象机的定义，并在表2中进行了说明。

当 $\alpha_0 = \top$ 时，我们有 $res = \bot$（参见表 2 中 $time = 0$ 对应的行）。由于我们有$Mod_\bot = \top$，因此对 $Q$ 执行以下操作： $add(Q, \mathcal{M})$; $modify(Q, [0,0], \bot_\mathcal{M})$;$del(Q,1)$;。由于位置 1 处没有内容，$del$ 操作不会执行任何操作。

在时间=1， $\alpha_0 = \bot$，因此我们有 $res = \top$。由于我们有 $Mod_\top = \top$，我们对 $Q$ 执行以下操作： $add(Q, \mathcal{M})$; $modify(Q, [0,0], \top_\mathcal{M})$; $del(Q, 1)$;。被删除的

**表2**：($\neg\alpha_0$) 的监视器工作示例

| time | $\alpha_0$ | $\alpha_1$ | 结果 | 队列修改 | 队列内容 | | | | | $verdict$ |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | [<区间>]: <值> | | 0 | 1 | 2 | 3 | |
| t = 0 | $\top$ | - | $\bot$ | [0,0]: $\bot_{\mathcal{M}}$ | $add$之后<br>$modify$之后<br>$del$之后 | $\mathcal{M}$<br>$\bot$<br>$\bot$ | | | | |
| t = 1 | $\bot$ | - | $\top$ | [0,0]: $\top_{\mathcal{M}}$ | $add$之后<br>$modify$之后<br>$del$之后 | $\mathcal{M}$<br>$\top$<br>$\top$ | $\bot$<br>$\bot$ | | | $r^0 = \bot$ |
| t = 2 | $\bot$ | - | $\top$ | [0,0]: $\top_{\mathcal{M}}$ | $add$之后<br>在 $modify$ 之后<br>在 $del$ 之后 | $\mathcal{M}$<br>$\top$<br>$\top$ | $\top$<br>$\top$ | | | $r^1 = \top$ |

**表3**：展示 ($\alpha_0\ \mathcal{U}_{[1,2]}\alpha_1$) 的监视器运行情况的示例

| time | $\alpha_0$ | $\alpha_1$ | res | 队列修改 | 队列内容 | | | | | $verdict$ |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | [<区间>]: <值> | | 0 | 1 | 2 | 3 | |
| t = 0 | $\bot$ | $\bot$ | $\bot, \bot, \bot$ | [0,0]: $\bot_{\mathcal{M}}$, [2,2]: $\bot_{\mathcal{M}}$, [1,1]: $\bot_{\mathcal{M}}$ | 在 $add$ 之后<br>在 $modify$ 之后<br>在 $del$ 之后 | $\mathcal{M}$<br>$\bot$<br>$\bot$ | | | | |
| t = 1 | $\top$ | $\bot$ | $\top, \bot, \top$ | [2,2]: $\bot_{\mathcal{M}}$ | 在 $add$ 之后<br>在 $modify$ $\mathcal{M}$ 之后<br>在 $del$ 之后 | $\mathcal{M}$<br>$\mathcal{M}$ | $\bot$<br>$\bot$ | | | |
| t = 2 | $\top$ | $\bot$ | $\top, \bot, \top$ | [2,2]: $\bot_{\mathcal{M}}$ | 之后 $add$<br>之后 $modify$<br>之后 $del$ | $\mathcal{M}$<br>$\mathcal{M}$<br>$\mathcal{M}$ | $\mathcal{M}$<br>$\mathcal{M}$<br>$\mathcal{M}$ | $\bot$<br>$\bot$<br>$\bot$ | | |
| t = 3 | $\bot$ | $\top$ | $\bot, \top, \top$ | [0,0]: $\bot_{\mathcal{M}}$, [1,2]: $\top_{\mathcal{M}}$ | 之后 $add$<br>在 $modify$ 之后<br>在 $del$ 之后 | $\mathcal{M}$<br>$\bot$<br>$\bot$ | $\mathcal{M}$<br>$\top$<br>$\top$ | $\mathcal{M}$<br>$\top$<br>$\top$ | $\bot$<br>$\bot$ | $r^0 = \bot$ |
| t = 4 | $\top$ | $\top$ | $\top, \top, \top$ | [1,2]: $\top_{\mathcal{M}}$ | 在 $add$ 之后<br>在 $modify$ $\mathcal{M}$ 之后<br>在 $del$ 之后 | $\mathcal{M}$<br>$\mathcal{M}$ | $\bot$<br>$\bot$ | $\top$<br>$\top$<br>$\top$ | $\top$<br>$\top$ | $r^1 = \top$ |

值构成对应于时间 $(current_time - Head) = (1-1) = 0$ 的判定结果，即， $r^0 = \bot$ （我们将对应于时间 $i$ 的判定结果记为 $r^i$）。类似地，在时间=2，对应于时间 1 ($r^1 = \top$) 的判定结果由监视器输出。因此，我们在每个时间步都会得到一个新的判定结果。

**例2.** $EM$ 对于 $\mathcal{U}_{[1,2]}$ 是由三个在同一队列上运行的 AM 组成的集合 $Q$如表 1所示。这台机器的操作由 AM 的定义得出，并在表 3中进行了说明。

在每个时间步，三个抽象机分别执行给定操作（（$wire\ \alpha_0$）、（$wire\ \alpha_1$）和（$\alpha_0\ or\ \alpha_1$））。所得的三个结果分别标示在

$res$ 表3中的列。在时间步0，第一个抽象机计算出 $res = \perp$，由于其 $Mod_{\perp} = \top$，它将索引位于范围 $[0, (t_1 - 1)] = [0, 0]$ 内的 $Q$ 个单元修改为 $\perp$，因为该单元的旧值为 $\mathcal{M}$。类似地，第二个抽象机尝试将单元2 修改为 $\perp$（如果 $\mathcal{M}$），第三个抽象机尝试将单元1修改为 $\perp$（如果 $\mathcal{M}$）。但由于这些单元为空，因此 $Q$没有变化。

在时间步1，第一个和第三个抽象机不会修改 $Q$，因为它们的 $res$ 值为$\top$，而它们的 $Mod_{\top}$ 值为 $\perp$。然而，第二个抽象机计算出 $res = \perp$，且由于其 $Mod_{\perp} = \top$，它尝试将 $Q$ 单元格 $t_2 = 2$ 修改为 $\perp$（如果 $\mathcal{M}$）。然而，由于该单元格为空，$Q$ 保持不变。后续时间步中，$Q$ 的更新以类似的方式进行。

在时间步3，与时间$(current_time - Head) = 0$对应的判定结果，即 $r^0 = \perp$，由监视器输出。同样，在时间步4，与时间1 $(r^1 = \top)$对应的判定结果被输出。

### 3.2.1 求值机的正确性

预期是在任意时间 $i$，对于所有 $l(\odot) \leq k \leq i$（假设 $k < Head$），单算子公式 $\phi$在时间 $i-k$ 对应的判定结果都存在于单元格 $Q^i[k]$中。我们通过两个定理来描述这一正确性要求。定理 1指出，求值机在时间 $i$会在单元格$Q^i[l(\odot)]$, 中正确计算出公式 $\phi$ 在时间$i - l(\odot)$对应的判定结果。定理2指出，一旦给出判定结果，只要它仍存在于 $Q$中，就不会被修改。定理 1和定理2的证明见附录。

**定理 1.** 设 $\phi$ 是一个单算子度量时序逻辑公式，$\odot$ 是其中的算子。则对于 $\phi$的监视器，对所有时间实例 $i$，满足 $i \geq l(\odot)$，

$$if\ x^{i-l(\odot)} \vDash \phi,\ then\ Q^i[l(\odot)] = \top.$$

$$if\ x^{i-l(\odot)} \nvDash \phi,\ then\ Q^i[l(\odot)] = \perp.$$

**定理2.** 设 $\phi$ 是一个单算子度量时序逻辑公式，且 $\odot$ 是其中的算子。则对于 $\phi$的监视器，对所有满足 $j \geq l(\odot)$的 $j$，以及所有满足 $1 \leq k \leq (Head - l(\odot))$的 $k$，$Q^{(j+k)}[l(\odot) + k] = Q^j[l(\odot)]$。

由于我们使用 $Head \geq l(\odot)$，根据定理 1 和定理 2，我们得到以下推论，从而确立求值机的正确性。

**推论 1.** 对于给定的度量时序逻辑公式 $\phi$和输入词 $x$，队列中已删除值流给出判定流 $x'$，使得 $x'(i) = \top$ 当且仅当 $x^i \vDash \phi$。

**命题 1.** 在针对 $\mathcal{U}_{[t_1,t_2]}$ 算子的监视器中，修改单个队列 $Q$的三个 AM 不会同时（在同一时间步）修改 $Q$ 的相同单元。

$\mathcal{U}_{[0,t_2]}$ 算子同样适用。证明见附录。

值得注意的是，所有 AM 都是同构的，任何 MTL 算子的监视器都可以使用任意 AM 来实现。这种高度灵活性与典型的 FPGA 方法形成对比：后者为每个 MTL 算子实现预先定义数量的实例，并在运行时配置它们之间的连接，以改变所监视的性质。这种 FPGA 方法限制了可处理的 MTL 公式数量。

接下来，我们将描述如何实现大小大于1 的 MTL 公式的监视器。

## 3.3 MTL 公式的监测器

为了评估任意给定的 MTL 公式，我们构造其抽象语法树（AST）。对于公式中的每个算子，分别由一个 EM 负责对其进行评估。根据 AST，某些 EM 的输出连接到其他 EM 的输入。如果算子 $x$ 是 AST 中算子 $y$ 的子节点，则与 $y$ 对应的 EM（$EM_y$）必须从与 $x$ 对应的 EM（$EM_x$）的队列中读取其源操作数。此外，$EM_y$ 必须从 $EM_x$ 的队列中的正确位置读取数据，以维持功能正确性。这意味着必须正确设置 $EM_x$ 的 $Head$ 。

请注意，对于单算子，函数 $l$ 和 $H$ 是相同的。根据定理 1，对于每个 $EM_x$ ，$Head$的值应至少为 $l(x)$。然而，在某些情况下，为了维持功能正确性，$Head_x$ 必须大于 $l(x)$ ，如示例3所示。

**例 3.** 考虑公式 $\diamond_{[0,1]}\neg\sigma_1 \vee \diamond_{[1,4]}\sigma_2$。该公式对应的抽象语法树如图 1(a) 所示，算子分配至各监视器的情况如图 1(b) 所示。

表4 展示了两种情形下不同队列单元格中的值。$r_i^j$ 表示由 $EM_i$ 计算得出的、对应于时间 $j$的判定结果。在第一种情形中，每个 $EM$ 的 $Head$ 都设为对应算子的 $l$ () 值。由此得到 $Head_1 = 1$、$Head_2 = 2$、$Head_3 = 5$和 $Head_4 = 1$。在时间步2，$EM_2$ 从单元格 $Q_4^1[1]$取入判定结果$r_4^0$，并开始在单元格 $Q_2^2[0]$中计算判定结果 $r_2^0$。同样，在时间步6，$EM_1$ 对单元格 $Q_2^5[2]$ 和 $Q_3^5[5]$的内容执行 $\vee$ 操作。然而，这些内容分别对应于时间1 的 $EM_2$ 的判定结果（$r_2^1$），以及时间0 的 $EM_3$ 的判定结果（$r_3^0$）。这种时间上的差异在功能上是不正确的；在后续每个周期中，$EM_1$ 的源操作数对应的时间也存在类似的不匹配。

我们进一步深入探究这一功能错误的原因。如表4中的示例所示，$EM_4$ 需要2 个时间步才能得出判定结果（判定结果会经过 $Q_4[0]$与 $Q_4[1]$）。同样，$EM_2$ 需要3个时间步，而 $EM_3$ 需要6个时间步。这意味着，对于时间 $i$ ，与 $EM_1$左子树对应的判定结果会在时间 $i+2+3=i+5$时于 $EM_1$ 处可用，而右子树的判定结果则会在时间 $i+6$时可用。这种不匹配导致了功能错误。

我们可以通过设置 $Head_2 = 3$来纠正这一点，正如表 4中第二种情形所做的那样。现在，在时间6，$Q_2^5[3]$ 包含 $r_2^0$ （与 $Q_2^4[2]$保持不变，根据定理 2），

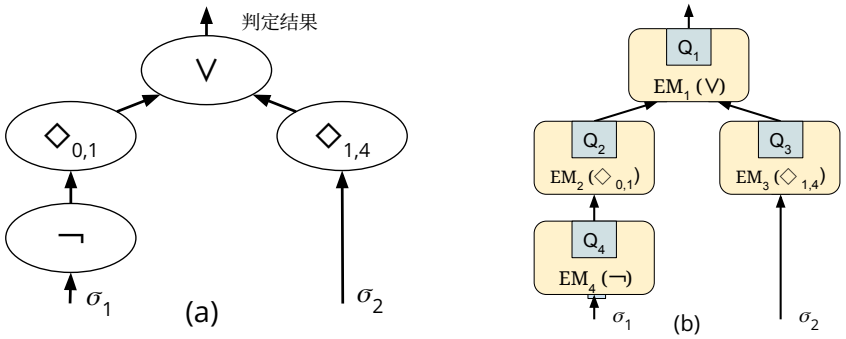



**图1**：抽象语法树与 EM for $\diamond_{[0,1]}\neg\sigma_1 \vee \diamond_{[1,4]}\sigma_2$

| **场景1：** $Head_2 = 2$ | | | | | | | | | **场景2：** $Head_2 = 3$ | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| time | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | time | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| $Q_4[0]$ | $r_4^0$ | $r_4^1$ | $r_4^2$ | $r_4^3$ | $r_4^4$ | $r_4^5$ | $r_4^6$ | $r_4^7$ | $Q_4[0]$ | $r_4^0$ | $r_4^1$ | $r_4^2$ | $r_4^3$ | $r_4^4$ | $r_4^5$ | $r_4^6$ | $r_4^7$ |
| $Q_4[1]$ | | $r_4^0$ | $r_4^1$ | $r_4^2$ | $r_4^3$ | $r_4^4$ | $r_4^5$ | $r_4^6$ | $Q_4[1]$ | | $r_4^0$ | $r_4^1$ | $r_4^2$ | $r_4^3$ | $r_4^4$ | $r_4^5$ | $r_4^6$ |
| $Q_2[0]$ | | | $r_2^0$ | $r_2^1$ | $r_2^2$ | $r_2^3$ | $r_2^4$ | $r_2^5$ | $Q_2[0]$ | | | $r_2^0$ | $r_2^1$ | $r_2^2$ | $r_2^3$ | $r_2^4$ | $r_2^5$ |
| $Q_2[1]$ | | | | $r_2^0$ | $r_2^1$ | $r_2^2$ | $r_2^3$ | $r_2^4$ | $Q_2[1]$ | | | | $r_2^0$ | $r_2^1$ | $r_2^2$ | $r_2^3$ | $r_2^4$ |
| $Q_2[2]$ | | | | | $r_2^0$ | $r_2^1$ | $r_2^2$ | $r_2^3$ | $Q_2[2]$ | | | | | $r_2^0$ | $r_2^1$ | $r_2^2$ | $r_2^3$ |
| | | | | | | | | | $Q_2[3]$ | | | | | | $r_2^0$ | $r_2^1$ | $r_2^2$ |
| $Q_3[0]$ | $r_3^0$ | $r_3^1$ | $r_3^2$ | $r_3^3$ | $r_3^4$ | $r_3^5$ | $r_3^6$ | $r_3^7$ | $Q_3[0]$ | $r_3^0$ | $r_3^1$ | $r_3^2$ | $r_3^3$ | $r_3^4$ | $r_3^5$ | $r_3^6$ | $r_3^7$ |
| $Q_3[1]$ | | $r_3^0$ | $r_3^1$ | $r_3^2$ | $r_3^3$ | $r_3^4$ | $r_3^5$ | $r_3^6$ | $Q_3[2]$ | | $r_3^0$ | $r_3^1$ | $r_3^2$ | $r_3^3$ | $r_3^4$ | $r_3^5$ | $r_3^6$ |
| $Q_3[2]$ | | | $r_3^0$ | $r_3^1$ | $r_3^2$ | $r_3^3$ | $r_3^4$ | $r_3^5$ | $Q_3[2]$ | | | $r_3^0$ | $r_3^1$ | $r_3^2$ | $r_3^3$ | $r_3^4$ | $r_3^5$ |
| $Q_3[3]$ | | | | $r_3^0$ | $r_3^1$ | $r_3^2$ | $r_3^3$ | $r_3^4$ | $Q_3[3]$ | | | | $r_3^0$ | $r_3^1$ | $r_3^2$ | $r_3^3$ | $r_3^4$ |
| $Q_3[4]$ | | | | | $r_3^0$ | $r_3^1$ | $r_3^2$ | $r_3^3$ | $Q_3[4]$ | | | | | $r_3^0$ | $r_3^1$ | $r_3^2$ | $r_3^3$ |
| $Q_3[5]$ | | | | | | $r_3^0$ | $r_3^1$ | $r_3^2$ | $Q_3[5]$ | | | | | | $r_3^0$ | $r_3^1$ | $r_3^2$ |
| $Q_1[0]$ | | | | | | | $r_1^0$ | $r_1^1$ | $Q_1[0]$ | | | | | | | $r_1^0$ | $r_1^1$ |
| $Q_1[1]$ | | | | | | | | $r_1^0$ | $Q_1[1]$ | | | | | | | | $r_1^0$ |
| $r_i^j$ ： $EM_i$ 在时间 $j$对应的判定结果 | | | | | | | | | | | | | | | | | |

**表4**：说明值选择重要性的示例 $Head$

从而实现功能正确性。我们看到，在每个周期中， $EM_1$ 都会接收对应于同一时间的操作数，正如所要求的那样。

算法1给出了一个递归过程，用于计算 $Head$对某个EM的正确值。给定一个公式，为了正确设置其监视器中所有EM的 $Head$，我们在该公式抽象语法树的根节点上调用此函数。该函数从叶节点开始向根节点移动，计算各节点的 $Head$；每当遇到二元运算符时，便平衡子树的高度。

**算法1** 在监视器中计算 EM 的 $Head$

**函数** 计算头部($EM_{cur}$，用于计算 MTL 算子 $\sigma$)
  $Head_{cur} \leftarrow l(\sigma)$
  **如果** $EM_{cur}$ 是叶节点**则**
    $Height_{cur} = Head_{cur} + 1$
  **否则，如果** $\sigma$ 是一元运算符**则**
    $(Head_{chld},\ Height_{chld}) \leftarrow$ 计算头部($Child\ of\ EM$)
    $Height_{cur} \leftarrow Head_{cur} + 1 + Height_{chld}$
  **否则，如果** $\sigma$ 是二元运算符**则**
    $(Head_{lchld},\ Height_{lchld}) \leftarrow$ 计算头部($Left\ Child\ of\ EM$)
    $(Head_{rchld},\ Height_{rchld}) \leftarrow$ 计算头部($Right\ Child\ of\ EM$)
    **如果** $Height_{lchld} = Height_{rchld}$ **则**
      $Height_{cur} \leftarrow Head_{cur} + 1 + Height_{lchld}$
    **否则如果** $Height_{lchld} < Height_{rchld}$ **则**
      $Height_{cur} \leftarrow Head_{cur} + 1 + Height_{rchld}$
      $Head_{lchld} \leftarrow Head_{lchld} + Height_{rchld} - Height_{lchld}$
      $Height_{lchld} \leftarrow Height_{rchld}$
    **否则如果** $Height_{rchld} < Height_{lchld}$ **则**
      $Height_{cur} \leftarrow Head_{cur} + 1 + Height_{lchld}$
      $Head_{rchld} \leftarrow Head_{rchld} + Height_{lchld} - Height_{rchld}$
      $Height_{rchld} \leftarrow Height_{lchld}$
    **结束如果**
  **结束 if**
  **返回** $(Head_{cur}, Height_{cur})$
**结束函数**

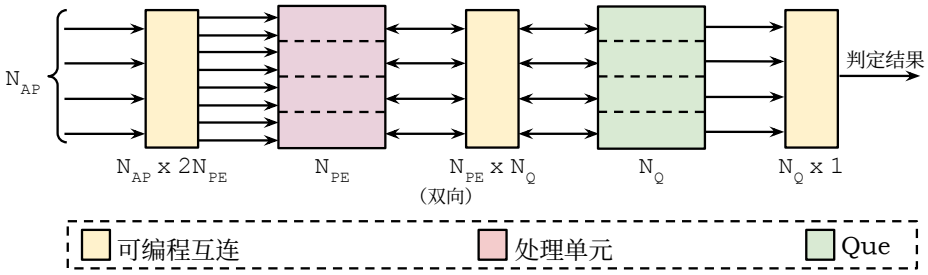



**图 2**：所提出的 MTL监视器架构（针对 $N = 4$）

# 4 可编程MTL监视器

图2 展示了可编程MTL监视器的硬件设计。整体架构大致可分为三个部分：（i）可编程互连，（ii）$N_{PE}$ 个处理单元（*PE*），以及（iii）$N_Q$ 个*Que*单元（*Qs*）。PE是抽象机的实现，而Que是第3.1节所述Que的实现。该监视器最多支持$N_{AP}$ 个AP。AP通过可编程互连连接到PE，该互连支持沿$AP2PE$（AP $\rightarrow$ PE）方向进行通信。输入AP值可以转发到任意PE输入端。类似地，PE和Que通过另一可编程互连连接，该互连支持任意PE与任意Que之间的双向通信。因此，它支持沿 $N_Q$ 方向同时建立$PE2Q$（PE $\rightarrow$ Q）方向的

最多有 $2N_{PE}$ 条连接沿 $Q2PE$ (Q $\rightarrow$ PE) 方向传输。每个 Q 内部均实现一个大小为 $Q_{SZ}$ 的缓冲区。更大的 $Q_{SZ}$ 支持 MTL 算子的更大时间区间。为了实现 MTL 算子的监视器，我们会编程一个或多个处理单元，并配置互连，将选定的处理单元连接到合适的队列。根据 $\phi$ 的抽象语法树，通过编程互连将不同的队列连接到合适的处理单元，即可实现对 MTL 公式 $\phi$ 的监视器。为 $\phi$ 生成的判定结果可在监视器中与抽象语法树根节点对应的队列中获取。该判定结果会通过另一个可编程互连（沿 $Q2OUT$ 方向传输）转发至输出引脚。这种配置允许使用任意可用的处理单元和 Q 来实现 MTL 算子。

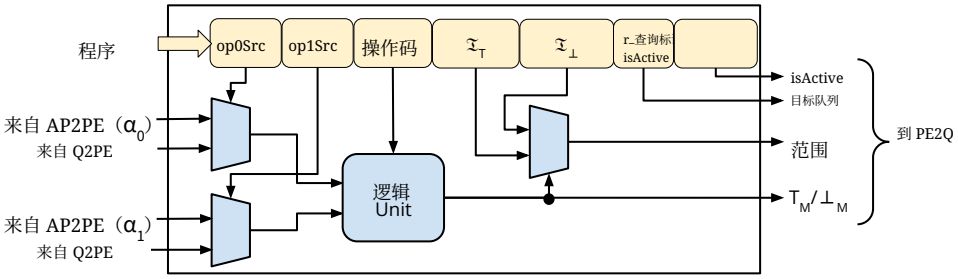
## 4.1 处理单元

**表5**：处理单元的编程

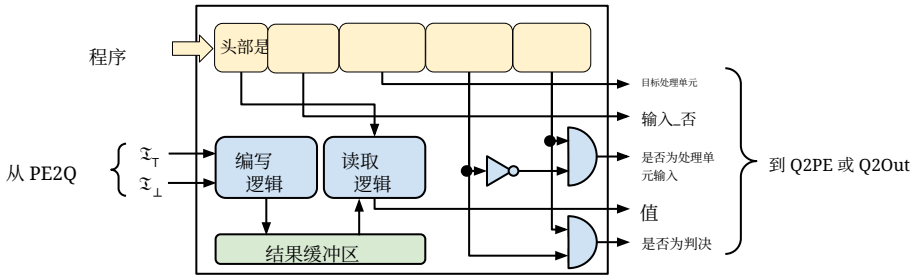
| 字段 | 描述 | 位数 |
|---|---|---|
| *isActive* | 此处理单元是否正在使用 | 1 |
| *op0Src* | 第一个操作数的来源（0：AP，1：Q） | 1 |
| *op1Src* | 第二个操作数的来源（0：AP，1：Q） | 1 |
| 操作码 | 操作码（000：导线，001：非，010：或，011：与，and 100：蕴含） | 3 |
| $r_$查询标识符 | 存放此处理单元结果的队列的标识符位置 | $\lceil$日志$_2\, N_Q\rceil$ |
| $\mathcal{I}_\top$ | 对应于 AM 定义中的 $\mathcal{I}_\top$ | $2\lceil$日志$_2\, Q_{SZ}\rceil$ |
| $\mathcal{I}_\bot$ | 对应于 AM 定义中的 $\mathcal{I}_\bot$ | $2\lceil$日志$_2\, Q_{SZ}\rceil$ |

可以使用表 5 中所述格式的指令对处理单元进行编程。字段列中的条目遵循抽象机的定义。处理单元可以接收一个或两个操作数，每个操作数可以是来自输入轨迹的 AP，也可以来自队列（即由另一个处理单元产生的结果）。处理单元通过修改标识符为 $r_qid$ 的特定队列中的内容来提供结果。最后，由于处理单元的数量在设计阶段是固定的，因此某些处理单元可能不会用于评估当前输入的 MTL 公式。将 $isActive$ 位设为0即可停用这些处理单元。若该位设为0，则会忽略其余指令字节。$isActive$ 位也会发送到 PE2Q 互连网络，以便在处理单元处于非活动状态时，互连网络能够忽略其输出（引脚上的过时值）。

图 3 展示了处理单元的架构。处理单元的数据通路始于输入引脚，这些引脚通过 AP2PE 或 Q2PE 互连网络接收两个操作数的输入。输入进入逻辑单元，该单元能够执行 全部 五种基本操作（参见表 5中的操作码）。操作结果用作多路复用器的选择线。如果操作结果为 $\top$，则已编程的 $\mathcal{I}_\top$ 会发送到输出引脚，并由 PE2Q 互连网络读取。如果结果为 $\bot$，则




(a) 处理单元




（b）队列

**图 3**： (a) 处理单元和 (b) 队列的架构

$\mathcal{I}_{\perp}$ 会发送到输出引脚。互连网络使用 $r_qid$字段作为目的字段，将值转发到相应的队列。

## 4.2 队列

图3 展示了 Q 的架构。队列的操作遵循第3.1节中给出的描述。每个周期，队列都会从 PE2Q 互连网络接收 $\mathcal{I}_{\top}$ 与 $\mathcal{I}_{\perp}$ 区间，并据此修改其缓冲区的内容。Q 可按照表6所示进行编程。从队列中删除的值构成一个或多个处理单元的源操作数，或者构成公式的判定结果（$isVerdict$被编程为 $\top$）。在前一种情况下，该值会发送到 Q2PE 互连网络；在后一种情况下，则会发送到 Q2OUT 互连网络，以便传递给外部世界。执行删除的位置可以编程设定。

## 4.3 可编程互连

四种互连均实现为单跳交叉开关。这蕴含着，在每种互连中，所有发送器模块都通过专用路径连接到所有接收器模块，并且对于每个发送器，可以通过对开关进行编程，在运行时选择其中一条路径。由于多个处理单元可以将其结果（$\mathcal{I}_{\top}/\mathcal{I}_{\perp}$ 与 $\top_{\mathcal{M}}/\perp_{\mathcal{M}}$）转发到同一个 Q，如 $\mathcal{U}$ 算子（见表 8）的情况，PE2Q 互连网络会先合并来自多个处理单元的输入，再将其转发到目标

表 6：队列的编程

| 字段 | 描述 | 位数 |
|---|---|---|
| *isActive* | Q 是否正在使用 | 1 |
| 是否为判决 | 队列内容是否为监视器总体判定 | 1 |
| $readerPE$ | 正在读取队列内容的处理单元的标识符 | $\lceil$日志$_2\, N_{PE}\rceil$ |
| $inp_no$ | 队列内容是第一个还是第二个操作数 *readerPE* | 1 |
| $Head$ | 删除位置 | $\lceil$日志$_2\, Q_{SZ}\rceil$ |

表7：不同监控组件的编程

| 处理元素编程 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PE | *isActive* | *op0Src* | *op1Src* | 操作码 *r* | $_qid$ | $\mathcal{I}_\top$ | $\mathcal{I}_\bot$ | 备注 |
| $PE_0$ | true | AP | X | *not* | 0 | [0, 0] | [0, 0] | $EM_4$ 见图 1(b) |
| $PE_1$ | true | AP | X | *wire* | 1 | [1, 4] | [4, 4] | $EM_3$ 见图 1(b) |
| $PE_2$ | true | Q2PE | X | *wire* | 2 | [0, 1] | [1, 1] | $EM_2$ 见图 1(b) |
| $PE_3$ | true | Q2PE | Q2PE | *or* | 3 | [0, 0] | [0, 0] | $EM_1$ 见图 1(b) |
| $PE_{4-7}$ | 假 | X | X | X | X | X | X | 未使用 |

| Qs 的编程 | | | | | | |
|---|---|---|---|---|---|---|
| Q | *isActive* | 是否为判决 | *readerPE* 输入 | $_no$ | $Head$ | 备注 |
| $Q_0$ | true | 假 | 2 | 0 | 1 | $Q_4$ 来自图 1(b) |
| $Q_1$ | true | 假 | 3 | 1 | 5 | $Q_3$ 来自图 1（b） |
| $Q_2$ | true | 假 | 3 | 0 | 3 | $Q_2$ 来自图 1（b） |
| $Q_3$ | true | true | X | X | 1 | $Q_1$ 来自图 1（b） |
| $Q_{4-7}$ | 假 | X | X | X | X | 未使用 |

Q。这种合并操作涉及将来自不同处理单元的 $\mathcal{I}_\top$（等价地，$\mathcal{I}_\bot$）区间合并为一个连续的 $\mathcal{I}_\top$（$\mathcal{I}_\bot$）区间。这个合并后的区间是目标 Q 的输入。可以保证，合并只会扩大范围，并且要写入这些范围的值不会相互冲突（见命题 1），因此合并操作很简单。

## 4.4 监视器编程

我们的编译器采用一种简单的策略。首先构造输入 MTL公式的抽象语法树。然后，按照逆拓扑序处理节点，为每个节点分配一个或多个处理单元，并为每个节点唯一的出边分配一个队列。我们按照第3.3节所述进行队列均衡。最后，如前文所述，生成用于配置处理单元、队列和互连的相应位。

我们通过第3.3节中的示例来阐释编译器的工作原理。表7 列出了不同监视器组件的编程配置。假设监视器具有 $N_{PE} = 8$、 $N_Q = 8$和 $N_{AP} = 4$ （支持 4 个 AP，从 $\sigma_0$ 到 $\sigma_3$）。

对处理单元进行编程所需的位数为 $N_{PE} \times (6 + \lceil \log_2 N_Q \rceil + 4 \times \lceil \log_2 Q_{SZ} \rceil)$。对队列进行编程所需的位数为 $N_Q \times (3 + \lceil \log_2 N_{PE} \rceil + \lceil \log_2 Q_{SZ} \rceil)$。对 AP2PE互连进行编程所需的位数为$N_{PE} \times 2 \times \lceil \log_2 N_{AP} \rceil$。

# 5 工具描述与评估

由于 MTL监视器的设计由 $N_{PE}$、 $N_Q$、 $N_{AP}$ 与 $Q_{SZ}$ 参数化，我们开发了一种工具，该工具将这些参数作为输入，并以 Clash硬件描述语言 [28] 生成相应的可编程监视器设计。随后，使用 Clash编译器从生成的可编程监视器 Clash 描述中生成可综合的 Verilog代码。当监视器需要针对给定公式进行配置时，我们的工具还会生成处理单元和队列的编程位。每个时钟周期将 8 位编程位依次输入 MTL监视器。这样做是为了限制设计中的输入/输出引脚数量，但不会影响电路的功能。该工具还提供判定延迟，即判定结果可用的时间与该判定结果所对应时间之间的差值。该工具可在线获取，且采用开源方式发布 [13]。

我们的工具使用 Python 编写。通过针对若干 MTL性质 对设计进行仿真，验证了所生成的硬件描述语言的正确性。输入属性会动态更改，如图4 所示（模拟运行时重新编程）。测试平台也使用 Clash 编写。

我们使用 Synopsys设计编译器 (DC) 和教育设计套件 (EDK) 中提供的 $32nm$ 标准单元库（SAED32nm_EDK）对设计进行综合。

## 5.1 仿真

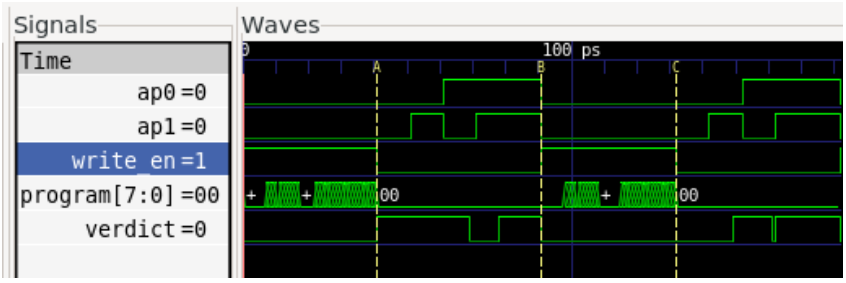



**图 4**：展示 MTL监视器 在运行时重新编程的仿真

图4 展示了这样的场景所对应的波形：监视器在运行时动态重新编程，以便依次监视两个不同的属性

。值变更转储（VCD）文件由设计 $N_{PE} = N_Q = 4$、 $N_{AP} = 8$ 和 $Q_{SZ} = 16$ 的 Clash 仿真生成。我们使用 GTKWave 查看 VCD 文件。设计的时钟、复位和使能信号在 Clash 中是隐式的，因此不会显示在波形中。仿真采用 Clash 的默认设置，即时钟周期为 $1ps$。由于我们只关注验证每个时钟周期内的行为，因此仿真时选择的时钟周期并不重要。设计本身经过流水线化处理，在互连的输入端添加了触发器，以提高综合后的时钟频率。

在此场景中监视的第一个性质是 $AP_0 \rightarrow \mathcal{X} AP_1$，随后是$AP_0 \vee \diamond_{[1,3]} AP_1$。重新编程监视器所需的周期数取决于 $N_{PE}$、 $N_Q$ 与 $N_{AP}$等参数的选择。对于我们的仿真设计，由于每个时钟周期只有 8 个编程位被发送到监视器中（参见信号 program[7:0]），因此重新编程监视器需要 41 个周期。我们观察到，从仿真开始（周期 0）到标记 A（周期 40 处的垂直虚线）的这些周期将监视器编程为验证第一个性质，而周期 91 到 131 则将监视器重新编程为验证第二个性质（位于标记 B 与 C 之间）。在这些时段内，信号 write_en 为高电平。MTL监视器在周期 41 到周期 90 之间验证第一个性质，该时段位于标记 A 与 B 之间。根据第一个性质的语义，当 $AP_0$ 为真且下一周期的 $AP_1$ 为假时，判定结果为假，否则判定结果为真。在图4中，我们看到 $AP_0$ 在周期 41 到 61 之间为假，并在周期 62 变为真。周期 63 中， $AP_1$ 为假。因此，周期 62 对应公式的判定结果为假。该判定结果在周期 70 可用。同样，周期 63 的判定结果（同样为假）在周期 71 可用，依此类推。因此，此处的延迟为 8 个周期（周期 $i$ 的判定结果在周期 $(i + 8)$ 可用），而吞吐量为每周期 1 个判定结果。该延迟是所监视公式时域的函数。在监视第二个性质时，我们可以对标记 C（位于 $132ps$）与仿真结束（位于 $182ps$）之间的时段作出类似观察。在这种情况下，判定结果在延迟 11 个周期后出现。

这表明，生成的 MTL 监视器只需更改其输入引脚上的值即可重新编程，无需重新综合其内部电路（类似于通用处理器）。

## 5.2 综合结果

我们通过调整 $N_{PE}$(2, 4, 8, 16)、 $N_Q$(2, 4, 8, 16) 和 $Q_{SZ}$(4, 16, 64, 256)，对一个硬件监视器（支持 $N_{AP}$ =16）进行了设计空间探索。在所有实验中， $N_Q$ 均设为$N_{PE}$。图5展示了每个综合后的 MTL 监视器所占用的总面积，以及各组成部分的面积细分。图6展示了相同面积细分的百分比。标记为处理单元、队列和互连的柱形表示这些组件中的标准单元所占的面积，而对应路由的柱形表示连接这些标准单元的导线所占的面积。为简洁起见，我们将 AP2PE、PE2Q、Q2PE 和 Q2OUT 等组件对应的标准单元归入互连。

总面积会随着 $N_{PE}$ 的增加而超线性增长（保持 $Q_{SZ}$ 不变）。这是因为总面积的主要构成部分——路由与

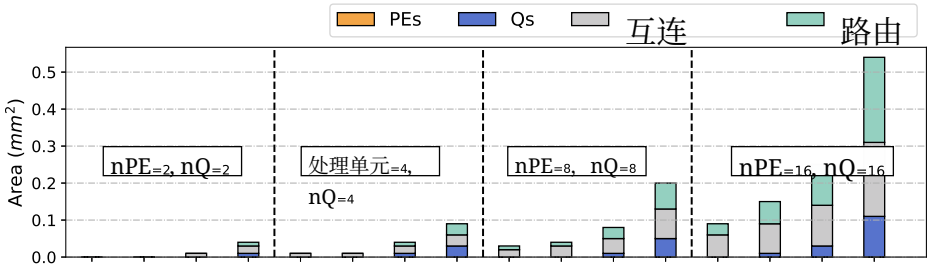



**图 5**：处理单元、队列与互连所占面积（针对 $N_{AP} = 16$）

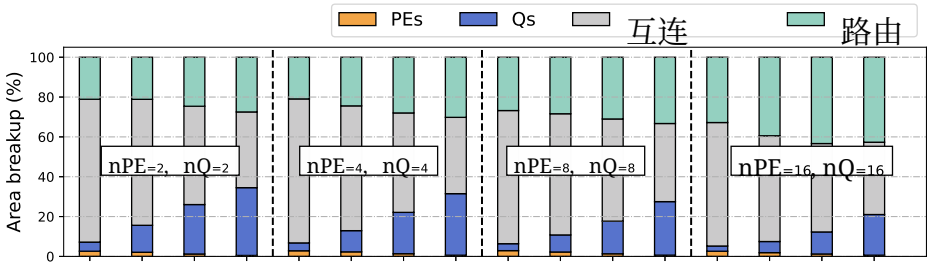



**图 6**：MTL监视器的面积占比细分（针对 $N_{AP} = 16$）

互连——呈超线性增长。原因在于，随着 $N_{PE}$ 增加，（i）互连必须支持更多输入和输出端口；（ii）整体设计规模更大，导致导线更长。采用更好的互连架构，而非单跳交叉开关，可以在一定程度上缓解面积的急剧增长。更好的互连设计探索留待未来工作。一般而言，这类设计空间探索是硬件设计中的标准做法，有助于设计人员微调电路，使其满足典型输入属性的要求。这种探索是一次性的（设计）工作，无需针对每个输入属性重复进行。随着 $Q_{SZ}$ 增加，我们观察到，尽管队列占用的面积呈超线性增长，但总面积呈次线性增长，因为主要的路由与互连组件也呈现这一趋势。

我们还注意到，与情况 $N_{PE} = N_Q = 16$,$N_{AP} = 16$ 和 $Q_{SZ} = 256$ 对应的最大设计占用 $0.55mm^2$ （$547781\mu m^2$），仅占典型 $400mm^2$ 芯片面积的0.13%（通常是通用处理器的芯片）。

我们综合得到的监视器的关键路径延迟为 $0.8ns$（对应的时钟频率为 $1.25GHz$）。这意味着监视器具有很高的吞吐量——每秒可观察 $1.25 \times 10^9$ 次观测，每个周期产生一个判定结果。这显著优于使用 FPGA 时通常能够达到的时钟频率（在 $MHz$ 中）。如有需要，还可以通过添加

更多触发器来进一步提高时钟频率。这会增加流水线深度，从而增加针对给定输入事件获得判定结果所需的时钟周期数（即延迟），并增大电路占用的总面积。不过，增加流水线深度时，吞吐量仍保持为每个周期一个判定结果。由于设计（或电路）不会随输入属性的变化而改变，因此只需在设计阶段探索一次面积与频率之间的这种权衡。

# 6 结论

在本研究中，我们提出了一种硬件MTL监视器，该监视器适合使用标准单元实现，同时仍具备灵活性，能够处理待监视性质的变化。这是通过使用一组可编程处理单元、队列和互连实现的。每个处理单元实现五种基本操作和一组队列更新规则，而队列内部的结果缓冲区负责保持时序行为，并保存所生成的中间结果和判定结果。互连经过编程，用于在处理单元和队列之间传递结果，从而使它们能够组合起来监视复杂的MTL性质。我们开发了一个监视器生成器，该生成器以 $N_{PE}$ 和 $N_Q$ 等参数作为输入，并生成所提出监视器的可综合Verilog代码。我们还开发了一个配套的编译器，它除了接收上述参数外，还接收待监视的性质，并生成用于对监视器进行适当编程的编程位。我们的仿真表明，生成的设计可以在运行时进行编程。我们还观察到，即使是一个支持多达16个AP的较大型MTL监视器，其面积也仅为 $0.55mm^2$ ，同时运行于 $1.25GHz$。

# 参考文献

[1]Reinbacher, T., Rozier, K.Y., Schumann, J.：用于实时系统健康管理的基于时序逻辑的运行时观察器对。载于：系统构造与分析工具和算法：第20届国际会议，TACAS 2014，作为欧洲软件理论与实践联合会议 ETAPS 2014 的一部分举行，法国格勒诺布尔，2014年4月5日至13日。论文集第20卷，第357–372页（2014）。Springer

[2] Finkbeiner, B., Kuhtz, L.：用于有界和无界未来线性时序逻辑的监视器电路。载于：国际运行时验证研讨会，第60–75页（2009）。Springer

[3] Jakšić, S., Bartocci, E., Grosu, R., Kloibhofer, R., Nguyen, T., Ničković, D.：从信号时序逻辑到 FPGA 监视器。载于：2015年 ACM/IEEE 国际形式化方法与协同设计模型会议（MEMOCODE），第218–227页（2015）。IEEE

[4] Lu, H., Forin, A.：面向零开销在线软件验证的自动处理器定制。IEEE超大规模集成电路学报（VLSI）

系统**16**(10), 1346–1357 (2008)

[5] Moosbrugger, P., Rozier, K.Y., Schumann, J.：R2u2：面向无人飞行系统的安全威胁监控与诊断。系统设计中的形式化方法**51**(1), 31–61 (2017)[6] Pellizzoni, R., Meredith, P., Caccamo, M., Rosu, G.：面向可靠的基于商用现货的实时嵌入式系统的硬件运行时监控。载于：2008年实时系统研讨会，第481–491页（2008）。IEEE[7] Reinbacher, T., Brauer, J., Horauer, M., Steininger, A., Kowalewski, S.：微控制器二进制代码的过去时线性时序逻辑运行时验证。载于：工业关键系统形式化方法：第16届国际研讨会，FMICS 2011，意大利特伦托，2011年8月29—30日。论文集16，第37–51页（2011）。Springer[8] Jindal, N., Chandran, S., Panda, P. R., Prasad, S., Mitra, A., Singhal, K., Gupta, S., Tuli, S.：Dhoom：复用面向调试的硬件进行在线监控。载于：第56届年度设计自动化会议论文集，2019年，第1–6页（2019）[9] Kempa, B., Zhang, P., Jones, P.H., Zambreno, J., Rozier, K.Y.：在Robonaut2中嵌入在线运行时验证以进行故障消歧。载于：国际定时系统形式化建模与分析会议，第196–214页（2020）。Springer[10] Rahmatian, M., Kooti, H., Harris, I.G., Bozorgzadeh, E.：利用部分运行时重配置实现自适应入侵检测。载于：2012年IEEE第30届国际计算机设计会议（ICCD），第147–152页（2012）。IEEE [11] Benny, A., Chandran, S., Kalayappan, R., Phawade, R., Kurur, P.P.：faRM-LTL：用于灵活且加速监控线性时序逻辑性质的领域专用架构。载于：国际运行时验证会议，第109–127页（2024）。Springer[12]Alur, R., Henzinger, T.A.：实时逻辑：复杂度与表达能力。《信息与计算》**104**(1), 35–77(1993)[13] Hebballi, C., Poptani, A., Benny, A., Kalayappan, R., Chandran,S., Phawade, R.：可编程度量时序逻辑监控器。https://doi.org/10.5281/zenodo.19084848 。https://doi.org/10.5281/zenodo.19084848[14] Reinbacher, T., Függer, M., Brauer, J.：实时嵌入式系统的运行时验证。《系统设计中的形式化方法》**44**，203–239（2014）[15]Thati, P., Roşu, G.：度量时序逻辑规范的监控算法。《理论计算机科学电子札记》**113**，145–162（2005）

[16] Bartocci, E., Deshmukh, J., Donzé, A., Fainekos, G., Maler, O., Ničković, D., Sankaranarayanan, S.：信息物理系统的规约监控：理论、工具与应用综述。《运行时验证讲义：入门与高级主题》，135–175（2018）[17] Maler, O., Nickovic, D., Pnueli, A.：离散、定时和连续行为时序性质的检查。《计算机科学支柱：献给 Boris (Boaz) Trakhtenbrot 85 岁诞辰的论文集》，475–505（2008）[18] Ho, H.-M., Ouaknine, J., Worrell, J.：度量时序逻辑的在线监控。载于：《国际运行时验证会议》，第178–192页（2014）。Springer[19] Geilen, M.：一种改进的实时时序逻辑即时表格构造方法。载于：《国际计算机辅助验证会议》，第394–406页（2003）。Springer[20] Selyunin, K., Nguyen, T., Bartocci, E., Nickovic, D., Grosu, R.：使用 IBM 脉冲神经元模型监控 MTL规约。载于：2016年欧洲设计、自动化与测试会议暨展览（DATE），第924–929页（2016）。IEEE[21] Basin, D., Klaedtke, F., Zălinescu, E.：实时性质监控算法。载于：《国际运行时验证会议》，第260–275页（2011）。Springer[22] Maler, O., Nickovic, D., Pnueli, A.：基于有界响应性质综合控制器。载于：《国际计算机辅助验证会议》，第95–107页（2007）。Springer[23] Divakaran, S., D'Souza, D., 等：度量时序逻辑中的冲突容忍实时规范。载于：2010年第17届国际时序表示与推理研讨会，第35–42页（2010）。IEEE[24] Wang, K., Angstadt, K., Bo, C., Brunelle, N., Sadredini, E., Tracy, T., Wadden, J., Stan, M., Skadron, K.：Micron 自动机处理器概述。载于：第十一届IEEE/ACM/IFIP硬件/软件协同设计与系统综合国际会议论文集，第1–3页（2016）[25] Chattopadhyay,A.,Mamouras,K.：一种经过验证的、具有定量语义的度量时序逻辑在线监视器。载于：《运行时验证：第20届国际会议》，RV 2020，美国加利福尼亚州洛杉矶，2020年10月6日至9日，论文集20，第383–403页（2020）。Springer[26] Donzé, A.：论信号时序逻辑。载于：《运行时验证：第四届国际会议》，RV 2013，法国雷恩，2013年9月24日至27日。论文集4，第

382–383（2013）。Springer

[27]Donzé, A., Ferrere, T., Maler, O.：面向 STL 的高效鲁棒监控。载于：《计算机辅助验证：第25届国际会议》，CAV2013，俄罗斯圣彼得堡，2013年7月13日至19日。论文集25，第264–279页（2013）。Springer

[28] Baaij, C., Kooijman, M., Kuper,J., Boeijink, A., Gerards, M.：Cλash：使用 Haskell 对同步硬件进行结构描述。载于：2010年第13届Euromicro数字系统设计会议：架构、方法与工具，第714–721页（2010）。IEEE

# 附录：求值机的正确性证明

**表 8**：对应不同 MTL 运算符的求值机操作

| **操作数值 在时间 $i$** | | **队列修改** | | **推理** |
|---|---|---|---|---|
| $\alpha_0$ | $\alpha_1$ | **值** | **位置** | |
| **非算子：** $\neg\alpha_0$ | | | | |
| $\bot$ | - | $\top$ | 0 | $r^i = \neg\alpha_0^i$ |
| $\top$ | - | $\bot$ | 0 | |
| **或算子：** $\alpha_0 \vee \alpha_1$ | | | | |
| $\bot$ | $\bot$ | $\bot$ | 0 | |
| $\bot$ | $\top$ | $\top$ | 0 | $r^i = \alpha_0^i \vee \alpha_1^i$ |
| $\top$ | $\bot$ | $\top$ | 0 | |
| $\top$ | $\top$ | $\top$ | 0 | |
| **与算子：** $\alpha_0 \wedge \alpha_1$ | | | | |
| $\bot$ | $\bot$ | $\bot$ | 0 | |
| $\bot$ | $\top$ | $\bot$ | 0 | $r^i = \alpha_0^i \wedge \alpha_1^i$ |
| $\top$ | $\bot$ | $\bot$ | 0 | |
| $\top$ | $\top$ | $\top$ | 0 | |
| **蕴含算子：** $\alpha_0 \to \alpha_1$ | | | | |
| $\bot$ | $\bot$ | $\top$ | 0 | |
| $\bot$ | $\top$ | $\top$ | 0 | $r^i = \alpha_0^i \to \alpha_1^i$ |
| $\top$ | $\bot$ | $\bot$ | 0 | |
| $\top$ | $\top$ | $\top$ | 0 | |
| **下一步算子：** $\mathcal{X}\ \alpha_0$ | | | | |
| $\bot$ | - | $\bot$ | 1 | $r^{i-1} = \alpha_0^i$ |
| $\top$ | - | $\top$ | 1 | |
| **框算子：** $\Box_{[t_1,t_2]}\ \alpha_0$ | | | | |
| $\bot$ | - | $\bot$ | $[t_1, t_2]$ | $\alpha_0^i = \bot \to r^t = \bot, \forall t \in [i-t_2, i-t_1]$ |
| $\top$ | - | $\top_{\mathcal{M}}$ | $t_2$ | $r^{i-t_2} \neq \bot \wedge \alpha_0^i = \top \to r^{i-t_2} = \top$ |
| **菱形算子：** $^{\Diamond_{[t_1,t_2]}}\ \alpha_0$ | | | | |
| $\bot$ | - | $\bot_{\mathcal{M}}$ | $t_2$ | $r^{i-t_2} \neq \top \wedge \alpha_0^i = \bot \to r^{i-t_2} = \bot$ |
| $\top$ | - | $\top$ | $[t_1, t_2]$ | $\alpha_0^i = \top \to r^t = \top, \forall t \in [i-t_2, i-t_1]$ |

| 直到算子： $\alpha_0$ | | | | $\mathcal{U}_{[t_1,t_2]}\ \alpha_1$ |
|---|---|---|---|---|
| $\bot$ | $\bot$ | $\bot_{\mathcal{M}}$ | $t_2$ | $[M_2:\ wire\ \alpha_1]\ \alpha_1$ 从未在 $[t_1,t_2]$ 窗口中为真时间 $i-t_2$ |
| | | $\bot_{\mathcal{M}}$ | $[t_1,t_2-1]$ | $[M_3:\ or\ \alpha_0,\ \alpha_1]\ \forall t\in[i-t_2+1,i-t_1],\ r^t\neq\top\wedge\alpha_0^i=\bot\wedge\alpha_1^i=\bot\rightarrow r^t=\bot$ |
| | | $\bot$ | $[0,t_1-1]$ | $[M_1:$ 已发生 $wire\ \alpha_0]\ \alpha_0=\bot$ 在时间为 $[0,t_1-1]$ 的窗口内 $[i,i-t_1+1]$ |
| $\bot$ | $\top$ | $\bot$ | $[0,t_1-1]$ | $[M_1:$ 已发生 $wire\ \alpha_0]\ \alpha_0=\bot$ 在时间为 $[0,t_1-1]$ 的窗口内 $[i,i-t_1+1]$ |
| | | $\top_{\mathcal{M}}$ | $[t_1,t_2]$ | $[M_2:\ wire\ \alpha_1]\ \forall t\in[i-t_2,i-t_1],\ r^t\neq\bot\wedge\alpha_1^i=\top\rightarrow r^t=\top$ |
| $\top$ | $\bot$ | $\bot_{\mathcal{M}}$ | $t_2$ | $[M_2:\ wire\ \alpha_1]\ \alpha_1$ 从未在 $[t_1,t_2]$ 窗口中为真时间 $i-t_2$ |
| $\top$ | $\top$ | $\top_{\mathcal{M}}$ | $[t_1,t_2]$ | $[M_2:\ wire\ \alpha_1]\ \forall t\in[i-t_2,i-t_1],\ r^t\neq\bot\wedge\alpha_1^i=\top\rightarrow r^t=\top$ |
| **直到算子：** $\alpha_0$ | | | | $\mathcal{U}_{[0,t_2]}\ \alpha_1$ |
| $\bot$ | $\bot$ | $\bot_{\mathcal{M}}$ | $t_2$ | $[M_2:\ wire\ \alpha_1]\ \alpha_1$ 从未在 $[0,t_2]$ 窗口中为真时间 $i-t_2$ |
| | | $\bot_{\mathcal{M}}$ | $[0,t_2-1]$ | $[M_1:\ or\ \alpha_0,\ \alpha_1]\ \forall t\in[i-t_2+1,\ i],\ r^t\neq\top\wedge\alpha_0^i=\bot\wedge\alpha_1^i=\bot\rightarrow r^t=\bot$ |
| $\bot$ | $\top$ | $\top_{\mathcal{M}}$ | $[0,t_2]$ | $[M_2:\ wire\ \alpha_1]\ \forall t\in[i-t_2,i],\ r^t\neq\bot\wedge\alpha_1^i=\top\rightarrow r^t=\top$ |
| $\top$ | $\bot$ | $\bot_{\mathcal{M}}$ | $t_2$ | $[M_2:\ wire\ \alpha_1]\ \alpha_1$ 从未处于在 $[0,t_2]$ 窗口中为真，持续时间 $i-t_2$ |
| $\top$ | $\top$ | $\top_{\mathcal{M}}$ | $[0,t_2]$ | $[M_2:\ wire\ \alpha_1]\ \forall t\in[i-t_2,i],\ r^t\neq\bot\wedge\alpha_1^i=\top\rightarrow r^t=\top$ |

$\top_{\mathcal{M}}$：如果单元格的当前值为 $\mathcal{M}$，则写入值 $\top$
$\bot_{\mathcal{M}}$：如果单元格的当前值为 $\mathcal{M}$，则写入值 $\bot$
$\alpha_k^t$：操作数 $\alpha_k$ 在时间 $t$ 的值
$r^t$：截至当前时间计算得出的、对应时间 $t$ 的结果

表 8 描述了监视器针对每个 MTL 算子所遵循的运行逻辑。对于 AM 在某个周期可能接收的所有输入操作数值组合 $i$ ($\alpha^i_{\{0,1\}}$)，第 4 列列出了监视器中结果队列的索引；这些索引对应的值将被更新为第 3 列所示的值，以记录 $\alpha^i_{\{0,1\}}$ 对（部分）判定结果产生的影响。第 5 列给出的推理有助于直观地证明，监视器的行为准确地体现了相应算子的运算逻辑。

现在，我们分别为每个算子给出形式化证明：

**(定理1 关于布尔运算符的证明)**

：根据定义， $l()$ 将所有布尔运算符映射为 1。我们证明，对所有时间实例 $i \geq 1$:

- 若 $x^{i-1} \vDash \phi$ ，则 $Q^i[1] = \top$。
- 若 $x^{i-1} \nvDash \phi$ ，则 $Q^i[1] = \bot$。

$\phi = \neg\alpha$：假设 $x^{i-1} \vDash \neg\alpha$。只有在时间步 $i-1$，公式 $\alpha$ 求值为 $\bot$时，才有可能出现这种情况。在此情况下，抽象机将 $Q^{i-1}[0] = \top$设为判定结果 $r^{i-1}$，对应于 $\neg\alpha$。

假设 $x^{i-1} \nvDash \neg\alpha$。只有在时间步 $i-1$，公式 $\alpha$ 求值为 $\top$时，才有可能出现这种情况。在此情况下，抽象机将 $Q^{i-1}[0] = \bot$设为判定结果 $r^{i-1}$，对应于 $\neg\alpha$。

在下一个时间步，当插入 $\mathcal{M}$ 时，判定结果 $r^{i-1}$ 会按要求移至 $Q^i[1]$，因为抽象机不会修改单元格 1。

□

$\phi = \alpha \wedge \beta$：假设 $x^{i-1} \vDash \alpha \wedge \beta$。这仅在时间实例 $i-1$ 时才可能，即公式 $\alpha$ 与 $\beta$ 均求值为 $\top$。在这种情况下，抽象机将 $Q^{i-1}[0] = \top$ 设为 $r^{i-1}$，即 $\alpha \wedge \beta$ 的判定结果。

假设 $x^{i-1} \nvDash \alpha \wedge \beta$。这仅在时间实例 $i-1$ 时才可能，即 $\alpha$ 和 $\beta$ 中至少一个公式求值为 $\bot$。在这种情况下，抽象机将 $Q^{i-1}[0] = \bot$ 设为 $r^{i-1}$，即 $\alpha \wedge \beta$ 的判定结果。

在下一个时间步，当插入 $\mathcal{M}$ 时，由于抽象机不修改单元格 1，判定结果 $r^i$ 会按要求移至 $Q^i[1]$。□

**（定理 1 关于 $\mathcal{X}\alpha$ 的证明）**

：

证明 根据定义， $l()$ 将 $\mathcal{X}$ 算子映射为 2。
在此我们证明，对于所有时间实例 $i$，满足 h $i \geq 2$,

$$if\ x^{i-2} \vDash \mathcal{X}\alpha,\ then\ Q^i[2] = \top.$$

$$if\ x^{i-2} \nvDash \mathcal{X}\alpha,\ then\ Q^i[2] = \bot.$$

根据抽象机的定义，在时间实例 $i-2$，队列 $Q^{i-2}[0]$ 包含 $\mathcal{M}$。这是对应于时间 $i-2$的判定结果，并且此时抽象机尚未对此作出判定。

假设 $x^{i-2} \vDash \mathcal{X}\alpha$。这意味着 $x^{i-1} \vDash \alpha$。因此抽象机设置 $Q^{i-1}[1] = \top$，这是针对实例 $i-2$的 $\mathcal{X}\alpha$ 的判定结果 $r^{i-2}$。
假设 $x^{i-2} \nvDash \mathcal{X}\alpha$。这意味着 $x^{i-1} \nvDash \alpha$。因此抽象机设置 $Q^{i-1}[1] = \bot$，这是针对实例 $i-2$的 $\mathcal{X}\alpha$ 的判定结果 $r^{i-2}$。
在下一个时间步，当插入 $\mathcal{M}$ 时，判定结果 $r^{i-2}$ 按要求移至 $Q^i[2]$，因为抽象机不会修改单元格 2。

□

**（定理1 的证明 $\Box_{[t_1,t_2]}\alpha$）**

：

证明 根据定义， $l()$ 将 $\Box_{[t_1,t_2]}$ 算子映射到 $t_2+1$。要证明对于
所有时间实例 $i \geq t_2+1$:

• 若 $x^{i-t_2-1} \vDash \Box_{[t_1,t_2]}\alpha$，则 $Q^i[t_2+1] = \top$。 • 若 $x^{i-t_2-1} \nvDash \Box_{[t_1,t_2]}\alpha$，则 $Q^i[t_2+1] = \bot$。(假设 $x^{i-t_2-1} \vDash \Box_{[t_1,t_2]}\alpha$)：于是，对于所有属于 $[i-1-t_2+t_1, i-1]$, $x^l \vDash \alpha$ 的 $l$，都有。我们将区间 $[i-1-t_2, i-1]$ 分成四部分：

• 在时间 $i-t_2-1$，根据抽象机的定义，值 $\mathcal{M}$ 被插入索引 0 处的队列中，并且该值 $Q^{i-1-t_2}[0]$ 在此时间单位内不会改变。该值对应判定结果 $r^{i-1-t_2}$。

• 对于所有时间 $l' \in [i-1-t_2+1, i-1-t_2+t_1-1]$，根据抽象机的定义，单元 $[0, t_1-1]$ 中的值不会改变。设定于 $Q^{i-1-t_2}[0]$ 的值 $\mathcal{M}$ 只会未经改变地依次经过 $Q^{l'}[l'-(i-1-t_2)]$。

• 对于所有时间 $l'' \in [i-1-t_2+t_1, i-1-t_2+1]$，根据抽象机的定义，单元 $[0, t_2-1]$ 中的值不会改变。设定于 $Q^{i-1-t_2}[0]$ 的值 $\mathcal{M}$ 只会未经改变地依次经过 $Q^{l''}[l''-(i-1-t_2)]$。

• 在时间 $i-1$，由于 $x^{i-1} \vDash \alpha$，根据抽象机的定义，单元格$Q^{i-1}[t_2]$中的值被设为 $\top$。

(假设 $x^{i-t_2-1} \nvDash \Box_{[t_1,t_2]}\alpha$)：那么，对于某个 $l$，在 $[i-1-t_2+t_1, i-1]$, $x^l \nvDash \alpha$。我们将区间 $[i-1-t_2, i-1]$ 分成五部分：

• 在时间 $i-t_2-1$，根据抽象机的定义，值 $\mathcal{M}$ 被插入队列索引 0处，并且在此时间单位内，值 $Q^{i-1-t_2}[0]$ 不会改变。此值对应于判定结果 $r^{i-1-t_2}$。

• 对于所有时间 $l' \in [i-1-t_2+1, i-1-t_2+t_1-1]$，根据抽象机的定义，单元格 $[0, t_1-1]$ 中的值不会改变。设于 $Q^{i-1-t_2}[0]$ 中的值 $\mathcal{M}$ 只是未经改变地依次经过 $Q^{l'}[l'-(i-1-t_2)]$。

• 对于所有时间 $l'' \in [i-1-t_2+t_1, l-1]$，由于 $x^{l''} \vDash \alpha$，根据抽象机的定义，单元 $[0, t_2-1]$ 中的值不会改变。已在 $Q^{i-1-t_2}[0]$ 设定的值 $\mathcal{M}$ 只是未经更改地移过 $Q^{l''}[l''-(i-1-t_2)]$。

• 在时间 $l$，由于 $x^l \nvDash \alpha$，根据抽象机的定义，单元 $Q^l[t_1,t_2]$中的值被设为 $\bot$。此范围包括单元 $Q^l[l-(i-1-t_2)],$，其旧值为 $\mathcal{M}$。
• 对于所有时间 $l''' \in [l+1, i-1]$，根据抽象机的定义，在 $Q^l[l-(i-1-t_2)]$ 设定的值 $\bot$ 只是未经更改地移过 $Q^{l'''}[l'''-(i-1-t_2)]$。因此，在时间 $i-1$，单元 $Q^{i-1}[t_2]$ 中的值为 $\bot$。

在下一个时间步中，当插入 $\mathcal{M}$ 时，判定结果 $r^{i-1-t_2}$ 按要求移动到 $Q^i[t_2+1]$，因为抽象机不会修改单元格 $t_2+1$。□

定理1 关于 $\phi = \Diamond_{[t_1,t_2]}\alpha$ 可以用类似的方法证明。

## （定理1 关于 $\phi = \alpha\, \mathcal{U}_{[t_1,t_2]}\beta$）

:

证明根据定义， l() 将 $\mathcal{U}_{[t_1,t_2]}$ 算子映射到 $t_2+1$。要证明，对于所有时间实例 $i \geq t_2+1$:

• 如果 $x^{i-t_2-1} \vDash \alpha\, \mathcal{U}_{[t_1,t_2]}\beta$，那么 $Q^i[t_2+1] = \top$。
• 如果 $x^{i-t_2-1} \nvDash \alpha\, \mathcal{U}_{[t_1,t_2]}\beta$，那么 $Q^i[t_2+1] = \bot$。

(假设 $x^{i-t_2-1} \vDash \alpha\ \mathcal{U}_{[t_1,t_2]}\beta$):

令 $l$为 $[i-1-t_2+t_1, i-1]$中满足 $x^l \vDash \beta$且对所有满足 $i-1-t_2 \le k < l-1$, $x^l \vDash \alpha$的 $k$而言最早的时间实例。我们将区间 $[i-1-t_2, i-1]$分为五个部分:

- 在时间 $i-1-t_2$，根据抽象机的定义，将值 $\mathcal{M}$ 插入队列索引 0 处，并且该值 $Q^{i-1-t_2}[0]$ 在此时间单位内不会改变。此值对应于判定结果 $r^{i-1-t_2}$。

- 对于所有时间 $l' \in [i-1-t_2+1, i-1-t_2+t_1-1]$，我们知道 $x^{l'} \vDash \alpha$；并且根据抽象机的定义，单元格 $[0, t_1-1]$ 中的值不会改变。已在 $Q^{i-1-t_2}[0]$ 中设置的值 $\mathcal{M}$ 仅会沿着 $Q^{l'}[l' - (i-1-t_2)]$ 移动，不发生改变。- 对于所有时间 $l'' \in [i-1-t_2+t_1, l-1]$，我们知道 $x^{l''} \vDash \alpha$ 且 $x^{l''} \nvDash \beta$。根据抽象机的定义，单元格 $[0, t_2-1]$ 中的值不会改变。已在 $Q^{i-1-t_2}[0]$ 中设置的值 $\mathcal{M}$ 仅会沿着 $Q^{l''}[l'' - (i-1-t_2)]$ 移动，不发生改变。

- 在时间 $l$，我们知道 $x^l \vDash \beta$。根据抽象机的定义，若当前值为 $\mathcal{M}$，单元 $Q^l[t_1, t_2]$ 中的值将设为 $\top$ 。此范围包含单元$Q^l[l-(i-1-t_2)],$，其旧值为 $\mathcal{M}$。

- 对于所有时间 $l''' \in [l+1, i-1]$，根据抽象机的定义，在 $Q^l[l-(i-1-t_2)]$ 中设定的值 $\top$ 会原封不动地依次移动 $Q^{l'''}[l''' - (i-1-t_2)]$。因此，在时间 $i-1$，单元 $Q^{i-1}[t_2]$ 中的值为 $\top$。

在下一个时间步，当插入 $\mathcal{M}$ 时，判定结果 $r^{i-1-t_2}$ 会按要求移至 $Q^i[t_2+1]$，因为抽象机不会修改单元 $t_2+1$。

(假设 $x^{i-t_2-1} \nvDash \alpha\ \mathcal{U}_{[t_1,t_2]}\beta$):

这种情况可能发生在三种场景中。我们分别考虑每种场景。

场景*1*: 设存在某个 $k$，使得 $[i-1-t_2, i-1-t_2+t_1-1]$ $x^k \nvDash \alpha$。我们按如下方式考虑此区间:

- 在时间 $i-1-t_2$，根据抽象机的定义，值 $\mathcal{M}$ 被插入到索引 0 处的队列中。该值对应于判定结果 $r^{i-1-t_2}$。
- 对于所有时间 $l' \in [i-1-t_2, k-1]$，我们知道 $x^{l'} \vDash \alpha$，并且根据抽象机的定义，单元格 $[0, t_1-1]$ 中的值不会改变。设定在 $Q^{i-1-t_2}[0]$ 中的值 $\mathcal{M}$ 仅沿着 $Q^{l'}[l'-(i-1-t_2)]$ 移动而不发生改变。
- 在时间 $k$，我们知道 $x^k \nvDash \alpha$。根据抽象机的定义，$[0, t_1-1]$中的所有单元格都设为 $\perp$。这包括单元格 $Q^k[k-(i-1-t_2)],$，其旧值为 $\mathcal{M}$。
- 对于所有时间 $l'' \in [k+1, i-1]$，根据抽象机的定义，在 $Q^k[k-(i-1-t_2)]$ 中设定的值 $\perp$ 会原样经过 $Q^{l''}[l''-(i-1-t_2)]$ 移动。因此，在时间 $i-1$，单元格 $Q^{i-1}[t_2]$ 中的值为 $\perp$。

在下一个时间步，当插入 $\mathcal{M}$ 时，判定结果 $r^{i-1-t_2}$ 会按要求移动到 $Q^i[t_2+1]$，因为抽象机不会修改单元格 $t_2+1$。

场景*2*: 令 $l$为满足 $x^l \vDash \beta$的 $[i-1-t_2+t_1, i-1]$中最早的时间实例，并且对于所有 $k'$，都有 $[i-1-t_2, i-1-t_2+t_1-1]$, $x^{k'} \vDash \alpha$，且存在某个 $k''$属于 $[i-1-t_2+t_1, l-1]$，使得 $x^{k''} \nvDash \alpha$。我们将区间 $[i-1-t_2, i-1]$ 分为五部分:

• 在时间 $i-1-t_2$，根据抽象机的定义，值 $\mathcal{M}$ 被插入队列的索引 0 处，并且该值在此时间单位内不会改变。此值对应于判定结果 $r^{i-1-t_2}$。

• 对于所有时间 $l' \in [i-1-t_2+1, i-1-t_2+t_1-1]$，我们知道 $x^{l'} \vDash \alpha$，并且根据抽象机的定义，单元 $[0, t_1-1]$ 中的值不会改变。在 $Q^{i-1-t_2}[0]$ 中设置的值 $\mathcal{M}$ 只是经过 $Q^{l'}[l' - （i-1-t_2）]$，保持不变。• 对于所有时间 $l'' \in [i-1-t_2+t_1, k''-1]$，我们知道 $x^{l''} \vDash \alpha$ 与 $x^{l''} \nvDash \beta$。根据抽象机的定义，单元 $[0, t_2-1]$ 中的值不会改变。在 $Q^{i-1-t_2}[0]$ 中设置的值 $\mathcal{M}$ 只是经过 $Q^{l''}[l'' - （i-1-t_2）]$，保持不变。• 在时间 $k''$，我们知道 $x^{k''} \nvDash \alpha$ 与 $x^{k''} \nvDash \beta$。根据抽象机的定义，如果当前值为 $\mathcal{M}$，则单元 $Q^{k''}[t_1, t_2]$ 中的值被设置为 $\perp$。该范围包含单元 $Q^{k''}[k'' - （i-1-t_2）],$，其旧值为 $\mathcal{M}$。

• 根据抽象机的定义，对于所有时间 $l''' \in [k''+1, i-1]$，在 $Q^{k''}[k''-(i-1-t_2)]$ 中设定的值 $\perp$ 会原封不动地在 $Q^{l'''}[l'''-(i-1-t_2)]$ 中移动。因此，在时间 $i-1$，单元格 $Q^{i-1}[t_2]$ 中的值为 $\perp$。

在下一个时间步，插入 $\mathcal{M}$ 时，判定结果 $r^{i-1-t_2}$ 会按要求移动到 $Q^{i}[t_2+1]$，因为抽象机不会修改单元格 $t_2+1$。

情形 3：对于所有 $l$，若其属于 $[i-1-t_2+t_1, i-1]$，则 $x^l \nvDash \beta$；并且对于所有 $k$，若其属于 $[i-1-t_2, i-1]$，则 $x^k \vDash \alpha$。我们将区间 $[i-1-t_2, i-1]$ 分为五部分：

• 在时间 $i-1-t_2$，根据抽象机的定义，值 $\mathcal{M}$ 被插入索引 0 处的队列中，并且该值 $Q^{i-1-t_2}[0]$ 在此时间单位内不会改变。该值对应于判定结果 $r^{i-1-t_2}$。

• 对于所有时间 $l' \in [i-1-t_2+1, i-1-t_2+t_1-1]$，我们知道 $x^{l'} \vDash \alpha$，并且根据抽象机的定义，单元格 $[0, t_1-1]$ 中的值不会改变。设置在 $Q^{i-1-t_2}[0]$ 中的值 $\mathcal{M}$ 仅在 $Q^{l'}[l'-(i-1-t_2)]$ 中移动，且保持不变。• 对于所有时间 $l'' \in [i-1-t_2+t_1, i-2]$，我们知道 $x^{l''} \vDash \alpha$ 且 $x^{l''} \nvDash \beta$。根据抽象机的定义，单元格 $[0, t_2-1]$ 中的值不会改变。设置在 $Q^{i-1-t_2}[0]$ 中的值 $\mathcal{M}$ 仅在 $Q^{l''}[l''-(i-1-t_2)]$ 中移动，且保持不变。

• 在时间 $i-1$，我们知道 $x^{i-1} \vDash \alpha$ 与 $x^{i-1} \nvDash \beta$。根据抽象机的定义，单元格 $Q^{i-1}[t_2]$ 中的值被设为 $\perp$，因为当前值为 $\mathcal{M}$。

在下一个时间步，当插入 $\mathcal{M}$ 时，判定结果 $r^{i-1-t_2}$ 按要求移动到 $Q^{i}[t_2+1]$，因为抽象机不会修改单元格 $t_2+1$。

定理 1 关于 $\phi = \alpha \, \mathcal{U}_{[0,t_2]} \beta$ 可以用类似的方法证明。

□

## （定理 2的证明）

:

证明 这是因为针对所有单算子 MTL 公式 $\phi$ 构造了 $EM$。根据定理 1，时间 $j - l(\odot)$ 对应的判定结果在时间 $j$ 可用，且该判定结果位于单元格 $Q^{j}[l(\odot)]$ 中。抽象机不会更改位置大于 $l(\odot)$ 的单元格内容。因此，在时间 $j+k$，经过 $k$ 次移位后，时间 $j - l(\odot)$ 对应的判定结果仍未改变，位于单元格 $Q^{(j+k)}[l(\odot)+k]$ 中。□

**（命题1的证明）**

:

证明 从表 1中可以看出，机器 $M_1$ 修改 $I_1 = [0, t_1 - 1]$ 与 $M_2$ 修改 $I_2 = [t_1, t_2]$ 与 $I_3 = [t_2, t_2]$ 与 $M_3$ 修改 $I_4 = [t_1, t_2 - 1]$。

重叠的区间对为（$\mathcal{I}_2$，$\mathcal{I}_3$）和（$\mathcal{I}_2$，$\mathcal{I}_4$）。区间 $\mathcal{I}_2$ 和 $\mathcal{I}_3$ 由同一台机器 $M_2$针对 $\alpha_1$的不同值进行修改，但在给定的周期内这是不可能的，因此无法同时修改这两个区间。

区间 $\mathcal{I}_2$ 在 $\alpha_1 = \top$时由 $M_2$ 修改。区间 $\mathcal{I}_4$ 在 $\alpha_1 \vee \alpha_0 = \bot$时由 $M_3$ 修改，这意味着 $\alpha_1 = \bot$。因此，无法同时修改这两个区间。

□