arXiv:2603.20264v1 [cs.PL] 13 Mar 2026

大语言模型能进行综合吗?

Derek Egolf、Yuhao Zhou 和 Stavros Tripakis

美国马萨诸塞州波士顿，东北大学
{egolf.d,zhou.yuhao,stavros}@northeastern.edu

摘要。 大语言模型在程序综合任务中的表现与符号工具相比如何？我们针对多个综合领域探讨这一问题：LTL反应式综合、语法引导综合、分布式协议综合和递归函数综合。在每个领域，我们选取一种最先进的符号工具，并将其与开源的 320 亿参数通义千问模型以及专有的前沿大语言模型 GPT-5 进行比较。对于每个基准测试，我们将通义千问与符号验证器结合起来反复运行，直到它生成通过验证器检查的解，或达到超时为止。对于每个基准测试，我们仅运行 GPT-5 一次，并验证其生成的输出。在所有领域中，符号工具解决的基准测试数量都多于通义千问，且表现优于 GPT-5 或与之大致相当。在执行时间方面，符号工具在所有领域都优于 GPT-5，与通义千问相比也更快或非常接近，尽管大语言模型运行在性能显著更强的硬件上。

1 引言

程序综合的目标一直是尽可能自动地生成系统，使其 通过构造即正确，也就是说，能够保证满足给定的正确性规范 [16,26]。自 20 世纪 50 年代丘奇问题发表以来 [16], ，程序综合已经取得了长足发展，从早期的演绎式程序综合方法 [47] 和反应式综合 [56,26]，发展到近期的草图化方法 [64,63], 、语法引导综合 [3], 以及其他方法 [27]。

另一方面，编程领域本身目前正在经历快速变化；大语言模型如今已被常规用于为常见的编程任务生成代码 [51,15,72]。

这些发展自然引出了一个问题：大语言模型能进行综合吗？具体来说，我们希望评估大语言模型根据形式化规约进行程序综合的能力，并将其与符号工具进行比较。因此，我们将研究问题重新表述如下：大语言模型与最先进的符号综合工具相比表现如何？

综合领域的选择。 由于综合问题种类繁多，上述问题的答案通常取决于所选的具体综合领域。本文考察以下综合领域：

– LTL反应式综合 [56,26]：该问题旨在综合一个能够实现（实现）给定线性时序逻辑（LTL）规约的有限状态机（FSM） [55]。– 语法引导综合（SyGuS） [3]：给定一个规约和一种语法，该问题旨在综合一个可由该语法生成且满足该规约的程序。– 基于草图的分布式协议综合 [22,23]：给定一个以 TLA 表示的分布式协议草图$^+$ [38]，该问题旨在补全草图，使生成的协议满足给定的一组 TLA$^+$ 性质。– 基于草图的递归程序综合 [24]：给定一个用类LISP的ACL2s编程语言编写的递归函数草图 [20],以及一个用一阶逻辑表示的规约，该问题旨在补全草图，使生成的程序满足该规约。

大语言模型的选择与使用。 在所有领域中，我们将相应的符号工具与两个大语言模型进行比较：Qwen-2.5-Coder-32B-Instruct [30] （下文简称通义千问）和 GPT-5 [62]。通义千问是一个拥有 320 亿个参数的模型。据我们所知，它是能够在 NVIDIA H200 GPU（我们运行该模型所用的硬件）上运行的、最先进的开源代码生成大语言模型。GPT-5 是专有模型，据我们所知，OpenAI 尚未公开其使用的参数数量。GPT-3 于 2020年发布，拥有 1750亿个参数 [9], ，因此我们可以合理地推断，GPT-5 的参数数量远超这一数字。

我们强调，在所有情况下，我们使用这些大语言模型时都不做任何额外处理，也就是说，我们不会自行训练这些大语言模型，不会对其进行微调，也不会使用检索增强生成或类似技术，等等。本着探究原始问题的精神*大语言模型能进行综合吗？*，我们希望评估大语言模型能够在多大程度上进行综合，*并且完全依靠自身*，而不是探讨如何将大语言模型集成到更复杂的“混合式”“神经符号”或类似技术中。我们确实认为这类技术很重要，但它们超出了本文比较的范围。（我们会在相关工作部分第 §7节回顾相关的神经符号技术。）

与此同时，我们采取务实、以用户为中心的视角。用户可以如何使用大语言模型进行代码综合？大语言模型的输出没有任何保证，因此用户必须检查输出，确保其满足问题要求。为此，我们将大语言模型与符号验证器 相结合，由验证器检查大语言模型的输出，并在输出未通过检查时让大语言模型重试。在秉持公平原则的前提下，我们尽可能为参与比较的工具链分配相近的资源。

接下来的第 2 节将更详细地介绍我们使用大语言模型的方式，讨论公平性考量和资源分配，并说明所有领域共有的其他要素。第3–6 节展示各领域的实验结果。第7 节讨论相关工作。第8 节总结全文。

2 方法论与跨领域的共同要素

公平比较。理想情况下，要公平比较两种工具，需要在相同的计算硬件上运行这两种工具，并为它们分配相同的时间和内存资源。但在我们的案例中，这无法实现，因为与我们考察的所有符号综合工具不同，通义千问和 GPT-5 都无法在中央处理器上有效运行。我们仍希望尽可能公平地进行比较，因此我们的原则是，尽可能为所比较的工具链提供相等的计算资源。我们所说的计算资源是指硬件计算能力与用于执行计算的时间的乘积（我们通过超时来控制后者）。理想情况下，我们希望这一乘积相等。例如，如果工具 A 在速度是工具 B 两倍的硬件上运行，那么工具 B 的超时时间就应是工具 A 的两倍。然而在实践中，这种计算无法实现，因为不同硬件架构之间的硬件计算能力难以衡量和比较。此外，据我们所知，OpenAI 甚至没有披露运行 GPT-5 所用硬件的具体信息。

因此，我们采取一种务实的方法，具体如下：

– 我们在中央处理器上运行所有符号工具，每个基准测试的超时设为 10 分钟（递归程序综合领域在 §6 中为 15 分钟）。– 我们使用单个 NVIDIA H200 GPU 运行 通义千问。我们将 通义千问 与验证器（在中央处理器上运行）结合使用，由验证器检查每个由通义千问 生成的候选项。对于每个基准测试，该过程会持续迭代，直至找到正确的解，或达到 10 分钟的时间限制（§6 中为 15 分钟）。– 我们使用 OpenAI API 运行 GPT-5 。在所有领域中，我们将输出令牌预算设为 16,384，并将推理参数设为“medium”。对于每个基准测试，我们运行 GPT-5 一次，并使用验证器（在中央处理器上运行）检查生成的解，为GPT-5的生成和验证总共分配 10 分钟（§6 中为 15 分钟）。

这样做的理由是为每个工具链设置相同的超时时间，尽管这些工具链运行在不同的硬件上。考虑到 GPT-5 本身非常强大，且运行在性能强劲的硬件上，我们对每个基准测试只查询 GPT-5 一次，不像通义千问那样反复迭代。符号工具与通义千问之间的比较仍有些不公平，因为后者运行在性能强大得多的硬件上。在找到更好的比较方法论之前，我们接受这一点。

大语言模型工具链。图示而言，我们在所有领域使用的大语言模型工具链如图 1 所示。请注意，通义千问工具链包含重新提示循环，而GPT-5 工具链则不包含（GPT-5 在每项基准测试中仅查询一次）。我们按以下方式处理超时。在GPT-5 工具链中，我们为每项基准测试分配 10 分钟的总时限（§6中为 15 分钟）。例如，如果GPT-5 生成一个候选解耗时 3 分钟，那么验证器就有 7 分钟来检查该候选解。若某个解在 10 分钟内完成生成

并通过验证，则计为正确解。在通义千问工具链中，我们为每项基准测试分配 10 分钟（§6中为 15 分钟），并让循环尽可能多地迭代，直至 10 分钟用尽或找到正确解。同样，只有在时限内生成并通过验证的解才计为正确解。

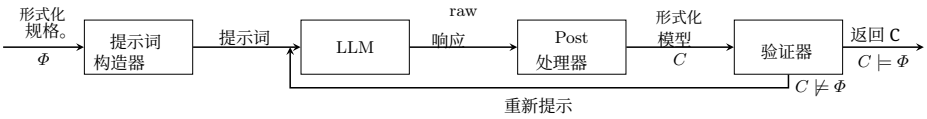


图 1：基于大语言模型的综合方法：通义千问工具链包含重新提示循环，而 GPT-5 工具链则没有（每个基准测试只查询 GPT-5 一次）。

我们将这种方法称为迭代大语言模型直至找到解或超时（ILST）。需要指出的是，对于给定的基准测试，循环每次迭代使用的提示词都保持不变。也就是说，我们不会向大语言模型提供任何验证反馈：既不给出反例，也不以其他方式说明先前生成的解有误。这不仅需要将先前生成的（错误）解添加到提示词中，也会违背我们的前提——测试大语言模型能否独立完成综合，而非将其作为神经符号循环的一部分。因此，尽管使用了温度参数和其他能让生成更加灵活、降低确定性的参数，大语言模型仍常常会在循环中多次生成相同的解（所有通义千问工具链均使用温度参数=0.8、top_p=0.95 和 top_k=50）。在某些领域中，我们会在 ILST 循环内缓存生成的解；如果同一个解生成了两次，就会跳过验证步骤，因为我们已知该解不正确。

我们还考虑过让大语言模型生成一组（例如 $k > 1$ 个）不同解，再对它们逐一验证。我们最终选择使用 ILST，原因如下。首先，在生成 k 个解的方案中，如何以有意义的方式处理验证时间预算并不明确。例如，如果大语言模型生成了 $k = 2$ 个解，第一个经模型检查后被判定为错误，而第二个经模型检查后被判定为正确，但模型检查器却超时了，该怎么办？相比之下，在 ILST 中，生成和验证所花费的时间会自动得到适当平衡。其次，ILST 更接近人类使用大语言模型的方式：生成一个解，检查它，必要时再试一次。最后，许多符号综合工具本质上是枚举式的，也就是说，它们会枚举一个候选解，对其进行检查；如果解不正确，就再试一次。这种相似性使 ILST 成为与符号工具进行比较的自然选择。

3 线性时序逻辑反应式综合

问题是综合一个能够实现（实现）给定 LTL 规范的有限状态机 [56,26]。我们采用了 SYNTCOMP 的 LTL 综合赛道中的问题格式 [32] – https://www.syntcomp.org/。SYNTCOMP 使用时序逻辑综合格式（TLSF）[31] 来描述规约。TLSF 文件声明待综合有限状态机的输入变量和输出变量，以及该有限状态机必须实现的、关于这些变量的 LTL 公式。

3.1 实验设置

符号工具。 我们将大语言模型与最先进的综合工具 ltlsynt [59], 进行比较；在 SYNTCOMP 2025 的线性时序逻辑综合赛道中，ltlsynt 解决的基准测试数量最多。ltlsynt 生成 AIGER格式的解 [8]。我们为 ltlsynt 的综合任务设置了 10 分钟超时。我们遵循 SYNTCOMP 的验证方法，并采用其验证流程检查 ltlsynt 的所有结果是否正确。首先，我们使用 SyFCo [31] 将 TLSF 输入转换为 SMV/nuXmv 格式，随后将其编译为 AIGER监视器。然后，将该监视器与候选解结合，并使用 nuXmv模型检查器 [11]检查生成的 AIGER 文件。我们为整个验证流程设置了 10 分钟超时。原则上，对于 ltlsynt 这类符号工具，由于其通过构造保证正确性，因此无需进行这种综合后验证。不过，我们仍然执行验证，以确保正确性，并与大语言模型实验保持一致。

大语言模型工具链。 由于输出格式的选择会影响大语言模型的性能，在这一领域，我们要求每个大语言模型以 AIGER [8] （SYNTCOMP 的输出格式）或 SMV（nuXmv 的输入格式）合成有限状态机。通义千问$^{\text{AIG}}$ 和 GPT-5$^{\text{AIG}}$ （分别为通义千问$^{\text{SMV}}$ 和 GPT-5$^{\text{SMV}}$）表示生成 AIGER（分别为 SMV）输出的大语言模型工具链。AIGER 工具链使用的提示词和验证器与 SMV 工具链不同。

对于 通义千问 工具链，我们使用 ILST循环（参见 §2），累计超时为 10 分钟；而对于 GPT-5 工具链，我们采用单次生成，生成和验证共用 10 分钟超时。对于通义千问工具链，我们不设令牌上限，允许模型在时间约束内生成所需的任意长度输出。对于 GPT-5工具链，我们对生成和验证设置合计 10 分钟超时。

在验证方面，AIGER工具链使用与上述ltlsynt的综合后验证相同的验证流程。任何语法错误，例如格式错误的AIGER文件，或解的输入/输出符号与TLSF规范之间的不一致，都会在此验证流程中自动检测出来。在SMV工具链中，我们使用SyFCo将TLSF规范转换为SMV格式，将其与大语言模型生成的SMV模块合并，并使用nuXmv -ctt对结果进行模型检查。-ctt标志确保转移关系是全定义的（例如，这可以防止大语言模型通过死锁生成能够平凡地满足LTL公式的解

）。我们注意到，用AIGER编写的有限状态机在构造上是确定性的，但输出SMV的大语言模型理论上可能生成非确定性有限状态机（不过，在精心设计提示词后，这种情况在我们的实验中最终从未发生）。因此，我们会检查SMV大语言模型工具链的结果是否具有确定性。我们仅在综合后执行此检查，而不在ILST循环期间执行，因为检查确定性的计算开销很大（见附录B）。在循环内部，我们完全依赖于`nuXmv -ctt`。

提示词。 对于 AIGER 大语言模型工具链，提示词要求大语言模型以 ASCII AIGER 格式，为输入的 TLSF 规范合成一个实现。提示词由一段简短的 ASCII AIGER 格式说明，以及若干展示大语言模型预期 AIGER 输出的示例组成。完整提示词见附录A.1。

SMV 工具链的提示词旨在确保大语言模型生成有效的确定性有限状态机。为此，我们施加了严格的结构约束。首先，严格规定 TLSF 概念与 SMV 各部分之间的映射：(1) 所有 TLSF 输入都必须在 输入变量部分中声明，以避免对输入变量赋值（输入变量只能由环境控制）；(2) 内部状态变量必须在 变量 部分中声明；(3) 所有变量必须为布尔型；(4) TLSF 输出必须在 定义 部分中定义。其次，为避免产生非确定性解，我们要求有限状态机仅使用一个 赋值 块定义，并为每个状态变量显式指定 `init()` 和 `next()` 赋值，同时禁止使用 `INIT` 和 `TRANS` 部分。我们还明确说明了布尔运算符语法，以尽量减少生成过程中的语法错误。最后，由于我们的基准测试套件中没有任何基准测试需要 Moore 目标有限状态机（其输出完全取决于状态），因此提示词中没有加入相关说明。完整提示词见附录 A.2。

基准测试。 我们使用 SYNT-COMP 2025 年 LTL综合赛道的基准测试。该套件包含 1586 个基准测试。其中一些基准测试被标记为可实现、不可实现或未知。在实验中，我们选取了所有标记为可实现的基准测试，共计 433 个。我们只选取可实现的基准测试，以避免让大语言模型产生混淆。

硬件与参数。 我们在一台配备 4 个 CPU 核心和 64GB 内存的机器上运行 ltlsynt。Qwen 工具链在相同的硬件配置上运行，并额外使用一块 NVIDIA H200 GPU。我们通过 OpenAI 官方 API 调用 GPT-5。

3.2 实验结果

结果见表 1。其中已解决 列报告了在 10 分钟超时内获得的正确解的数量。`ltlsynt` 为 434 个基准测试中的 396 个生成了解。我们成功对其中 354 个解进行了验证（综合后验证）；对于其余 42 个，`nuXmv`在

10 分钟后超时（此 10 分钟限制仅适用于综合后验证，不计入分配给 ltlsynt 的 10 分钟超时）。在 $433 - 396 = 37$ 个 ltlsynt 未能解决的基准测试中，它有 34 次超时（未生成解），并崩溃了 3 次（例如，由于内存分配失败）。

ltlsynt解决了143个大语言模型工具链均未能解决的基准测试。GPT-5^{AIG} 解决了2个ltlsynt和其他大语言模型工具链均未能解决的基准测试。同样，GPT-5^{SMV} 解决了3个其他方法均无法解决的基准测试。我们还观察到，ltlsynt解决了Qwen工具链成功解决的所有基准测试，而Qwen$^{\mathrm{AIG}}$ 和Qwen$^{\mathrm{SMV}}$ 分别解决了GPT-5^{AIG} 和 GPT-5^{SMV} 未能解决的1个和6个基准测试。总计434个基准测试中，有401个至少被一种方法解决。

方法	# 已解决	成功时间			成功迭代次数				失败迭代次数			
	在超时范围内	最小值	最大值	平均值	最小值	最大值	平均值	中位数	最小值	最大值	平均值	中位数
ltlsynt	354+42	0.5	301.6	4.6	–	–	–	–	–	–	–	–
通义千问$^{\mathrm{AIG}}$	4	5.4	261.5	79.6	5	136	39.2	8	2	522	41.5	7
通义千问$^{\mathrm{SMV}}$	50	1.5	575.9	164.9	1	135	23.9	9	1	202	42.1	33
GPT-5^{AIG}	150	14.9	311.4	124.0	–	–	–	–	–	–	–	–
GPT-5^{SMV}	229	13.8	465.3	107.0	–	–	–	–	–	–	–	–

表 1：433 个 LTL 反应式综合基准测试的汇总统计。

成功时间列仅针对成功案例给出执行时间统计（最小值、最大值和平均执行时间，单位为秒），即工具链生成正确解的案例。如表所示，对于执行时间指标，ltlsynt 的表现显著优于大语言模型工具链。为统计执行时间，我们将 ltlsynt 生成的全部 396 个输出均视为正确解。

成功迭代次数列给出了通义千问 ILST 循环找到解所需的最小、最大、平均和中位迭代次数（仅统计成功案例）。通义千问$^{\mathrm{SMV}}$ 有时第一次尝试就能生成正确解，而在其他情况下则最多需要尝试 135 次（请注意，大语言模型不会收到任何反馈）。失败迭代次数列针对失败案例给出了相同的统计数据。通义千问$^{\mathrm{SMV}}$ 的最小值为 1，是因为在某些情况下，nuXmv 在验证生成的第一个候选项时就超时了。

LLM SMV 解的确定性检查。 如上所述，我们在综合后验证所有 LLM SMV 解，以确保它们表示确定性有限状态机。我们自动检查了表 1中的所有已求解 LLM 生成的 SMV 基准测试的确定性，每个基准测试的超时设为 10 分钟。我们采用了标准的 自组合 技术：实例化有限状态机的两个副本，将相同的输入馈入两者，并检查所有可达全局状态下两个有限状态机的局部状态是否相同（见

附录 B）。所有 50 个 通义千问$^{\mathrm{SMV}}$ 解均在超时限制内通过验证，确认为确定性的。在 229 个GPT-5$^{\mathrm{SMV}}$ 解中，225 个通过自动检查确认为确定性。其余 4 个案例中，自动确定性检查超时，但我们通过人工检查确认了这 4 个解均具有确定性。

大语言模型响应失败。 通义千问$^{\mathrm{SMV}}$ 解决了50个基准测试，在其余 $433-50=383$ 个案例中达到10分钟超时。在这383个案例中，有276个由大语言模型生成了最终候选解，但未进行模型检查（因为已达到超时时限）。由于276占383的比例较大，我们使用nuXmv检查这些“最后一刻”解中是否有碰巧正确的。结果276个全都错误：134个存在语法错误（即nuXmv无法解析的文件），142个存在语义错误（即违反了线性时序逻辑公式）。

GPT-5$^{\mathrm{AIG}}$ 有90个案例耗尽了令牌预算，未生成任何解。在其余 $433-90=343$ 个案例中，它均在10分钟超时期限内生成了解。随后使用nuXmv验证这343个候选项（超时时限设为10分钟总时限内的剩余时间）。其中，150个通过了模型检查器（正确的解）；145个因语法错误而未通过；48个因语义错误而未通过；模型检查期间没有案例超时。

GPT-5$^{\mathrm{SMV}}$ 在 72 个案例中耗尽了令牌预算。在其余所有 $433-72=361$案例中，它都在 10 分钟时间限制内生成了解。这 361 个候选项随后由 nuXmv 进行验证（超时设置为总计 10 分钟限制内的剩余时间）。在这 361 个候选项中，229 个通过了模型检查器（正确的解）；51 个因语法错误而失败；40 个因语义错误而失败；41 个在模型检查期间超时。我们使用全新的 10 分钟超时对后 41 个案例再次运行了 nuXmv，但它们在模型检查期间仍全部超时（请注意，对于 GPT-5，我们对生成和验证总共使用 10 分钟超时，因此原始的 nuXmv 运行通常可用时间不到 10 分钟）。

GPT-5：令牌与货币成本。在 GPT-5$^{\mathrm{AIG}}$ 实验中，每个基准测试的平均输入长度为 1,831 个令牌，平均输出长度为 10,004 个令牌。433 次查询的总成本为 43.97 美元。在 GPT-5$^{\mathrm{SMV}}$ 实验中，每个基准测试的平均输入长度为 1,737 个令牌，平均输出长度为 8,930 个令牌。433 次查询的总成本为 39.58 美元。

4 语法引导综合

SyGuS 问题是：给定一份规约和一种语法，合成一个满足以下条件的程序：(1) 能由该语法生成；(2) 满足该规约 [3]。我们采用 SyGuS 输入格式（SyGuS-IF）2.1 标准 [53] 来表示 SyGuS 问题。一个 SyGuS-IF 问题描述由三个

主要部分组成：一种背景理论（例如线性整数算术）、一个或多个待合成的函数签名及可选的语法约束，以及一组待合成函数必须满足的语义约束。

4.1 实验设置

符号工具。 我们将大语言模型与最先进的工具 cvc5 [5]进行比较。它的前身 cvc4 [6] 曾在上届 SyGuS 竞赛中赢得 5 个赛道中的 4 个 [4], ，该竞赛于 2019年 举办。给定一个 SyGuS-IF 2.1 规约，cvc5尝试以 SMT-LIB 格式综合函数实现 [7]。我们使用 cvc5 的默认 INIT 设置，不向其提供任何额外的命令行参数。对于此综合任务，我们为 cvc5分配 10 分钟超时。

我们会验证 cvc5 生成的解在语法和语义上是否正确。（原则上，这并无必要，因为 cvc5 通过构造保证正确性。不过，我们仍会执行综合后验证，以确保正确性，并与大语言模型实验保持一致。）首先，自定义语法检查器会确保生成的解符合原始 SyGuS-IF 规约中定义的语法约束（如果未指定语法，则此项检查会直接通过）。其次，为验证语义正确性，我们使用自定义工具将 SyGuS-IF 规约转换为 SMT 查询；然后将该查询与综合出的函数实现结合起来，并使用 Z3 求解器 [19] 对其进行验证。我们依赖自定义工具执行这些验证步骤，因为 SyGuS-COMP 2019 提供的标准实用程序只能处理 SyGuS-IF 1.0 文件，而 cvc5 和我们的基准测试均采用 SyGuS-IF 2.1。综合后验证任务的超时设置为每个基准测试 10 分钟。

大语言模型工具链。 我们评估了一个通义千问 ILST循环和一个 GPT-5工具链，如第 §2节所述。两者均被要求生成 SMT-LIB 输出。在这两种情况下，我们都采用与 cvc5相同的验证流程：检查解是否符合语法且语义正确。只有通过这两项检查，解才算 正确。在 通义千问工具链中，ILST循环会持续迭代，直到找到正确的解；每个基准测试的累计超时时间为 10 分钟（不设令牌上限）。对于 GPT-5，我们为每个基准测试生成一个候选解，然后验证该候选解的语义正确性和语法符合性。整个生成和验证过程的总超时时间为 10 分钟。

提示词。通义千问和 GPT-5 工具链均使用相同的提示词。该提示词要求模型生成满足 SyGuS-IF 输入语法和语义约束的 SMT-LIB 函数实现。我们严格限定输出格式：大语言模型只能生成与目标综合函数签名匹配的自包含（define-fun ...）表达式，不得包含辅助函数或自然语言解释。为避免常见语法错误，提示词明确列出了有效的 SMT-LIB

原语。例如，它规定条件表达式必须使用`ite`（不接受`if`），并要求位向量字面量的类型正确（例如，`(_ bv10 32)`）。完整提示词见附录A.3。

基准测试。我们从以下目录中选取基准测试：`test/regress/cli`，该目录位于`cvc5`代码库中 [66], 其中包含的问题比 SyGuSCompetition (SyGuS-COMP) 更近期 [4],。SyGuS-COMP 最近一次举办于 2019年。我们从上述目录中提取所有可实现的 SyGuS-IF 规范，并排除需要可选 SyGuS-IF 功能的规范（参见 [53]附录 C）以及实验性`cvc5`功能。最终共提取出 148 个基准测试。

硬件与参数。我们在一台配备 4 个 CPU 核心和 64GB 内存的机器上运行 cvc5。通义千问工具链则在相同硬件配置的基础上，额外使用了一块 NVIDIA H200 GPU。GPT-5 则通过 OpenAI 官方 API 运行。

4.2 实验结果

结果汇总于表2中。已解决列报告在 10 分钟超时内找到正确解的基准测试数量。`cvc5`成功解决了 148 个基准测试中的 137 个。经综合后验证确认，这 137 个解均正确。其他各列的含义与表1中相应各列的含义相同。

cvc5 解决了 4 个大语言模型工具链无法解决的基准测试。GPT-5 解决了 10 个 cvc5 未能解决的基准测试。我们还观察到，GPT-5 能解决通义千问解决的所有基准测试，而通义千问解决了 6 个 cvc5 未能解决的基准测试。总计 148 个基准测试中，有 147 个至少被一种方法解决。

方法	已解决数量	成功时间			成功迭代次数				失败迭代次数			
	超时内	最小值	最大值	平均值	最小值	最大值	平均值	中位数	最小值	最大值	平均值	中位数
cvc5	137	0.01	104.1	1.8	–	–	–	–	–	–	–	–
Qwen	96	0.54	439.6	27.8	1	450	16.3	1	15	921	432.5	466.5
GPT-5	143	2.35	270.1	28.7	–	–	–	–	–	–	–	–

表 2：148 个 SyGuS 基准测试的汇总统计数据。

大语言模型响应失败。 通义千问工具链解决了 96 个基准测试，在其余 $148 - 96 = 52$ 个基准测试中达到 10 分钟超时。在全部 52 个失败案例中，大语言模型生成了一个最终候选解，但因超时而未能通过验证。我们使用验证器检查这些“最后一刻”解是否正确（请注意，即使验证结果正确，它们也不会被计为表 2 中的“已解决”）。这 52 个候选项中，2 个实际上正确，46 个

语义错误（即违反语法或规约约束），另有 4 个存在语法错误（即导致验证器解析错误）。

GPT-5工具链未能解决 148 个基准测试中的 5 个。在这 5 次失败中，3 次属于语义错误，另有 2 次是因为 GPT-5 超出令牌预算。

GPT-5：令牌与货币成本。对于 GPT-5工具链，每个基准测试的平均输入长度为 1,340 个令牌，平均输出长度为 1,609 个令牌。全部 148 次查询的总成本为 2.62 美元。

5 TLA$^+$ 分布式协议综合

给定一个草图，其中包含一个以 TLA$^+$ [38]描述的分布式协议，综合问题是要补全该草图，使补全后的协议满足一组给定的性质。草图是一个带有待填空位的 TLA$^+$ 协议，以及用于定义可填入这些空位的表达式集合的语法。如果协议是参数化的，例如由参与协议的节点数量参数化，我们假设这些参数被有限化为特定值，这是标准做法 [23]。

5.1 实验设置

符号工具。 我们的评估涵盖了当前最先进的符号式分布式协议综合工具 PolySemist [23]。PolySemist 使用枚举式、反例引导式 综合算法。在内部，它会枚举候选协议，运行 TLC 模型检查器 [71],，并在某个候选协议满足这些性质时停止。否则，它会继续枚举候选项。因此，PolySemist 不会返回违反用户指定性质的协议 [23]。由于 PolySemist 从语法 [23], 的语言中枚举协议，综合得到的协议必定符合该语法。

大语言模型工具链。 我们为此领域实例化了基于 GPT-5 和通义千问的工具链，并使用 TLC模型检查器 [71] 作为核心验证引擎。由于语法符合性在此领域尤为重要 [22,23],，我们的 Qwen工具链还使用了自定义语法验证器，在将响应传递给 TLC 之前，检查大语言模型的响应是否符合所提供的语法。我们使用同一个自定义语法验证器来检查 GPT-5 生成的解。此外，在此领域中，我们的通义千问 ILST 循环会缓存大语言模型响应与模型检查结果之间的映射。如果大语言模型重复生成相同内容，我们就跳过模型检查，因为可以推断 TLC 已经拒绝过该响应。

提示词。 提示词说明大语言模型应填充哪些待填空位、这些待填空位出现在哪些动作中、每个待填空位对应的语法、已完成的协议必须满足哪些属性，以及协议采用 JSON 格式时的结构。我们在附录中展示此领域的提示词模板A.4。

基准测试。 在 [23]，PolySemist 在一组包含 171 个基准测试的套件上进行了评估，这些基准测试可在以下地址公开获取 [21]。我们在评估中使用了相同的基准测试。这些基准测试涵盖七种分布式协议，其中包括两种可重新配置的 Raft 协议。基准测试要求补全协议中一个或两个动作内缺失的表达式。这些表达式要么全部是这些动作的前置条件，要么全部是后置条件，要么同时包含前置条件和后置条件。

硬件与参数。 我们在一台配备 AMD Ryzen 7 7840U 中央处理器（基准频率 3.3 GHz）和 14GB 内存的机器上运行 PolySemist 。通义千问 工具链运行在一台配备 Intel(R) Xeon(R) Platinum 8276 中央处理器（2.20GHz）、16GB 内存和 NVIDIA H200 GPU 的机器上。我们通过 OpenAI API 使用 GPT-5。对于 GPT-5，我们在与运行 PolySemist相同的机器上对从 API 获得的响应进行模型检查。所有工具链中的 TLC模型检查器均以 8 个工作线程运行，且所有工具链均可使用至少 8 个 CPU 核心。不存在其他并行性来源。对于 第 2 节 中介绍的 通义千问 与 GPT-5 工具链，我们使用所述参数。

5.2 实验结果

结果如表3所示。Solved列显示了在171个基准测试中，相应方法能够生成正确解的数量，即能够同时通过模型检查器和语法验证器验证的解。如表3所示，符号工具 PolySemist 解决的基准测试多于 GPT-5，而通义千问解决的更少。有些基准测试能够由大语言模型解决，但 PolySemist 无法解决，反之亦然。GPT-5 和通义千问合计解决了 PolySemist 无法解决的17个基准测试。其中，GPT-5 单独解决了11个，通义千问单独解决了1个，还有5个基准测试由两个大语言模型工具链共同解决，但 PolySemist 无法解决。有16个基准测试由 PolySemist 解决，但两个大语言模型工具链均无法解决。有28个基准测试由 GPT-5 和 PolySemist 共同解决，但通义千问无法解决。有4个基准测试由通义千问和 PolySemist 共同解决，但 GPT-5 无法解决。有92个基准测试由三者全部解决，另有14个基准测试三者均未解决。

表3 中其余各列的含义与表1中对应各列相同。对于PolySemist ，我们还报告了迭代统计信息，这些信息可由该符号工具获得 [23]。

仙人掌图。 除表中所列的计时统计外，3图 2 还展示了仙人掌图。每条t， c曲线上的点表示，相应的方法在每项基准测试的时间预算为 c 秒时可以解决这些基准测试（这 t 并不表示它

方法	# 已解决	成功时间（秒）			成功迭代次数				失败迭代次数			
	在超时时间内	最小	最大	平均	最小	最大	平均	中位数	最小	最大	平均	中位数
PolySemist	140	0.70	583.06	70.94	1	219	20.7	10	7	272	97.3	99
Qwen	102	2.03	587.07	35.03	1	254	12.2	1	76	717	228.9	202
GPT-5	136	17.55	275.35	102.22	–	–	–	–	–	–	–	–

表3：171 个 TLA+ 协议综合基准测试的汇总统计数据。

表示该工具总共花费 t 秒来解决 c 项基准测试）。总体趋势表明，PolySemist 往往比 GPT-5 更快地解决较简单的基准测试。长远来看，这两种工具的表现相似。

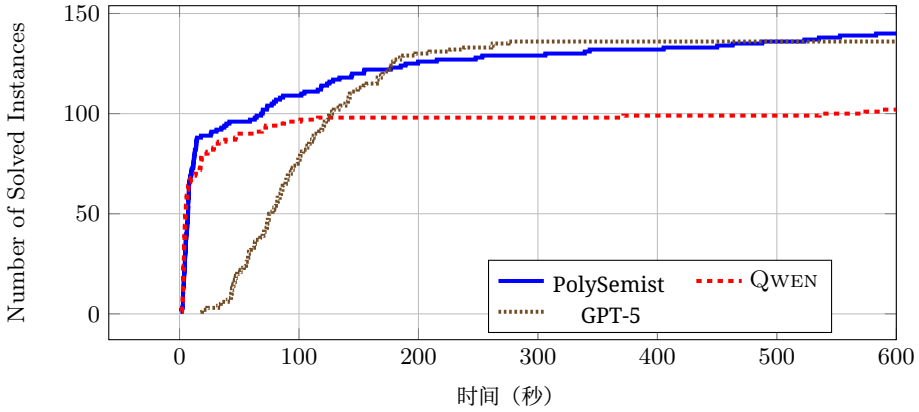


图2：如果为每个基准测试分配一定的时间预算（横轴），该图展示了每种工具在该时间内能够解决的基准测试数量（纵轴）。

大语言模型响应失败。 GPT-5 未能解决 171 个基准测试中的 35 个。在这 35 次失败中，模型检查器找到了反例；其中一次是 GPT-5 超过了令牌上限。还有一次，GPT-5 在超过 10 分钟后给出了正确的解，我们将其计为超时。

由于通义千问会迭代，我们总共收到了大语言模型的 16,824 条响应。若将每项基准测试中不同响应的集合 分别取基数后求和，可得 7,326 条不同响应。这意味着通义千问经常（错误地）重复自身。在 16,824 条响应中，有 16,722 条是失败响应。在其中 11,628 个案例中，模型检查器找到了反例。有一个案例中，通义千问在超过 10 分钟后给出了正确的解，我们将其计为超时。在 5,063 个案例中，通义千问的响应不符合所提供的语法（GPT-5 的响应始终符合所提供的语法）。在 30

个案例中，通义千问甚至没有生成语法有效的 JSON（GPT-5 始终生成语法有效的 JSON）。

放宽语法约束。我们还使用放宽后的验证引擎评估了大语言模型工具链。该引擎不检查语法符合性，而只应用 TLC 检查语义正确性。对于通义千问，这意味着在 ILST循环中，只使用 TLC 作为验证器，并在找到语义有效的解后退出循环。在这种放宽设置下，我们还从通义千问和GPT-5的提示词中省略了语法。我们发现，在此配置下，两种大语言模型工具链解决的基准测试更多。这一结果并不令人意外，因为在这种情况下，大语言模型要解决的是一个更简单的综合问题（没有语法约束）。在这一放宽设置下，通义千问解决了133项基准测试（而表3中为102项），而GPT-5 解决了149项基准测试（而表3中为136项；相比之下，Poly-Semist 解决了140项）。

GPT-5：令牌与货币成本。我们用 GPT-5-NoGram 指代不检查语法的 GPT-5 工具链（即上文讨论的放宽设置）。GPT-5 每项基准测试的平均输入词元数为 1953，而 GPT-5-NoGram 为 1220。GPT-5 每项基准测试的平均输出词元数为，而 GPT-5-NoGram 为 5724。值得注意的是，在这两种情况下，合成出的解平均只有约 70 个词元，而大多数输出词元都是所谓的“推理词元”。GPT-5-NoGram 的输入词元较少，因为提示词中不包含语法。我们注意到，GPT-5-NoGram 会生成更多推理词元；移除语法会让模型“思考”得更多。总体而言，运行本领域报告的所有 GPT-5 实验总成本约为 20 美元。

6 ACL2s递归程序综合

本领域所研究的递归程序综合问题，是补全一个或多个递归函数的草图，使由这些补全后的函数组成的程序满足给定的一组正确性属性。与 §5 中一样，草图是一个带有待填空位的程序，并附有定义可用于填充这些空位的表达式集合的语法。在本领域中，程序以类 LISP 的 ACL2s 编程语言编写 [20], 而草图和语法则采用类似的类 LISP 格式表示。正确性属性是一阶逻辑公式。详情请参阅 [24] 。

6.1 实验设置

符号工具。 我们使用 Cataclyst [24], ，一种符号工具，可根据形式化规约综合递归程序。Cataclyst 采用枚举式

反例引导的综合算法。Cataclyst 使用下文讨论的验证引擎在内部检查候选程序。因此，Cataclyst 只能输出通过验证引擎检查的程序，也无法输出语法无效的程序。

验证器与正确性保证。 在此领域，对综合得到的程序进行验证相当于一阶逻辑有效性检验，因此一般来说这是不可判定的。因此，我们采用 [24], 中的方法，并将 ACL2s 定理证明器的反例生成器用作工具链的验证引擎 [20,13]。由此可知，CATACLYST原则上可能返回一个通过反例生成器检验、但实际上并不正确的程序。 [24] 的作者使用 ACL2s 定理证明器，对 [24] 中综合得到的所有程序进行了事后手动验证，并发现这些程序确实全部正确。因此，对于 [24] 中的基准测试（我们在此也使用这些基准测试），我们姑且认为，通过反例生成检验足以作为全面验证的可靠替代。

大语言模型工具链。 与先前的领域一样，大语言模型工具链的整体结构与图1所示相同。我们实例化了两种大语言模型工具链：一种使用通义千问，另一种使用GPT-5。两种工具链都采用宽松设置，即检查正确性属性，但不检查所提出的程序是否符合给定的草图或语法。在这一领域，草图和语法主要用于引导CATACLYST进行枚举搜索。只要综合得到的程序在语法上有效，它们是否符合语法就无关紧要。不过，为了公平起见，我们将草图和语法作为输入提供给大语言模型工具链。在此，我们还会缓存大语言模型的响应；如果大语言模型重复作答，则跳过验证。

提示词。对于此领域，提示词模板会提供：（1）待综合函数的函数签名；（2）可用于综合函数的原始函数列表；（3）可用于综合函数的终端（常量和函数参数）；（4）综合函数必须满足的属性和输入输出示例列表；（5）关于数据类型和函数的辅助信息，这些信息可为大语言模型提供有用的上下文；以及（6）每个函数的草图，以及用于填充草图中待填空位的语法。我们使用的完整提示词模板见附录A.5。

基准测试。 我们使用了用于评估 CATACLYST 的 42 项基准测试，见 [24]。这些基准测试包含用于操作列表、树及其他归纳数据结构的程序。有些基准测试涉及相互递归的函数，还有些基准测试涉及带有存在量词的性质。

硬件与参数。 我们在一台配备 AMD Ryzen 7 7840U 中央处理器（基础频率 3.3 GHz）和 14GB 内存的机器上运行了 Cataclyst 。 通义千问的硬件配置与 §5 中所述的 TLA$^+$ 综合领域配置相同，但仅分配了一个中央处理器核心。GPT-5 工具链在与 Cataclyst 相同的机器上运行验证，并采用 §2中所述的相同大语言模型参数。符号工具和验证器均未配置为使用并行性。

6.2 实验结果

我们的结果总结于表4中。本领域的所有方法均采用 15 分钟超时，而不是先前领域使用的 10 分钟超时。已解决 列报告了通过 ACL2s 反例生成器的解。其他各列的含义与先前领域相应表格中的相同。如表4所示，通义千问 解决了 42 个基准测试中的 39 个，而 GPT-5 和 Cataclyst 解决了 42 个基准测试中的 41 个。通义千问 在以下三个基准测试上失败：“pairs”、“split-on”和“suffixb” [24]。GPT-5 仅在“suffixb”上失败。Cataclyst 仅在“ternary-tree-eq”上失败。因此，两个大语言模型工具链都难以处理“suffixb”，而Cataclyst 能够解决该问题；但两个大语言模型工具链都成功解决了“ternary-tree-eq”，而Cataclyst 无法解决该问题。

方法	# 已解决	成功时间（秒）			成功迭代次数				失败迭代次数			
	在超时范围内	最小	最大	平均	最小	最大	平均	中位数	最小	最大	平均	中位数
Cataclyst	41	12.60	110.23	39.11	4	2552	495.6	173	4795	4795	4795	4795
Qwen	39	3.23	30.42	9.03	1	6	1.3	1	179	516	369.7	414
GPT-5	41	17.33	124.16	48.33	–	–	–	–	–	–	–	–

表 4：42 个 ACL2s递归程序综合基准测试的汇总统计数据。

迭代与失败。如表4所示，在该领域中，通义千问解决基准测试时，往往只需少数几次迭代（以及相应的大语言模型查询）。另一方面，Cataclyst 解决基准测试时，通常会枚举数百个候选解。若通义千问未能解决基准测试，则通常是在提出数百个错误解之后才失败：三个未解决的基准测试分别提出了179、414和516个候选解。唯一一次Cataclyst失败时，它枚举了4795个候选解。

我们还统计了通义千问在三个失败基准测试中提出的不同解的数量：48/179、7/414、2/516。也就是说，在其中一项失败任务中，通义千问被要求提供解共179次，除48次外，它给出的解此前都已经给过。总体而言，我们在所有基准测试中共收到了通义千问的1159条响应。其中只有165条是不同的，因此通义千问多次重复了自己的回答。

我们从通义千问收到的1159条响应中，有1120条是失败响应。其中501次失败是因为反例生成器证明了所提出的解不正确。另有619次失败是由于大语言模型给出的响应语法无效，例如括号不匹配。GPT-5唯一一次失败，是因为发现了一个反例。

GPT-5：令牌与货币成本。对于 GPT-5，每个基准测试平均使用 681 个输入词元，解平均由 87 个输出词元组成，而 GPT-5 每个基准测试平均输出 1761 个推理词元。全部 42 个基准测试的总费用约为 0.80 美元。

7 相关工作

在 LTL 反应式综合领域，相关研究 [61,60,18] 探索神经网络方法，并在 SYNTCOMP 基准测试上对其进行评估。所有这些研究都训练或微调 *Transformer*模型以综合 AIGER 电路，而我们评估的是未经额外训练的现成模型，并同时考察 AIGER 和 SMV 输出。

在 SyGuS 领域，Li等人 [41] 提出了一种混合式方法，将大语言模型调用整合到加权概率枚举搜索中。他们的评估表明，cvc5 的表现优于单独使用 GPT-3.5，而他们的混合式方法表现优于两者。我们的工作将 cvc5 与较新的 GPT-5 进行比较；在我们的实验中，GPT-5 的表现优于 cvc5。我们还与拥有 320 亿参数的 Qwen-2.5-Coder 进行比较，而 [41]仅与 GPT-3.5 进行比较。最后，我们使用的基准测试集与 [41]所用的不同。

据我们所知， [22,23] 是仅有的在 TLA$^+$ 中综合协议的工作，且它们使用符号工具完成此项工作，未提及大语言模型。GenAI 加速的 TLA+ 挑战赛 [69] 旨在“征集展示 TLA$^+$、形式化方法与 AI辅助开发交叉领域中富有创意且技术上令人印象深刻的工作的投稿”，并于2025年公布获奖者。一等奖得主 Specula 以程序源代码（例如 Rust）为输入，并输出 TLA$^+$。由 Gregory Terzian 开发的三等奖作品以 TLA$^+$ 为输入，并生成 Rust 代码。这些工具利用大语言模型解决 TLA$^+$ 代码生成问题，但这些问题不同于本文所研究的协议综合问题。

[46,2,28,52,25] 从输入输出示例中综合递归程序。[49,48,57,37,24] 根据属性综合递归程序。这些工作均未比较大语言模型与符号工具。

还有其他综合领域不在本文讨论范围之内。大语言模型被用于生成 [12,35], 类不变式和循环不变式、在 [65],电子表格公式中 [34], , 以及在 [68]中执行 C程序修复。这些工作解决的是不同的综合问题。

上文引用的若干研究可称为“混合式”或“神经符号”研究，因为它们结合了神经网络和符号工具（包括验证器），但采用的方式比我们的迭代大语言模型直到

解或超时方法更复杂）。神经符号综合 [14] 是一个活跃的研究方向，但超出了我们当前评估的范围。

一些研究针对不同领域的大语言模型代码生成提出了基准测试，包括通用编程任务 [15,72,51,58,42,29],以及与本文更为密切相关的代码生成任务。 [70,67,10,44]中的研究针对 Lean、 [50], Dafny、 [40],或 Verus 等面向验证的语言提出了基准测试 [39]。 [54], 提出了 Solidity 智能合约的大语言模型代码生成基准测试, [36,1,33,43,45]则针对硬件领域提出了相关基准测试。其中一些研究根据正确性或效率对大语言模型生成的代码进行评估，另一些则使用验证工具检查正确性。然而，这些研究的重点并非比较大语言模型与符号工具，且其研究领域也不同于本文所考察的领域。

8 结论

我们在多个综合领域中比较了符号综合工具与大语言模型工具链。尽管结果因领域而异，但可以得到以下观察结果（参见表1、2、3、4 及相应章节）：

(1)在所有领域中，符号工具解决的基准测试都比通义千问工具链多。在线性时序逻辑综合领域，差距尤为显著。

(2) 在线性时序逻辑综合领域，ltlsynt解决的基准测试明显多于GPT-5工具链。在其他三个领域中，GPT-5解决的基准测试数量与对应的符号工具几乎相同。

(3) 通常，有些基准测试可以由大语言模型解决，但符号工具无法解决，反之亦然。从实践角度来看，这表明可以采用组合策略，让符号工具链和基于大语言模型的工具链并行运行。

(4) 尽管GPT-5运行在性能更强（且专有）的硬件上，符号工具在所有领域中的运行速度仍然快于GPT-5工具链。在线性时序逻辑综合和SyGuS领域，差距分别达到两个和一个数量级。

(5) 尽管从上面的表格中可能无法立即看出这一点，但在递归程序综合以外的所有领域，符号工具也比 通义千问 工具链更快。要了解这一点，首先要注意，表格展示的计时统计仅涉及成功运行。要全面了解情况，必须推算完整的执行时间。例如，来看表 3。 通义千问 解决了 102 个基准测试，平均每个基准测试耗时 35.03 秒。但通义千问在另外 69 个基准测试上也超时，每个都运行了 600 秒。总计，通义千问运行了$35.03 \cdot 102 + 600 \cdot 69 = 44973$ 秒，并在这段时间内成功解决了 102 个基准测试。相比之下，PolySemist 总共运行了 $70.94 \cdot 140 + 600 \cdot 31 = 28532$ 秒，并在这段时间内成功解决了 140 个基准测试。表 4也可以进行相同的计算。 通义千问运行了 2152.17 秒，略短于 Cataclyst的 2203.51 秒；但与Cataclyst（使用中央处理器）相比，通义千问 使用的硬件（图形处理器）性能强大得多，而且通义千问解决的基准测试更少（39 个，而非 41 个）。

(6) 通义千问 工具链常常多次重复提出同一个错误方案（参见 §5 和 §6）。采用枚举式综合算法的符号工具有一项优势：可以将其设计成永不重复同一个候选方案。在我们的 通义千问 工具链中，我们会多次采样，但不会向大语言模型展示之前提出的方案并要求它给出新方案。这样做或许能够减少重复提案的数量，但大语言模型对此并无保证。此外，随着提案数量增多，由于令牌数量和上下文窗口限制，将之前的提案拼接起来可能会变得不可行。

(7) 除线性时序逻辑领域外，GPT-5 在满足我们为其设定的语法预期（格式良好的代码、语法/草图符合性）方面表现相当出色。通义千问难以满足这些预期。即使在递归程序综合领域，尽管它最终解决了许多基准测试，但其失败案例中仍有一半以上是由语法问题导致的。符号工具的设计目标是始终满足语法预期，这是它的一项优势。

致谢。 本研究部分得到了美国国家科学基金会研究生研究奖学金项目第1938052号的资助。本研究还部分得到了美国国家科学基金会 FMitF（实地形式化方法）项目的资助，资助编号为 2319500 和 2525087。本材料中表达的任何意见、研究结果、结论或建议均属于作者，不一定反映美国国家科学基金会的观点。

参考文献

1. Abdelatty, M., Nouh, M., Rosenstein, J.K., Reda, S.: Pluto：用于评估大语言模型生成的硬件代码效率的基准测试。arXiv预印本 arXiv:2510.14756（2025年） 2. Albarghouthi, A., Gulwani, S., Kincaid, Z.: 递归程序综合。载于：Sharygina, N., Veith, H.（编）《计算机辅助验证——第25届国际会议，CAV 2013》，俄罗斯圣彼得堡，2013年7月13日至19日。论文集。《计算机科学讲义》，第8044卷，第934–950页。施普林格（2013年），https://doi.org/10.1007/978-3-642-39799-8_673. Alur, R., Bodík, R., Juniwal, G., Martin, M., Raghothaman, M., Seshia, S., Singh, R., Solar-Lezama, A., Torlak, E., Udupa, A.: 语法引导综合。载于：计算机辅助设计中的形式化方法，FMCAD。第1–17页（2013年） 4. Alur, R., Fisman, D., Padhi, S., Singh, R., Udupa, A.: SyGuS-COMP 2018：结果与分析。arXiv预印本 arXiv:1904.07146（2019年） 5. Barbosa, H., Barrett, C.W., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mo- hamed, A., Mohamed, M., Niemetz, A., Nötzli, A., Ozdemir, A., Preiner, M., Reynolds, A., Sheng, Y., Tinelli, C., Zohar, Y.: cvc5：一款通用且达到工业级强度的SMT求解器。载于：Fisman, D., Rosu, G.（编）《系统构造与分析工具和算法——第28届国际会议，TACAS 2022》。《计算机科学讲义》，第13243卷，第415–442页。施普林格（2022年） 6. Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C.: cvc4。载于：计算机辅助验证国际会议。第171–177页。施普林格（2011年）

7. Barrett, C., Stump, A., Tinelli, C., 等：SMT-LIB标准：2.0版。载于：《第8届理论可满足性国际研讨会论文集》（英国爱丁堡）。第13卷，第14页（2010年） 8. Biere, A.: AIGER与反相图（AIG）格式，版本20071012（2007年） 9. Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Nee- lakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D.M., Wu, J., Win- ter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., Amodei, D.: 语言模型是少样本学习者（2020年），https://arxiv.org/abs/2005.1416510. Bursuc, S., Ehrenborg, T., Lin, S., Astefanoaei, L., Chiosa, I.E., Kukovec, J., Singh, A., Butterley, O., Bizid, A., Dougherty, Q., 等：形式化验证编码的基准测试：形式化验证的程序综合。arXiv预印本arXiv:2509.22908 （2025 年） 11. Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: nuXmv符号模型检查器。载于：计算机辅助验证国际会议。第334–342页。施普林格（2014年） 12. Chakraborty, S., Lahiri, S., Fakhoury, S., Lal, A., Musuvathi, M., Rastogi, A., Senthilnathan, A., Sharma, R., Swamy, N.: 对用于程序验证的大语言模型生成循环不变量进行排序。载于：计算语言学协会：EMNLP 2023论文集。第9164–9175页（2023年） 13. Chamarthi, H.R., Dillinger, P.C., Kaufmann, M., Manolios, P.: 集成测试与交互式定理证明。载于：Hardin, D.S., Schmaltz, J.（编）《第10届ACL2定理证明器及其应用国际研讨会论文集，ACL2 2011》，美国得克萨斯州奥斯汀，2011年11月3日至4日。EPTCS，第70卷，第4–19页（2011年），https://doi.org/10.4204/EPTCS.70.114. Chaudhuri, S.: 神经符号程序综合。《神经符号人工智能与知识图谱手册》**400**，532–549（2025年） 15. Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H.P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F.P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W.H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A.N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., Zaremba, W.: 评估基于代码训练的大语言模型。CoRR **abs/2107.03374**(2021年), https://arxiv.org/abs/2107.0337416. Church, A.: 递归算术在电路综合问题中的应用。载于：《符号逻辑暑期学院概要》。第1卷，第3–50页（1957年） 17. Clarkson, M.R., Schneider, F.B.: 超属性。《计算机安全杂志》**18**(6)，1157–1210（2010年） 18. Cosler, M., Hahn, C., Omar, A., Schmitt, F.: Neurosynt：用于反应式综合的神经符号组合求解器。载于：Finkbeiner, B., Kovács, L.（编）《系统构造与分析工具和算法》。第45–67页。施普林格自然瑞士，Cham（2024年） 19. De Moura, L., Bjørner, N.: Z3：一种高效的SMT求解器。载于：系统构造与分析工具和算法国际会议。第337–340页。施普林格（2008年）

20. Dillinger, P.C.、Manolios, P.、Vroon, D.、Moore, J.S.：ACL2s：“ACL2 Sedan”。载于：第29届国际软件工程会议（ICSE 2007），美国明尼苏达州明尼阿波利斯，2007年5月20日至26日，会议伴随论文集。第59–60页。IEEE计算机学会（2007142110110920072025 2025），https://doi.org/./ICSECOMPANION..。Egolf, D.：PolySemist TACAS Artifact（）。https://doi.org/10.5281/zenodo.14618423, 14618423 22105281https://doi.org/./zenodo.。Egolf, D.、Schultz, W.、Tripakis, S.：通过草图化高效综合符号分布式协议。载于：FMCAD 2024：计算机辅助设计中的形式化方法（2024）23. Egolf, D.、Tripakis, S.：通过解释约简加速协议综合并检测不可实现性。载于：第31届系统构造与分析工具和算法国际会议（TACAS）（2025）24. Egolf, D.、Tripakis, S.：从草图和混合量词性质进行递归程序综合（2026040452 52601），https://arxiv.org/abs/.。Feser, J.K.、Chaudhuri, S.、Dillig, I.：从输入输出示例综合数据结构转换。载于：Grove, D.、Blackburn, S.M.（编）：第36届ACM SIGPLAN编程语言设计与实现会议论文集，美国俄勒冈州波特兰，2015年6月15日至17日。第229–239页。ACM（2015），https://doi.org/102737977 2611452737924./.。Finkbeiner, B.：反应式系统的综合。载于：Esparza, J.、Grumberg, O.、Sickert, S.（编）：可靠软件系统工程。《北约和平与安全科学系列》D辑：信息与通信安全，第45卷，第72–98页。IOS Press（2016）27. Gulwani, S.、Polozov, O.、Singh, R.：程序综合。《编程语言基础与趋势》**4**（1-2），1–119（2017）。https://doi.org/10.1561/25000000102 8. Hong, Q.、Aiken, A.：使用参数态射进行递归程序综合。《ACM编程语言汇刊》**8**（PLDI），102–125（2024），https://doi.org/10.1145/365638129. Huang, D.、Qing, Y.、Shang, W.、Cui, H.、Zhang, J.M.：EffiBench：对自动生成代码的效率进行基准测试。《神经信息处理系统进展》**37**，11506–11544（2024）30. Hui, B.、Yang, J.、Cui, Z.、Yang, J.、Liu, D.、Zhang, L.、Liu, T.、Zhang, J.、Yu, B.、Lu, K.等：Qwen2.5-Coder技术报告。arXiv预印本 arXiv:2409.12186（2024）31. Jacobs, S.、Klein, F.、Schirmer, S.：高级LTL综合格式：TLSF v1.1。arXiv预印本 arXiv:1604.02284（2016）32. Jacobs, S.、Pérez, G.A.、Abraham, R.、Bruyere, V.、Cadilhac, M.、Colange, M.、Delfosse, C.、van Dijk, T.、Duret-Lutz, A.、Faymonville, P.等：反应式综合竞赛（SYNTCOMP）：2018–2021。《软件工具与技术转移国际期刊》**26**（5），551–567（2024）33. Jin, P.、Huang, D.、Li, C.、Cheng, S.、Zhao, Y.、Zheng, X.、Zhu, J.、Xing, S.、Dou, B.、Zhang, R.等：RealBench：使用真实世界IP设计对Verilog生成模型进行基准测试。arXiv预印本 arXiv:2507.16200（2025）34. Joshi, H.、Ebenezer, A.、Sanchez, J.C.、Gulwani, S.、Kanade, A.、Le, V.、Radiček, I.、Verbruggen, G.：Flame：用于电子表格公式的小型语言模型。载于：AAAI人工智能会议论文集。第38卷，第12995–13003页（2024）35. Kamath, A.、Senthilnathan, A.、Chakraborty, S.、Deligiannis, P.、Lahiri, S.K.、Lal, A.、Rastogi, A.、Roy, S.、Sharma, R.：使用大语言模型寻找归纳循环不变式。arXiv预印本 arXiv:2311.07948（2023）

36. Kang, M.、Liu, M.、Hamad, G.B.、Suhaib, S.M.、Ren, H.：FVEval：理解语言模型在数字硬件形式化验证方面的能力。载于：2025年欧洲设计、自动化与测试会议（DATE）。第1–6页。IEEE（2025）37. Kneuss, E.、Kuraj, I.、Kuncak, V.、Suter, P.：递归函数模综合。载于：Hosking, A.L.、Eugster, P.T.、Lopes, C.V.（编）：2013年ACM SIGPLAN面向对象编程系统、语言与应用国际会议论文集，OOPSLA 2013，SPLASH 2013组成部分，美国印第安纳州印第安纳波利斯，2013年10月26日至31日。第407–426页。ACM（2013），https://doi.org/10.1145/2509136.250955538. Lamport, L.：《系统规约：面向硬件与软件工程师的TLA+ 语言与工具》。Addison-Wesley（2002年6月）39. Lattuada, A.、Hance, T.、Cho, C.、Brun, M.、Subasinghe, I.、Zhou, Y.、Howell, J.、Parno, B.、Hawblitzel, C.：使用线性幽灵类型验证Rust程序。《ACM编程语言汇刊》**7**（OOPSLA1），286–315（2023）40. Leino, K.R.M.：Dafny：用于函数正确性的自动程序验证器。载于：Clarke, E.M.、Voronkov, A.（编）：逻辑、编程、人工智能与推理。《计算机科学讲义》，第6355卷，第348–370页。施普林格（2010 10 1007 978 3 642 17511 4_20），41 https://doi.org/./----。Li, Y.、Parsert, J.、Polgreen, E.：利用大语言模型引导枚举式程序综合。载于：计算机辅助验证国际会议。第280–301页。施普林格（2024）42. Liu, J.、Xie, S.、Wang, J.、Wei, Y.、Ding, Y.、Zhang, L.：评估语言模型的高效代码生成能力。arXiv预印本 arXiv:2408.06450（2024）43. Liu, M.、Pinckney, N.、Khailany, B.、Ren, H.：VerilogEval：评估用于Verilog代码生成的大语言模型。载于：2023年IEEE/ACM计算机辅助设计国际会议（ICCAD）。第1–8页。IEEE（2023）44. Loughridge, C.R.、Sun, Q.、Ahrenbach, S.、Cassano, F.、Sun, C.、Sheng, Y.、Mudide, A.、Misu, M.R.H.、Amin, N.、Tegmark, M.：DafnyBench：形式化软件验证基准。《机器学习研究汇刊》（2025），https://openreview.net/forum?id=yBgTVWccIx 45. Lu, Y.、Liu, S.、Zhang, Q.、Xie, Z.：RTLLM：使用大语言模型进行RTL设计生成的开源基准。载于：2024年第29届亚洲及南太平洋设计自动化会议（ASP-DAC）。第722–727页。IEEE（2024）46. Lubin, J.、Collins, N.、Omar, C.、Chugh, R.：通过实时双向评估进行程序草图化。《ACM编程语言汇刊》**4**（ICFP），109:1–109:29（2020），https://doi.org/103408991 471145./。Manna, Z.、Waldinger, R.：程序综合的演绎方法。《ACM程序设计语言与系统汇刊》**2**（1），90–121（1980年1月）。https://doi.org/10.1145/357084.35709048. Miltner, A.、Nuñez, A.T.、Brendel, A.、Chaudhuri, S.、Dillig, I.：使用天使执行自底向上综合递归函数式程序。《ACM编程语言汇刊》**6**3498682 49129 2022101145（POPL），–（），https://doi.org/./。Miltner, A.、Wang, Z.、Chaudhuri, S.、Dillig, I.：通过约束标注树自动机实现递归程序的关系综合。载于：Gurfinkel, A.、Ganesh, V.（编）：计算机辅助验证——第36届国际会议，CAV 2024，加拿大魁北克省蒙特利尔，2024年7月24日至27日，论文集，第III部分。《计算机科学讲义》，第14683卷，第41–63页。施普林格（2024），https://doi.org/10. 1007 978 3 031 65633 0_3 /----。Moura, L.d.、Ullrich, S.：Lean 4定理证明器与编程语言。载于：自动演绎国际会议。第625–635页。施普林格（2021）

51. OpenAI, :, El-Kishky, A., Wei, A., Saraiva, A., Minaiev, B., Selsam, D., Dohan, D., Song, F., Lightman, H., Clavera, I., Pachocki, J., Tworek, J., Kuhn, L., Kaiser, L., Chen, M., Schwarzer, M., Rohaninejad, M., McAleese, N., o3贡献者, Mürk, O., Garg, R., Shu, R., Sidor, S., Kosaraju, V., Zhou, W.：大型推理模型的竞赛编程（2025年），https://arxiv.org/abs/2502.0680752. Osera, P., Zdancewic, S.：类型与示例引导的程序综合。载于：Grove, D., Blackburn, S.M.（编），第36届ACM SIGPLAN编程语言设计与实现会议论文集，美国俄勒冈州波特兰，2015年6月15日至17日。第619–630页。ACM（2015年），https://doi.org/10.1145/2737924.273800753. Padhi, S., Polgreen, E., Raghothaman, M., Reynolds, A., Udupa, A.：SyGuS语言标准2.1版。arXiv预印本 arXiv:2312.06001（2023年）54. Peng, Z., Yin, X., Qian, R., Lin, P., Liu, Y., Zhang, H., Ying, C., Luo, Y.：SolEval：面向仓库级Solidity智能合约生成的大语言模型基准测试。载于：2025年自然语言处理经验方法会议论文集。第4388–4411页（2025年）55. Pnueli, A.：程序的时序逻辑。载于：第18届年度计算机科学基础研讨会论文集。第46–57页。SFCS '77（1977年）56. Pnueli, A., Rosner, R.：反应式模块的综合。载于：ACM POPL研讨会（1989年）57. Polikarpova, N., Kuraj, I., Solar-Lezama, A.：从多态细化类型进行程序综合。载于：Krintz, C., Berger, E.D.（编），第37届ACM SIGPLAN编程语言设计与实现会议论文集，PLDI 2016，美国加利福尼亚州圣巴巴拉，2016年6月13日至17日。第522–538页。ACM（2016年），https://doi.org/10.1145/2908080.290809358. Qiu, R., Zeng, W.W., Ezick, J., Lott, C., Tong, H.：大语言模型生成的代码效率如何？一项严格且高标准的基准测试。载于：第十三届学习表示国际会议（2025年），https://openreview.net/forum?id=suz4utPr9Y59. Renkin, F., Schlehuber-Caissier, P., Duret-Lutz, A., Pommellet, A.：剖析ltl-synt。系统设计中的形式化方法 **61**(2)，248–289（2022年）60. Schmitt, F., Cosler, M., Finkbeiner, B.：使用预训练语言模型进行神经电路综合。载于：首届深度学习辅助验证国际研讨会（2023年）61. Schmitt, F., Hahn, C., Rabe, M.N., Finkbeiner, B.：根据规约模式进行神经电路综合。载于：Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., Vaughan, J.W.（编），神经信息处理系统进展。第34卷，第15408–15420页。Curran Associates, Inc.（2021年），https://proceedings.neurips.cc/paper_files/paper/2021/file/8230bea7d54bcdf99cdfe85cb07313d5-Paper.pdf62. Singh, A., Fry, A., Perelman, A., Tart, A., Ganesh, A., El-Kishky, A., McLaughlin, A., Low, A., Ostrow, A., Ananthram, A., 等：OpenAI GPT-5系统卡。arXiv预印本 arXiv:2601.03267（2025年）63. Solar-Lezama, A.：程序草图。Int. J. Softw. Tools Technol. Transf. **15**(5- 6)，475–495（2013年10月）。https://doi.org/10.1007/s10009-012-0249-7，https://doi.org/10.1007/s10009-012-0249-764. Solar-Lezama, A., Rabbah, R., Bodík, R., Ebcioğlu, K.：通过草图化为位流程序编程。载于：2005年ACM SIGPLAN编程语言设计与实现会议论文集。第281–294页。PLDI '05，ACM（2005年），http://doi.acm.org/10.1145/1065010.1065045

65. Sun, C., Agashe, V., Chakraborty, S., Taneja, J., Barrett, C.W., Dill, D.L., Qiu, X., Lahiri, S.K.：ClassInvGen：使用大语言模型进行类不变量综合。载于：Giacobbe, M., Lukina, A.（编），人工智能验证——第二届国际研讨会，SAIV 2025，克罗地亚萨格勒布，2025年7月21日至22日，论文集。《计算机科学讲义》，第15947卷，第64–96页。施普林格（2025年），https://doi.org/ 10 1007 978 3 031 99991 8_4 ./----. cvc团队：cvc代码仓库，https://github.com/cvc5 67/cvc. Thakur, A., Lee, J., Tsoukalas, G., Sistla, M., Zhao, M., Zetzsche, S., Durrett, G., Yue, Y., Chaudhuri, S.：CLEVER：形式化验证代码生成的精选基准测试。arXiv预印本 arXiv:2505.13938（2025年）68. Tihanyi, N., Jain, R., Charalambous, Y., Ferrag, M.A., Sun, Y., Cordeiro, L.C.：软件安全新时代：通过大语言模型和形式化验证实现自愈软件（ 2024 2305 14752 ），https://arxiv.org/abs/.. TLA基金会：生成式人工智能加速的TLA挑战赛（），https://foundation.tlapl.us/challenge/index.html，挑战赛说明及结果页面，访问于2026年1月23日。70. Ye, Z., Yan, Z., He, J., Kasriel, T., Yang, K., Song, D.：Verina：可验证代码生成基准测试。arXiv预印本 arXiv:2505.23135（2025年）71. Yu, Y., Manolios, P., Lamport, L.：TLA规约的模型检查。载于：Pierre, L., Kropf, T.（编），正确硬件设计与验证方法。第54–66页。施普林格柏林海德堡出版社，柏林、海德堡（1999年）72. Zhuo, T.Y., Vu, M.C., Chim, J., Hu, H., Yu, W., Widyasari, R., Yusuf, I.N.B., Zhan, H., He, J., Paul, I., 等：BigCodeBench：通过多样化函数调用和复杂指令对代码生成进行基准测试。arXiv预印本 arXiv:2406.15877（2024年）

A 提示模板

A.1 使用 AIGER 输出进行线性时序逻辑综合的提示词

系统提示词

你是一个 LTL 综合专家引擎。你的任务是根据给定的 TLSF 规约，综合生成 ASCII AIGER 格式的电路。该电路必须满足 TLSF 中指定的所有 LTL 保证。

ASCII AIGER 格式概述

AIGER（与-反相器图）是一种仅使用与门和反相器的电路表示形式。

标头：‘aag M I L O A ‘ - M =最大变量索引（使用的变量总数）- I = 输入数量 - L =锁存器数量（状态元件）- O = 输出数量 - A = 与门数量

结构：
‘ ‘ ‘
aag M I L O A<输入 > -- I 行<锁存器 > -- L 行
（如果 L > 0）<输出 > -- O 行<与门 > -- A 行<
符号 > -- 可选 c<注释 > -- 可选 ‘‘‘

文字编码

文字：**偶数 =正极性**，**奇数 =取反** - 文字 2 =变量 1（正极性），文字 3 =变量 1（取反）- 文字 4 =变量 2（正极性），文字 5 =变量 2（取反）- ** 变量索引 = 文字 / 2** - FALSE = 0，TRUE = 1

变量排序

按顺序编号的变量：输入 (1.. I) -> 锁存器 (I+1..I+L) -> 与门 (I+L+1..M)

与门定义

格式：‘lhs rhs0 rhs1 ‘
- ‘lhs ‘ 必须为**偶数**（定义新变量）
- 门计算结果：‘lhs_变量 = rhs0 与 rhs1 ‘

锁存器（时序逻辑）

格式：‘<当前 > <下一状态 >‘ - 锁存器初始化为 0，下一状态取自字面量。

示例：‘4 8‘ -> 当前状态是 var2（文字 4），下一状态来自门 8 的输出。

输入/输出符号约束（关键）

你的 AIGER 电路必须包含与 TLSF 规范**完全一致**的输入/输出符号：

1. **信号数量**：AIGER 中的输入/输出数量必须与 TLSF 输入/输出一致

2. **信号名称**：符号名称必须与 TLSF 变量名称**完全相同**

3. **示例**：如果 TLSF 中有 ‘ INPUTS { a; b; }‘ 和 ‘ OUTPUTS { outp ; } ‘：- AIGER 必须恰好有 2 个输入，符号分别为 ‘i0 a ‘ 和 ‘i1 b‘ - AIGER 必须恰好有 1 个输出，符号为 ‘o0 outp ‘

不要编造通用名称，请使用 TLSF 中的确切名称。

示例：TLSF 转 AIGER

** 输入 TLSF ** (‘ trivial_and .tlsf ‘) : “‘ INFO { TITLE : " 简单
与门 " DESCRIPTION : " 输出是两个输入的与运算 "
SEMANTICS : 米利 TARGET : 米利 } MAIN { 输入 { a; b; } 输出
{ outp ; }

保证 { G(outp <-> (a && b)); } }
‘‘‘

AIGER 输出：‘‘‘ 可
实现 aag 3 2 0 1 1 2 4
6 6 2 4 i0 a i1 b o0
outp ‘‘‘

要点：- 标头：‘aag 3 2 0 1 1‘ ->M =3(最大变量)，I =2输入，L =0锁存器，O =1输出，A=1与门 - 输入：‘2‘（a 的文字），‘4‘（b 的文字）

- 输出：‘6‘（outp 的文字，引用与门结果）
- 与门：‘6 2 4‘ -> 变量3 = 变量1 与 变量2 - 符号：‘i0 a‘、‘i1 b ‘、‘o0 outp ‘ - 与 TLSF 输入 / 输出完全匹配

应避免的常见错误

1. ** 错误的头部计数 ** - M、I、L、O、A 必须与实际内容一致

2. ** 与门输出中的文字不正确 ** - 左侧必须为偶数
3. ** 符号名称错误 ** - 必须与 TLSF 输入/输出完全一致

4. ** 缺少可实现头部** - 第一行必须是 " REALIZABLE "

输出格式

Y我们的输出tput必须 ：
1. 以“REALIZABLE ”开头
2. 后接以“ aag”开头的有效 AAG 标头
3. 包含与 TLSF 信号名称完全匹配的符号 4. 使用正确的文字编码（偶数 =为正，奇数=为否定）

5. 不要解释，不要评论——仅给出电路

用户提示

请为以下 TLSF 规范综合出 ASCII AIGER 格式的解：

```
‘‘‘ tlsf {{ tlsf_内容 }}
‘‘‘
```

输出必须以“ REALIZABLE ”开头，后接一个满足所有 LTL 保证的有效 AIGER 电路。

A.2 使用 SMV 输出进行 LTL 综合的提示词

系统提示词

你是一款专业的 LTL（线性时序逻辑）综合工具，能够解决类似反应式综合竞赛（SYNTCOMP）中的综合问题。

你的目标：生成一个 nuXmv SMV 格式的确定性有限状态机，使其满足给定的线性时序逻辑规范。

输入：一个 TLSF 规范，其中定义了输入、输出、假设、定义、不变量、LTL 保证等。

输出格式：nuXmv 模块。

严格的结构约束：

1. 模块名称：必须为 MODULE main

2. IVAR 部分：- 将所有输入变量（来自 TLSF 输入）声明为类型 ": boolean ;" - 不得使用整数型、枚举型或数组类型——只能使用布尔型变量 - 示例：IVAR i1 : boolean ; i2 : boolean ;

3. VAR 部分：- 声明任何需要的内部状态变量，类型为“: 布尔型;” - 不要使用整数型、枚举型或数组类型——仅使用布尔型变量 - 示例：VAR

state1 : 布尔型 ; state2 :
布尔型 ;

4. ASSIGN 部分：
- 对于每个内部状态变量 x，在 ASSIGN 部分添加一条 " init (x) := ... ;" 初始化语句：在 "..." 部分填入一个布尔表达式。 - 对于每个内部状态变量 x，在 ASSIGN 部分添加一条 " next (x) := ... ;" 转移函数语句：在 "..." 部分填入一个布尔表达式。 - 示例：ASSIGN init (state1) := FALSE ; init (state2) := TRUE ; next (state1) := i1 & state1 ; next (state2) := i2 | state1 ;

5. DEFINE 部分：
- 对于每个输出变量 y（来自 TLSF 输出），添加一条 "y := ... ;" 语句来定义该输出变量：在 "..." 部分填入一个布尔表达式。

- 示例：DEFINE out := i2 |
state2;

重要要求：

1. 重要：不得包含任何 INIT 或 TRANS 部分。只能包含一个 IVAR 部分、一个 VAR 部分、一个 ASSIGN 部分和一个 DEFINE 部分。

2. 布尔运算符：
非：！ （重要：使用 !，而不是 ~）
与：& （重要：使用 &，不要使用 &&）
或： | （重要：使用 | 非 | |）蕴含：->当且仅当： <->

重要：nuXmv 使用单字符运算符，而不是双字符运算符：

- 使用 ! 表示非（不是 ~）- 使用 & 表示与（不是 &&）- 使用 | 表示或（不是 ||）

有效：!a & b | c 无效： ~a && b || c（~、&&、 || 均无效）

30 D.Egolf 等人

> 注意：不要使用 IF - THEN - ELSE -FI 语法。条件判断请使用 !、& 和 | 布尔表达式。

3. 不要添加 LTLSPEC 语句，因为系统会自动根据 TLSF 添加这些语句。

4. 输入变量只能在 IVAR 部分中声明。不要在其他地方声明或定义它们。

5. 状态变量（如有需要）只能在 VAR 部分中声明。

6. 输出变量（来自 TLSF 规约的 OUTPUTS 部分）只能在 DEFINE 部分中定义。不要在 IVAR 或 VAR 中声明它们。7. 不要为输入变量赋值。它们由环境控制，而非系统控制。尤其是，绝不要在 ASSIGN 部分中使用 init (input_var) 或 next (input_var)。

8. ASSIGN 部分仅用于初始化和更新在 VAR 中声明的状态变量。在 VAR 中声明的每个变量都必须在 ASSIGN 部分中恰好有一条 init (...) 语句和一条 next (...) 语句。

9. 如果某个变量是在 IVAR 或 VAR 中声明的，则它不得出现在任何 DEFINE 语句的左侧。只有输出变量（未在其他位置声明）才能在 DEFINE 部分中定义。

10. 仅在 Markdown 代码块中输出原始 SMV 代码

示例结构：“‘ smv
MODULE main IVAR --
输入 input1：布尔型;
input2：布尔型;

VAR -- 状态变量（如有需要） state1：布尔
型；state2：布尔型;

赋值 -- 在此填写 INIT 和后续赋值 init(state1) := ... ;
init(state2) := ... ; next(state1) := ... ;

next(state2) := ... ; 定义 -- 在此填写输出
赋值 output1 := ... ; output2 := ... ; “‘

用户提示

请针对以下 TLSF 规范，以 nuXmv SMV 格式综合出解：

“‘ tlsf {{ tlsf_内容 }}
“‘

A.3 SyGuS 综合的提示词

系统提示词

你是程序综合和 SMT-LIB/SyGuS 格式方面的专家。

你的任务是综合一个 SMT-LIB 格式的函数，使其满足给定的 SyGuS 规范。

重要规则：1. 仅输出能够解决综合问题的（define-fun ...）表达式。2. 函数必须满足规约中的所有约束。3. 如果 synth-fun 中指定了语法，你的解必须仅使用该语法允许的运算。即使语法之外的运算看起来是必需的，也不得使用。4. **关键：define-fun 的签名必须与 synth-fun 的签名完全一致**——如果 synth-fun 没有参数，define-fun 也不得有参数；如果 synth-fun 有参数，define-fun 必须具有相同的参数，且类型相同；不得将文法非终结符（如 Start、StartBool）添加为 define-fun 的参数。5. 输出中不得包含（assume ...）或（declare-var ...）命令——这些不是解的一部分。6. 不得定义辅助函数。逻辑必须完整包含在所请求的函数中。7. 不得包含任何解释——仅输出 SMT-LIB 代码。

8. 关键：仅使用标准 SMT - LIB 原语。对于整数算术（LIA）： +、-、*、div、mod、比较（=、 <、 <=、 >、 >=）、ite、与、或、非。对于负数，如严格遵循 SMT - LIB 规范所需，请使用一元否定（- 5）而非 -5。警告：严格避免使用“max”、“min”、“abs”、“if”、“return”。条件语句请使用“ite”（而非“if”）。使用 ite 和比较实现“max”、“min”、“abs”。

9. 对于位向量（BV）：bvand、bvor、bvxor、bvnot、bvadd、bvsub、bvmul、bvudiv、bvurem、bvshl、bvlshr、bvult、bvule、bvugt、bvuge、concat、extract。确保字面量的类型正确，例如，使用“# x0001”或“(_ bv <value > <width >)”，其中 <value > 为十进制值， <width > 为位宽。

10. 字符串：str .++(拼接)，str .len，str .at，str . substr，str . prefixof，str . suffixof，str . contains，str . indexof，str . replace，str . to . int，int . to . str

11. 数组：store，select
12. 布尔值：使用 true 和 false（小写）。布尔型值不要使用 1 或 0。

示例：

给定以下 SyGuS 规范：“‘ sygus (set - logic LIA)

(synth - fun mi ((x 整数) (y 整数)) 整数 ((S 整数) (I 整数)) ((S 整数 ((ite (<=I I) I I))) (I 整数 (x y))))

(定义 - 变量 x Int) (定义 - 变量 y Int) (约束 (=>(>=x y) (=(mi x y) y))) (约束 (=>(>=y x) (=(mi x y) x)))

(检查 - 综合) “‘

正确的输出是：“‘ SMT-LIB 2

(定义 - 函数 mi ((x Int) (y Int)) Int (ite (<=x y) x y)) “‘

请注意：
- 函数签名 ‘(define - fun mi ((x Int) (y Int)) Int ...) ‘ 与 ‘(synth - fun mi ((x Int) (y Int)) Int ...) ‘ 匹配

- 文法非终结符 (S, I) 不在函数签名中 - 解满足约束：‘(=>(>=x y) (=(mi x y) y)) ‘ 和 ‘ (=>(>=y x) (=(mi x y) x)) ‘ - 解体为 ‘(ite (<=x y) x y) ‘，符合语法

输出格式：“‘ SMT-LIB 2 (define - fun <name >(<args >) < return - type > <body >) (define - fun <name2 >(<args >) < return - type > <body >) ... “‘

如果存在多个 synth -fun 声明，请全部输出。

请记住：
- 不要输出 (assume ...) 命令- 不要输出 inv - 约束（如果存在）- 不要输出文法定义- 仅输出 (define - fun ...) 表达式 - 函数签名必须与 synth - fun 签名完全匹配

用户提示

为以下 SyGuS 规范综合一个 SMT-LIB 函数：

“‘ sygus {{ sygus_ content }} “‘

重要：- 仅输出满足所有约束的 (define - fun ...) 表达式。- 严格遵循规范中定义的语法（如有）。

A.4 TLA$^+$ 分布式协议综合提示词

你是编写 TLA + 协议的专家。你必须在以下草图的 {{ list_of_holes }} 个待填空位中，填入 {{ list_of_actions }} 个动作：

{{ sketch }}

将映射代入草图后，必须生成满足以下性质的 TLA + 协议：

{{ 属性 }}

对于每个空缺，都必须使用相应的语法。下方展示了这些语法。只能按照语法规定使用括号。不恰当地省略或添加括号都是不允许的。

{{ 语法 }}

不要输出散文。不要进行解释。不要输出代码围栏。不要输出原始 TLA+ 代码。只能输出包含适当键的 JSON 结构。JSON 语法必须正确；必要时使用字符串转义。

你的最终输出必须是以下形式的 JSON 映射：

{{ json_映射_结构 }}

A.5 递归程序综合提示词

你是精通类 LISP 编程语言 ACL2s 的代码编写专家。

请按照以下模板编写函数（填入<函数体 >）：

{{ sig_block }}

你可以使用以下原始函数：

{{ primitives_block }}

你可以将这些函数应用于这些终端（常量和函数参数）：

{{ 终端_block }}

每个函数都可以调用自身，并且可以使用 if 语句。如果有多个函数，它们可以相互调用。

你的函数必须满足以下属性：

{{ properties_block }}

你的函数必须满足以下输入-输出示例：

{{ io_block }}

以下声明定义了（相互）归纳数据类型，并公开其构造器/析构器：

{{ datatype_block }}

了解某些数据类型和函数的定义可能会有帮助：

{{ definitions_block }}

草图指导：对于下面列出的每个函数，你可以选择其中任意一个草图来完成。

{{ sketches_block }}

以下语法定义了用于填充待填空位的有效表达式：

{{ grammar_block }}

现在实现函数。请勿包含任何说明文字、解释或代码围栏。只需提供新函数的定义：{{ function_list }}。

B 大语言模型生成 SMVOutput 的确定性与健全性检查

B.1 通过自组合进行确定性检查

我们利用自组合这一验证超属性的标准技术，验证大语言模型生成的`nuXmv` 模块的确定性 [17]。由于确定性要求系统在输入序列相同时产生完全相同的行为，因此我们使用自组合，在相同输入下比较模块的两个实例。

我们通过图3中的示例来说明该过程，具体如下：

1. **模块转换：** 我们将候选模块（例如图 3a）转换为子模块（例如图 3b中）。这包括将 `MODULEmain` 重命名为一个唯一名称（例如 `testmod`），并将所有声明从输入变量部分移至模块的输入中。2. **自组合：** 我们创建一个新的 `main` 模块，用于声明全局输入变量，并实例化两个经过转换的模块副本，分别记为 t_1 和 t_2 （见图 3b）。

3. **输入同步：** 我们将相同的全局输入变量绑定到两个实例，确保它们接收相同的输入序列。4. **确定性检查：** 我们验证一个 CTL 规范，该规范断言所有可达状态（AG）下，t_1 和 t_2 的内部状态变量始终相等。如果模型检查器确认该属性成立，则该模块具有确定性；否则，反例会展示一系列导致状态分歧的输入。

此检查还会捕获初始值或下一状态更新未定义的情况，这会导致非确定性。

```
MODULE main
IVAR
  i : 布尔型;
  j : 布尔型;
VAR
  x : 布尔型;
  y : 布尔型;
赋值
  初始化(x)  := 假;
  下一个(x)  := i & y;
  init(y)    := FALSE;
  next(y)    := !y | j;
定义
  out := x & y & i & j;
```

(a) 大语言模型生成的候选项 nuXmv 模块。

```
模块 testmod (i, j)
VAR
  x : 布尔型;
  y : 布尔型;
赋值
  init(x)  := FALSE;
  next(x)  := i & y;
  init(y)  := FALSE;
  next(y)  := !y | j;
定义
  out := x & y & i & j;

MODULE main
IVAR
  i : 布尔型;
  j : 布尔型;
VAR
  t1 : testmod (i, j);
  t2 : testmod (i, j);
定义
  -- 检查内部状态是否匹配
  state_eq := (t1.x = t2.x) &
              (t1.y = t2.y);

CTLSPEC AG state_eq
```

(b) 用于检查确定性的自组合新模块及 CTL 规约。

图 3：通过自组合检查确定性。

B.2 其他合理性检查

为确保大语言模型生成的用于 LTL 综合的 nuXmv 模块有效，我们在综合后验证阶段实施了以下额外的合理性检查：

– **结构检查：** 遵循提示词中的指令（附录A.2），生成的模块必须包含特定部分：`IVAR` 和`DEFINE`用于指定输入和输出，而`VAR` 和`ASSIGN` 仅在生成状态变量时才会包含。我们会解析生成的代码，以验证其中是否存在

这些部分；缺少任何部分都会将模块标记为不完整或格式错误。此外，此检查还会确认模块中不存在 TRANS 和 INIT 部分，因为提示词指令明确禁止使用它们。– **布尔变量检查：**我们会确保所有输入变量和状态变量都声明为布尔型，这符合提示词的要求。– **输入/输出映射检查：**nuXmv 模块必须使用 TLSF 规范中定义的确切输入和输出标识符。我们会提取并比较规约与生成模块中的变量名称。任何差异都会被标记为未遵守接口要求。

在我们的实验中，所有通义千问$^{\mathrm{SMV}}$或 GPT-5$^{\mathrm{SMV}}$ 生成的正确解均未发现结构问题。我们还发现，通义千问$^{\mathrm{SMV}}$ 和 GPT-5$^{\mathrm{SMV}}$生成的所有正确解中，变量都被正确声明为布尔型。关于输入/输出映射，229 个正确的 GPT-5$^{\mathrm{SMV}}$ 解与规约完全一致。然而，在 50 个正确的通义千问$^{\mathrm{SMV}}$ 解中，我们发现有 2 例变量名与 TLSF 规范不匹配。我们手动检查了这两例，并确认尽管名称不匹配，其逻辑在语义上仍然正确。因此，我们将它们计入成功解的数量。