

信息与软件工程学院

企业实习总结报告

学 号： 2021090916028

姓 名： 刘诗月

专业方向： 系统与技术

企业名称： 清华大学

实习岗位名称： 研究助理

企业指导教师： 贺飞

院内指导教师： 戴瑞婷

# 摘要

本实习报告围绕着超级优化项目展开。超级优化技术旨在进一步优化代码，为一个程序找到其最优的版本。超级优化的目标一般是汇编代码，以将底层硬件对性能的影响纳入考量。

超级优化任务是有挑战性的，这是由于程序空间是不规则而十分庞大的离散空间，在其中找到最优解或者近似最优解非常不易。现有的典型超级优化方法是基于随机化搜索的，从初始的种子程序出发不断对程序作变异，比如删除一条指令，以探索程序空间。这种方法只有一个种子，容易困在低cost高时间代价的平台期，并在遍历程序时缺乏对程序语义的理解。实习期间尝试引入种群进化算法从多种子出发搜索，并引入大语言模型来为随机化搜索生成种子程序，以期利用大语言模型理解和撰写代码的能力来生成高质量的种子。

我实现了基于种群和大语言模型的新算法，并在经典的Hacker Delight测试集上进行初步实验，其中在16%的样例上取得了更好的效果。

关键词： 超级优化，程序优化，种群进化，大语言模型，符号执行

# ABSTRACT

This internship report revolves around the superoptimization project. Superoptimization techniques aim to further optimize code, by finding the most optimal version for a program. The target programming language of superoptimization is usually assembly code, taking into account the impact of the underlying hardware on performance.

The task of superoptimization is challenging due to the program space being a vast and irregular discrete space, where finding the optimal or near-optimal solution is not easy. Existing typical superoptimization methods are based on randomized search, mutating the program from an initial seed program, such as deleting an instruction, to explore the program space. This method only has one seed and is prone to getting stuck in the low cost but high time cost plateau period, and lacks understanding of the program semantics during exploration. During the internship, I attempted to introduce population evolution algorithm to search from multiple seeds and attempts to introduce the large language models to generate seed programs for randomized search, aiming to utilize the ability of large language models to understand and write code to generate high-quality seeds.

I have implemented this new algorithm based on population and large language models, and conducted preliminary experiments on the classic Hacker Delight test set, achieving better results on 16% of the test samples.

Keywords: Superoptimization, Program optimization, Population evolution, Large language model, Symbolic execution

# 目 录

[第一章 实习概况 1](#_Toc174807337)

[1.1 实习单位与实习岗位概况 1](#_Toc174807338)

[1.1.1 实习单位概况 1](#_Toc174807339)

[1.1.2 实习岗位概况 1](#_Toc174807340)

[1.2 实习项目课题背景、价值、意义以及国内外研究现状 2](#_Toc174807341)

[1.2.1 课题背景、价值、意义 2](#_Toc174807342)

[1.2.2 国内外研究现状 3](#_Toc174807343)

[1.3 实习项目整体执行完成情况概述 6](#_Toc174807344)

[第二章 复杂工程问题归纳与实施方案可行性研究 7](#_Toc174807345)

[2.1 实习课题的目标与任务 7](#_Toc174807346)

[2.1.1 实习目标 7](#_Toc174807347)

[2.1.2 任务 7](#_Toc174807348)

[2.2 解决方案及可行性研究 7](#_Toc174807349)

[2.2.1 实施方案 7](#_Toc174807350)

[2.2.2 可行性研究 17](#_Toc174807351)

[2.3 实习课题任务相关的详细需求分析 18](#_Toc174807352)

[2.4 相关复杂工程问题总结 19](#_Toc174807353)

[第三章 针对复杂工程问题的方案设计与实现 20](#_Toc174807354)

[3.1 硬件的设计与实现 20](#_Toc174807355)

[3.2 工具软件的设计及实现 20](#_Toc174807356)

[3.3 应用软件的设计与实现 20](#_Toc174807357)

[3.3.1 应用软件架构图 20](#_Toc174807358)

[3.3.2 子系统和功能点流程图 21](#_Toc174807359)

[3.3.3 应用软件的运行流程及效果 24](#_Toc174807360)

[3.4 应用软件测试 27](#_Toc174807361)

[第四章 知识技能学习情况 30](#_Toc174807362)

[4.1 开发环境和工具 30](#_Toc174807363)

[4.2 预备知识 30](#_Toc174807364)

[4.3 新知识点学习和掌握情况 31](#_Toc174807365)

[第五章 工程协作交流情况 34](#_Toc174807366)

[第六章 工程计划管控与执行情况 35](#_Toc174807367)

[第七章 职业素养与工程伦理的学习与培养 36](#_Toc174807368)

[第八章 对软件工程实践以及软件工程领域发展的认识 37](#_Toc174807369)

[第九章 结束语 38](#_Toc174807370)

[9.1 课题完成情况、有待进一步解决的问题及方向 38](#_Toc174807371)

[9.2 本人对于企业实习的收获及体会 38](#_Toc174807372)

[参考文献 39](#_Toc174807373)

[致谢 41](#_Toc174807374)

# 第一章 实习概况

## 1.1 实习单位与实习岗位概况

### 1.1.1 实习单位概况

清华大学软件学院是2001年经国家教育部和国家计委联合发文批准成立的首批全国示范性软件学院之一，隶属于清华大学信息科学技术学院。学院以推进办学机制改革，探索软件人才培养模式，培养具有国际竞争力、高层次、创新性、应用型软件人才为使命，坚持“质量第一、素质与技术并重、理论与实践结合”的教学宗旨，以“教学立院、管理建院、学科兴院、科技强院”为办学理念，遵循“练中学、练中闯、练中创”的实践教学策略，以国家重大战略需求和国际学术前沿为导向适时更新教学内容，调整课程结果，修订配合方案，探索教学模式创新，践行精品教育。

清华大学软件学院软件形式化验证课题组感兴趣、持续研究的方向有：程序分析与验证(Program Analysis, Program Verification, Program Termination, Abstract Interpretation, Invariant Synthesis, Regression verification, etc. )、模型检验（Model Checking）、约束求解（SAT/SMT solvers）、程序综合（Program synthesis）、程序语言相关的其他问题（Program merge, Program repair, etc.)，最近在做多线程程序（Multi-threaded programs）、智能合约程序（Smart Contract）、网络路由程序（P4 programs）、硬件程序（Verilog）等相关的验证工作，也开始做一些深度学习系统形式化验证的工作。

总之，与形式化方法和程序语言相关的问题都属于课题组的研究范围。发表论文的目标包括形式化验证顶会（CAV/TACAS/FM），程序语言顶会（POPL/PLDI/OOPSLA），以及软件工程顶会（ICSE/FSE/ASE）。

### 1.1.2 实习岗位概况

研究助理岗位职责：

1、负责完成编译优化中的超级优化项目，负责项目基本框架的搭建

2、查找阅读相关文献，学习探索用更高效地方法在指令空间中搜索

3、研究完成相关项目、与华为公司进行相关接口对接、书写相关项目方案等

## 1.2 实习项目课题背景、价值、意义以及国内外研究现状

### 1.2.1 课题背景、价值、意义

超级优化指的是为一个程序找到其最优的版本，这一术语最早由 Massalin 于1987 年提出[1]。这一术语的命名，是因为在英文中对程序优化（program optimization）一词略微有误用，在英文中其字面含义原指程序的最优化（optimization）。不过，正如其中文翻译的字面含义一样，它实际被用来指代对程序的改进。这就给Massalin带来了困难，所以他只好再加上一个前缀 “超级”，将他所考虑的程序最优化的问题称作 “超级优化”（superoptimization）[2]。

超级优化可以在许多对代码性能有极高要求的底层软件或领域特定软件发挥作用，比如嵌入式系统和和超文本传输安全协议（HTTPS，hypertext transfer protocol secure）。一般来讲，程序性能优化的任务主要是由编译器来承担，但编译器带来的提升往往有限，只有极巧合的情况下才会编译出最优的程序。专家人工优化的汇编代码有时会有比编译器生成的代码更高的性能，这也从一个侧面说明了虽然现有的编译器中已经含有复杂的代码优化策略，但其所编译出的代码性能依然有提升的空间。超级优化旨在不依赖人工介入的情况下尽可能地进一步地优化程序性能。超级优化会尽可能全面地考虑程序空间中的各种程序，再借助形式化验证技术检验所找到的程序的正确性，也就是与待优化程序的等价性，以期在找到待优化程序的一个性能尽可能优的版本。

这里我们进一步明确超级优化任务的目标编程语言，经典的编译优化往往是在将高级语言翻译到低级语言的过程中或者是在高级语言和低级语言之间的中间表示（intermediate representation）上进行的，不过超级优化的目标编程语言一般是汇编语言，这有两点原因。一方面，汇编代码是编译器输出的最终结果，是编译器优化的最终目标，是真正要去执行的程序，其中囊括了诸多硬件细节，在性能攸关的场合，是需要将硬件——比如说 CPU 的流水线——对性能的影响考虑进来的；另一方面，汇编代码结构简单，最基本的语法单元只有两个：指令和标签。指令编码了计算机要去执行的操作，例如加减法或者读写内存，而标签用来标记出指令序列中的位置，借助标签和向标签去跳转的指令便可以实现高级语言中的分支、循环和函数调用等。这样形式简单的程序构成的程序空间更加规则，也就更加易于探索。

一个具体的应用场景是华为通信软件业务，超级优化技术有潜力来帮助优化其中的代码。无线通信软件需要处理各种各样的任务，比如编解码、加密解密和机器学习等，而且需要在不同的芯片架构上运行。如果要优化这些代码，开发人员既需要了解业务特征，比如程序是计算密集型的还是数据密集型的，也需要了解芯片架构和指令集，这就意味着对投入的人力和时间有着极高的要求。超级优化有希望通过自动化的方式来解决这个问题，它可以在不同的芯片架构上自动地优化代码，从而减少了人工介入的成本。并且，无线通信软件作为基础软件，其中的漏洞会带来较为严重的影响，因此有必要借助形式化验证的方法，来严格地保障所优化的代码的正确性。

实习研究的课题便是如何提升现有的超级优化技术，会从两方面入手，一方面是如何更好地探索程序空间，找到更优的程序版本，另一方面是如何提升形式化验证的能力。

### 1.2.2 国内外研究现状

1、基于程序综合的超级优化

程序综合（program synthesis）指的是自动化地构建满足给定形式化规约的程序。该过程涉及到理解规约的含义，并且在可能的程序空间中找到一个符合这些规约的实现。在超级优化的领域，程序综合被用来寻找一个在离散度量（如指令数量）上最优的程序版本，而不是简单地满足等价性规约。

通过程序综合来解决超级优化问题的方法是这样的，考虑规约：“对于任意输入，待优化程序与程序 X的输出相等，并且 X 的指令数目不超过限制L”。通过逐渐增加L的值，就可以找到指令数最少的程序实现。

Denali[2]是在超级优化问题中使用程序综合方法的先驱。它通过构建 E 图（等价图），来表示所有可能的、与待优化程序等价的程序实现。然后将性质 “E 图中存在指令数目不超过L 的与待优化程序等价的程序” 编码为一个 SAT（布尔可满足性问题）公式，再交由 SAT 求解器来求解。

Souper[3]设计了一种领域特定的中间表示（IR, intermediate representation），这种表示是 LLVM IR（一种广为使用的中间表示）的一个函数式无控制流子集。Souper 通过符号执行技术从待优化程序中提取出候选程序必须满足的条件，然后利用反例引导的归纳综合（CEGIS, counter-example guided inductive synthesis）技术，尝试生成满足这些条件且在指令数目限制之内的程序。Slumps[4]项目将 Souper 的方法移植到了 WebAssembly 上，扩展了超级优化的适用范围，为在 Web 环境中优化程序提供了新的工具。

2、基于搜索的超级优化

(1)、基于穷举式的超级优化

Massalin[1]提出超级优化这一概念时，所给出的指令序列生成算法即为穷举式的暴力搜索。由于其优化目标是要找到最短的指令序列，所以它可以遵循这样的搜索流程：先搜索长度为 1 的指令序列，再搜索长度为 2 的指令序列……直到找到一个与待优化的指令序列等价的序列，或者是长度到达了待优化的指令序列的长度，或者是搜索长度抵达了一个指定的阈值。在搜索过程中一些剪枝策略也得以被应用进来，最主要的剪枝是这样的：如果我们已知某个指令序列和一个更短的指令序列的语义等价，那前者就无需被考虑，比如 “move X, Y; move X, Y”。

GNU Superoptimizer[5]将上述方法推到了极致，它支持许多 1990 年代流行的不同的指令集，并且针对指令集的特点做了许多剪枝。

GreenThumb[6]则考虑了另一个视角的搜索问题，它将最优指令序列的生成问题转化为了一个图上的搜索问题：以多个测例下的程序状态为点，以指令为边的图，每条路径对应一个程序。这种视角使得对一些高级算法技巧的引入成为可能：通过合并等价类来减少图中点的表示，通过双向搜索来加速搜索过程，并用 SAT求解器来生成反例。这些技术显著提高了穷举式搜索的实用性，但是能处理的指令序列长度依然有限。

(2)、基于随机化搜索的超级优化

STOKE[7-8] 使用随机化搜索的方法，以扩大所能探索的程序空间的范围。随机化搜索通过一个启发式的估价函数（也被称作代价函数）来引导，其对程序的评估包括效率提升和正确性两方面，其被定义为：

(2-1)

其中，R 为待评估程序，T 为待优化程序，eq(R, T ) 为正确性项，表示两程序有多相似，可以是经过严格形式化验证的相等或者不等，也可以是通过比较在指定测例上的输出来估计出的一个相似度；perf(R, T ) 为效率项，即候选程序相对于目标程序的性能提升，可以通过运行测例或者是数指令条数来估计。

STOKE 使用 Metropolis-Hastings 算法来随机化地变异程序以探索程序空间以尝试找到代价函数最小的解，其基本流程是这样的：从某个程序出发，不断对它作随机化地变异以探索程序空间。具体来讲，从 R 出发，STOKE 依据一个提议分布来抽样出 R′：

(2-2)

这个提议分布指的便是从 R 出发作变异所能得到的程序的分布，比如假如我们只允许删除一条指令这种变异，再假设 R 中恰好有 5 条不同指令，那么在分布 q(⋅|R)中就有 5 个程序有非零的概率，每个程序的概率是 1/5。变异得到的 R′ 不一定会被接收，而是根据 R′的代价函数，再计算一个接收标准：

(2-3)

然后，我们再在 [0, 1] 中均匀采样，如果采样值不超过 α(R → R′; T )，就接受 R，否则拒绝。如果 R′ 被接受了，那优化器就相当于是移动到了 R′这个程序，它会被作为新的 R 来继续在程序空间中探索；否则的话，R 保持不变，再尝试生成新的 R′。

在随机化搜索的过程中，为了高效地计算代价函数，需要提供一个测例集合，但测例集合所覆盖的路径可能是不足的，尤其是在程序中包含循环时。为了能够完善测例，可以使用一个有界验证器来生成反例[9]，这个有界验证器会将循环的迭代次数限定在一个阈值内，从使得在两个程序不等价时，可以发现一个反例，如果确实是未通过等价性验证得到反例，那么就将反例加入测例集中。

Bunel 等人[10]使用强化学习来改进 STOKE 的随机化搜索过程。他们注意到随机化搜索的过程可以被视作为一个马尔可夫决策过程，对当前程序的变异可以看作是智能体（agent）要采取的行动，变异前后的程序的代价函数的差可以看作是环境给智能体的反馈（reward），那么提议分布其实就可以看作是智能体的策略（policy）。所以可以用强化学习的技术来在随机化搜索的过程中学习得到更好的提议分布。

基于随机化搜索的超级优化方法可以优化数十行的汇编代码，是超级优化领域目前最领先的成果。但其在投入实用时依然面临困难，最大的困难在于，由于程序空间过于庞大，尽管随机化搜索可以避免去穷尽所有程序，但其探索程序空间的能力依然有限，尤其是涉及包含控制流的程序，仅仅通过变异难以找到合适的控制流变形。

3、基于神经网络的超级优化

Shypula 等人[11]直接使用一个序列到序列（sequence-to-sequence）的标准 transformer 神经网络模型作为超级优化器，忽略程序的结构信息，将其视为词语（token）的序列。他们从 GitHub 上收集了约两万五千个函数，使用 gcc -O0 编译出未优化版本的 x86-64 汇编代码，再使用 gcc -O3 编译出一个优化版本的汇编代码，以此作为一个训练深度学习模型需要的大规模数据集。他们的训练分两个阶段：使用常见的监督学习方法来做预训练，再做微调。微调可以使用同策略（on-policy）和异策略（off-policy）的方法，前者是指在探索过程中即时学习调整，后者是指做完一些探索后，再统一学习。经典的同策略方法便是基于策略梯度的 REINFORCE[12]；Shypula 等人[11]提出了一种新的异策略方法，面向优化的自模仿学习（SILO, selfimitation learning for optimization）。它的每轮迭代会包含两步，一个探索步骤和一个学习步骤，在探索步骤中，从数据集中抽样出一批样本，在这些样本上尝试去发现比编译器生成的结果更高效的输出代码，在学习步骤中，从数据集中再独立地抽样出一批样本，其中可能会包含模型生成的优化代码，用这批样本来去更新模型参数。实验表明子模仿学习是更好的微调方法，对于 5.9% 的测试集，其优化效果可以超过 gcc 的 O3 优化。

4、基于大语言模型的程序优化

虽然大语言模型理解和生成代码的能力已经被广为认可[13-14]，不过尝试用大语言模型优化程序的工作还不多。Cummins 和 Grubisic 等人[15-16]借助大语言模型来生成编译器的优化选项。Shypula 等人[17]首次尝试用大语言模型来端对端地优化代码，也就是直接输入给大语言模型一段代码，让它输出优化后的代码；不过，与本研究不同的是，他们优化的是高级语言代码，而不是汇编代码。

## 1.3 实习项目整体执行完成情况概述

1、完成了对超级优化课题的前期调研

2、探索了用遗传算法和岛群算法来优化算法，并实现代码

3、实现了种群+大语言模型的算法，并进行初步测试，在16%的样例上取得了更好的效果

4、在Asmjit工具上编写汇编语言的中间数据结构，使框架完整支持Aarch64指令集

5、将算法迁移至华为基于LLVM的框架上

# 第二章 复杂工程问题归纳与实施方案可行性研究

## 2.1 实习课题的目标与任务

### 2.1.1 实习目标

协作课题组完成与华为合作的超级优化项目，探索实现更高效有用的搜索方案，找到切实可行的汇编等价性验证方案，并实现项目代码与华为接口的迁移对接。

### 2.1.2 任务

1、针对当前技术能力以及设计需求进行可行性评估；

2、完善本地项目框架，增加更多的aarch64指令支持；

3、探索比stoke更高效的搜索算法；

4、将新算法迁移到华为LLVM大框架上；

5、探索汇编等价性验证的方案并实现，完成符号执行与限界模型检测的调研与支持；

6、基于Linux的aarch64环境，选择C++语言进行开发；

7、针对华为内部需要优化的代码，能够进行项目的检测，并对优化结果进行对比分析；

8、完成项目后，对项目进行改进分析，评估对社会，经济的影响。

## 2.2 解决方案及可行性研究

### 2.2.1 实施方案

针对上述任务，实施方案如下：

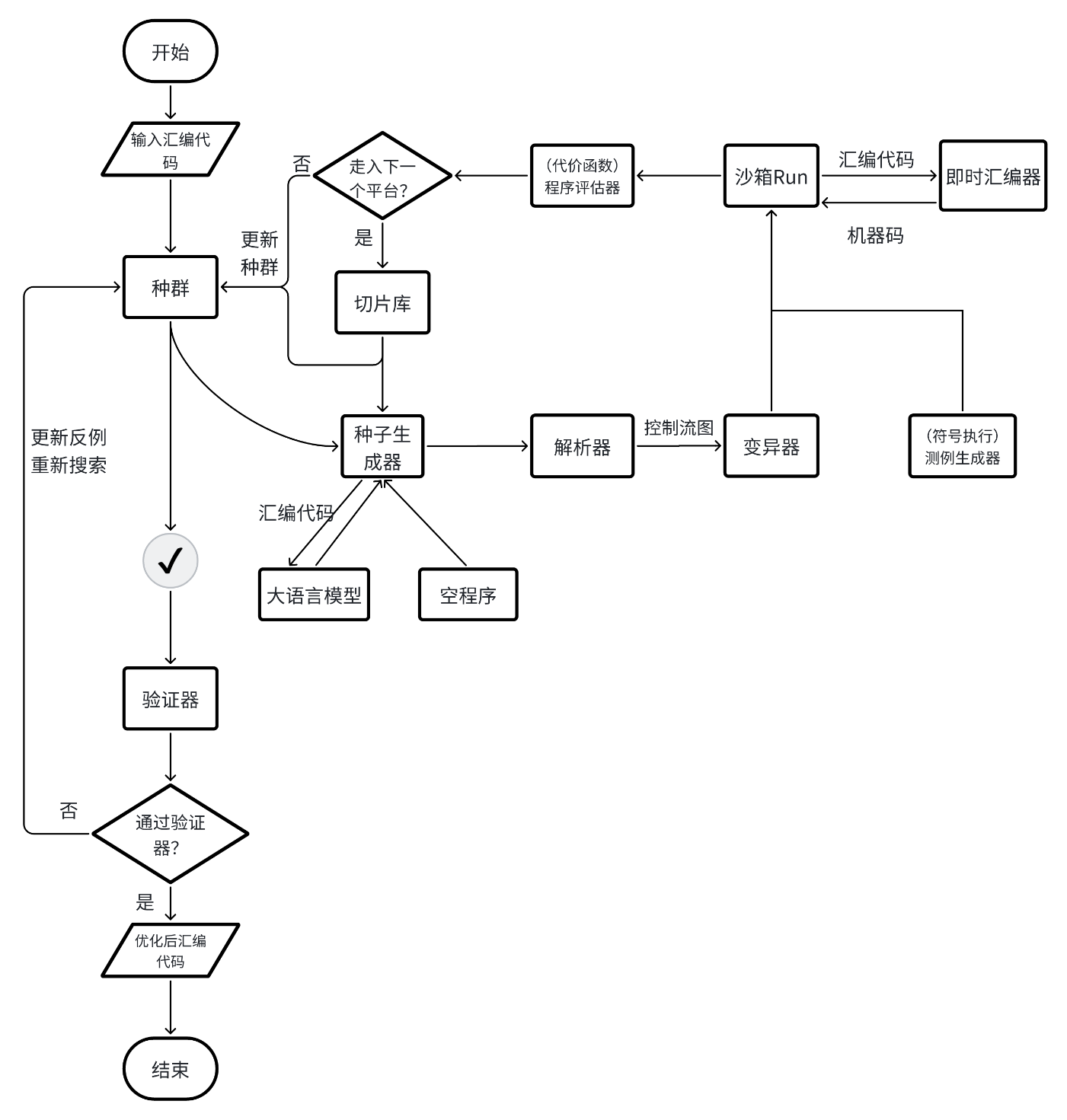


图2-1 应用方案架构图

主体算法部分为二层循环。内层循环是一个进化算法，从由种子构建的初始种群开始，生成新的个体程序，并借助沙箱来评估新程序，根据评估结果决定新程序是否要留存于种群中。沙箱会加载测例，并安全地执行程序，记录程序的输出和运行时间。外层循环负责更新测例集合，对于在内层循环中通过所有测例的程序，验证器会对它进一步地检查，检查其与初始程序在形式语义上的等价性，如果不等价，说明该程序其实是一个错误的优化，验证器会生成能够暴露出该程序的错误的测例，并重启内层的进化算法。

其中测试生成器采用简单的动态符号执行技术，解释器采用Antlr编写，沙箱采用Asmjit工具，验证器采用符号执行+BMC的方案，无特殊之处，故不做详细介绍，具体的种群进化算法设计收到了[18-19]论文的启发，融入大语言模型，并吸收了实习前期的经验教训，详细方案设计如下。

#### 2.2.1.1 种群结构设置

种群数据结构组织主要受到论文[18]的启发。论文[18]认为搜索经常会经历一系列plateus平台期，搜索的过程是不断走向下一个平台的过程，并且合成时间的分布经常是heavy-tailed重尾分布（越在后期，进展越慢）。由于搜索由cost function代价函数引导，如果一个搜索过程陷入低cost但高时间代价的强连通分量，它将陷入重尾分布的后端，长时间无法合成得到结果，于是引入Luby重启算法，多个程序并行搜索，避免因为单个种子合成进度停滞不前导致合成失败，并以一个二叉堆的数据结构进行多程序的维护。

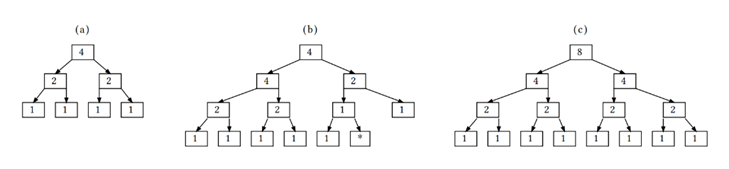


图2-2 多个体重启扩张示意图

首先介绍Luby重启序列⟨1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, ……⟩，意味着在第一次重启时，如果搜索没有成功，我们在第1次迭代后重启；第二次，我们在又进行了1次迭代后重启；第三次，我们在2次迭代后重启；如此类推。Luby序列能够在不知道具体分布D 的情况下提供一个有效的重启策略，每一次重启都是从起点开始。可以将Luby Sequence看作树的深度优先后序遍历，每当到达一棵树的根时，就可以通过复制树并将两个根附加到旧根的两倍标签下来构建下一个树。经典的Luby重启算法实际上是遍历这一系列的树，并对每个节点运行搜索，在搜索结束时停止，否则移动到下一个节点。

[18]讲述的新方法是在换一种方式构造出相同的sequence，即依然遵循深度后序遍历，但不采用复制的方式，而是将所有现有节点的标签加倍，并向每个现有叶子节点添加一对新的l标记的叶子，图2-2(b)是一个snapshot，就是具体描述了深度有序的过程。我们将每个节点与一个搜索状态关联起来，每次我们重复一个节点的标签l时，我们就运行搜索一个额外的l\*t0迭代，这样节点的标签并不一定表示搜索迄今为止运行了多少次迭代，而只表示它未来将分配多少次迭代。与顺序方法不同，我们必须保留部分执行的搜索，也就是比如4->8，新的迭代8次是从4结点结束后的状态开始，而不再是从0开始。

此外，[18]希望利用cost function代价函数，优先运行那些cost更低的结点，每当我们访问非根节点时，如果它的父节点比它的cost更高，就交换这两个结点的搜索状态。节点的标签并不一定表示搜索迄今为止运行了多少次迭代，而只表示它未来将分配多少次迭代。

[18]论文中虽然没有提到种群与进化算法，但已经初具种群的雏形，我将[18]提出的这一数据结构组织方式正式视为一个种群的组织方式，并以此二叉堆为底座进一步实施进化算法，将Stoke的单个体搜索转化为种群的进化。

此外，在维护搜索个体种群的同时，也加入了切片库的设计。切片库维护一定数量的可能对合成有重要帮助的精华程序片段，这些切片将在后续种群算法的交叉与初始化产生作用（详见下节）。对于每一个切片，维护一个成功次数，当切片数量超出最大容量时，优先淘汰成功次数低的切片。

#### 2.2.1.2 种群进化算法

种群进化算法以基本遗传算法为基础，遗传算法（Genetic Algorithm，GA）是一种模拟自然选择和遗传机制的优化算法。它基于达尔文的进化理论，通过模拟生物进化过程中的选择、交叉和变异等操作，来求解复杂的优化问题。

遗传算法的基本原理是模拟自然界中的进化过程，它包括几个重要步骤：

 初始化种群：随机生成一组初始解，称为种群。

 适应度评估：根据问题的定义，对每个个体（解）计算适应度评估值，衡量其在问题空间中的优劣程度。

 选择操作：根据个体的适应度评估值，按照一定的选择策略（如轮盘赌选择、竞争选择等），选择一部分个体作为父代。

 交叉操作：从父代中选取一对个体，通过某种交叉方式（如单点交叉、多点交叉等），生成新的子代个体。

 变异操作：对子代个体进行一定概率的变异操作，引入新的基因信息，增加问题空间的探索能力。

接下来介绍本方案的具体步骤实现：

(1) 初始化：每个个体的种子可以是空程序，切片库进行linear crossover得到的种子或llm得到的种子。其中llm得到种子的方式可以是给定c程序或语义生成，也可以是选择头部三个节点用llm crossover得到（类似[19]，在下一小节详细介绍）

(2) 适应度评估

参考STOKE，即公式TODO

(2) 变异Mutate

参考 STOKE，本方案支持以下几种变异：

• 指令替换：随机挑选一条指令，将其替换为另一条随机生成的新指令；

• 操作数替换：随机挑选一条指令，再随机挑选其中的操作数，替换成同类型

的其他值，比如说将立即数从 2 换成 5，或者是将寄存器从 x0 换成 x1；

• 操作码替换：随机挑选一条指令，将其中的操作码随机替换为另外一个操作

码，这时需要保证当前的操作数对于新的操作码而言也是合法的；

• 全局交换：随机交换程序中的两条指令；

• 局部交换：随机交换一个基本块中的两条指令；

• 指令删除：随机删除程序中的一条指令；

• 指令移动：随机挑选一个基本块，将其中的一条指令移动到另一个位置（某

两条相邻指令中间）。

(3) 交叉Crossover

1) Plateau

借用[18]的数学分析，将搜索过程可视化为经历多个平台期：

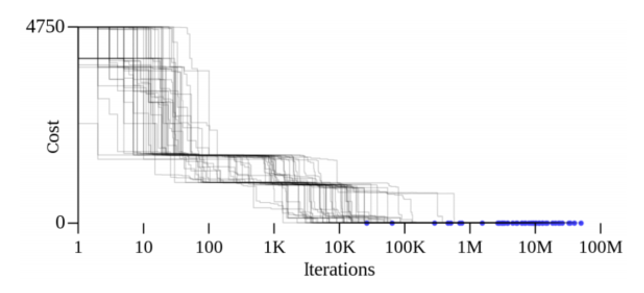


图2-3 搜索过程可视化

首先主动定义了plateau，比如在cost <1 的情况下经过一次变异后cost下降了，可以视作一个plateau；在其他情况下，cost下降比如2则可以视作是走到了下一个plateau。

2) 主要交叉方式

对于使得当前个体能走入下一个plateau的变异，记录下这些变异得到的重要指令，求出它们的前向切片（影响这条指令的其他指令）和后向切片（被这条指令影响的其他指令），将对应切片提取为一个新程序存入切片库，并向父节点传递这些切片（对“好消息”进行传播），具体传递方式采用经典的线性遗传编程的程序交叉。

切片示例如下：

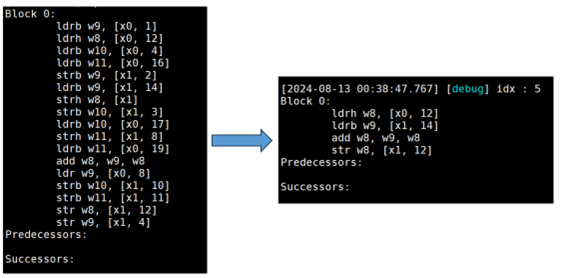


图2-4 切片示例

图2-4左图为搜索过程中的程序，假设找到的重要指令在程序中索引为5（即ldrb w9, [x1, 14]），图2-4右图为这条指令对应的切片，是这条指令所有数据流控制流相关的指令集合。

切片与父节点交叉的方式如下：

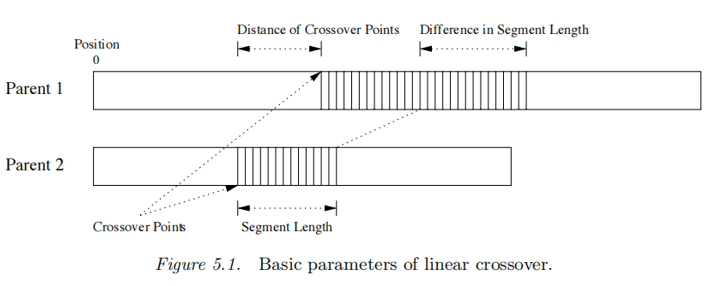


图2-5 lpg crossover示意图

每个切片与父节点交叉次数设置为父节点指令数乘2，在所有交叉得到的程序中取一个cost最小的交叉结果，如果比父节点cost低，则将新程序与父节点程序进行替换，视为“好消息”传递成功。

3) 隐藏交叉

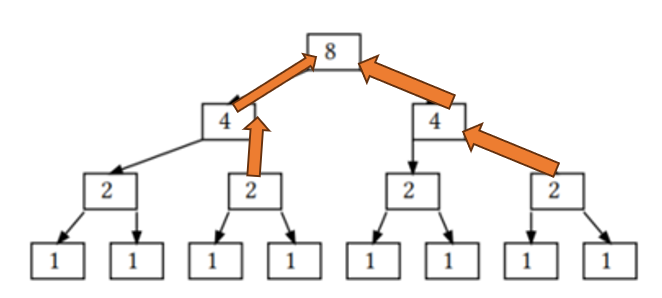


图2-6 切片向父节点传递示意图

对于越往上走的结点，越能吸收不同路径的精华并进行交叉整合（如上图2-6根节点能吸收两条橙色路径的变异精华并加以融合）；并且每次传播只影响当前路径，不影响其他路径，可以继续保持种群多样性（如上图两条橙色路径只向根节点传递，并不影响其他节点）。

(4) 选择更新：

在[18]的基础上限制种群大小，而不是像[18]中个体数可以无限扩大，并给每个节点的变异次数设立阈值，如果一个个体节点迭代次数到达阈值但是仍然离根节点很远（重尾），重置这个节点，以一个新种子替代。

#### 2.2.1.3 大语言模型对话设计

本节会介绍我们是如何与大语言模型对话的，以驱动它优化代码。本研究与大语言模型的交互并非一问一答式的，而是会与大语言模型做多轮对话。大语言模型处理多轮对话的原理是，在每次生成新的答复时，都会完整回顾之前的全部对话，包括所接收的输入和自己生成的回答，利用这一特性，我们可以在对话中引导大语言模型思考，逐步地给它提供信息，并让它从自身的错误中积累经验。

对话的设计目标包括：

1. 让大语言模型尽可能地去优化代码；

2. 生成代码需要尽量保证正确。

首先聚焦于第二个目标，我们利用汇编器（assembler）和链接器（linker）——来初步检查并帮助大语言模型矫正所生成的代码。所有大语言模型生成的种子都必须经历如下流程：

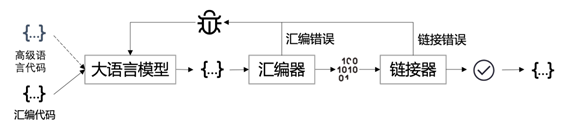


图2-7 对话构建：借助编译器来帮助大语言模型初步矫正所生成的程序

矫正的方式是把汇编器或链接器的报错再反馈给大语言模型，让其修复错误，直至通过编译。由于编译器报错信息的设计初衷便是为了让开发者容易理解错误及修复错误，会结合自然语言的描述和具体的代码错误，所以编译器的报错信息是适合反馈给有很强自然语言理解能力的大语言模型的。

对于第一个目标，我们设计了两种大语言模型的使用方式：

1、输入目标C程序和目标汇编代码以及大致语义的自然语言描述，由大语言模型得到一个初始化种子。

2、仿照[19]的方式，输入多个搜索过程中的程序和它们对应的cost代价，让大语言模型进行crossover得到一个新种子。

方式一的使用方式为字面意思，不做过多介绍。方式二受到[19]的启发，每次选择种群中堆顶的三个程序和它们的代价作为输入（并根据代价大小排序），并辅以输入寄存器等信息组成prompt，其中一个prompt示例如下：

Now, I will give you an assembly program and tell you the meaning of this program.

the target :{

eor x4, x3, x1

cmp x0, x3

csel x4, x4, xzr, eq

eor x2, x3, x2

cmp x0, x1

csel x2, x2, xzr, eq

eor x0, x4, x2

eor x0, x0, x3

}

The primary purpose of the function is to Cycling through 3 values.

I want to optimize this assembly , please optimize it according to the goal that summarized above.and now I have some priority versions: (Please notice that the lower the cost, the better the effect)

Hint: Algorithmic-level optimizations or specialized instructions may help a lot.

priority\_v0: {eor x0, x2, x1, 0

orr x0, x3, 32

}

cost : 25.093750

priority\_v1: {and x2, x0, -16

lsr x0, x1, x2

lsr x2, x3, 8

}

cost : 24.343750

priority\_v2: {subs x3, x0, 5

csel x0, x0, x2, eq

}

cost : 22.593750

priority\_v3: {csel x0, x2, x3, eq

cinc x0, x1, ne

lsr x2, x0, 6

cinc x0, x1, ne

}

cost : 16.718750

Complete your code here : improved version : priority\_v4: {

}

they are all incomplete. Can you apply your understanding of this assembly, Integrating the advantages of these assemblies (understanding why these two assemblies have a certain effect, rather than simply combining them) to generate a correct program ?(Just like making a crossover in genetic algorithms) .Approaching the ultimate goal is enough, not necessarily all of it must be correct. but do not return to the original target program(important !)

You can only use the registers in def-in (please !), and the final criterion for determining correctness is as long as the value of the live-out register is equal to the result of the original assembly program running. Here are the registers of def-in and live-out:

'def-in = ["x1","x2","x3", "x0"]'

'available registers = ["x3","x2","x1","x0",]'

'live-out = [x0",]'

You only need to return the specific assembly content, without adding. syntax unified, . thumb .global or other pseudo-ops

Imitate the format of the assembly code we provided. Do not write comments within the assembly

我们注意到携带一个反例，更有利于大语言模型理解优化，此处由于篇幅受限，不详细展示。

### 2.2.2 可行性研究

1. 技术可行性研究：目前学术界最先进的超级优化工具是STOKE，但由于它基于x86-64指令集，而本项目面向AArch64指令集，所以我们无法直接使用STOKE以及与STOKE在优化效果上作对比，这里我们会从使用的技术方法上阐述我们所选取的技术路线相比于STOKE的技术路线的改进。

我们做了两方面的改进：首先将随机化搜索算法由 Metropolis-Hastings 算法改进为进化算法，带来了搜索效率和搜索能力的提升。之所以会提升搜索效率，是因为进化算法更新迭代的是种群而不是单个个体，允许同时探索程序空间的多个区域，这就增加了算法的并行性；之所以会提升搜索能力，是因为其种群的多样性会帮助算法避免陷入局部最优解，特别是对于程序空间这种极其不规律的离散空间。

并且我们借助大语言模型来理解和生成程序，从而在生成程序时将现有程序的语义充分纳入考虑。自 2023 年初以来，大语言模型展现出了卓越的对（程序）语言的理解和生成能力，并且其也表现出了对基本算法和指令集知识的掌握，这提示我们可以用它来对汇编代码作初始优化，或者是借助大语言模型去融合两个程序的语义，而不是仅仅在语法层面将两个程序交叉。

2. 经济可行性研究：本方案与本地大模型的交互过程无需额外财力投入，因为它依赖于已有的本地资源。此外，其他代码部分的开发可利用免费的开源工具，从而避免了财务方面的开支。因此，从经济角度考虑，这个方案具有可行性。

3. 法律法规可行性研究：我们的方案通过代码优化的方式实现，不涉及任何违反法律法规的行为。我们遵守相关的法律要求，并确保该方案的实施不会影响法律法规的合规性。因此，从法律法规的角度考虑，这个方案是可行的。

## 2.3 实习课题任务相关的详细需求分析

1、功能需求

(1)、搜索算法设计

设计比stoke更高效的搜索算法，以提高程序优化的效率和精度。负责设计出一个比stoke更高效的搜索算法，负责让框架支持全面的Aarch64指令集。希望实现后的系统能优化华为网络部门的热点程序，使整体性能至少提升3%，最好提升5%。

实现针对汇编语言的特定优化策略，使得搜索结果更加针对性和有效。

(2)、Aarch64指令集支持

扩展框架的指令集支持，确保能够涵盖全面的Aarch64指令集。优化搜索器以适应Aarch64指令集的特性，从而更好地发现最佳程序版本。

2、性能需求

(1)、搜索时间限制

要求在4小时内对大约50行左右的Aarch64汇编代码进行搜索优化。优化搜索算法和架构，确保在规定的时间内完成对优化程序的搜索。

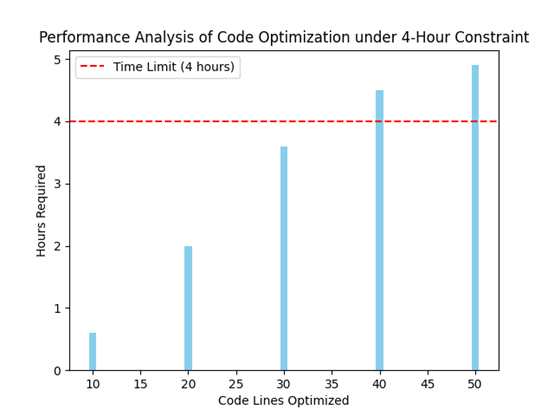


图2-8 搜索时间需求图

目前40行以上的程序在4小时内搜索出来较为困难，我们需要实现一个更优的搜索算法，使得搜索35行以上的程序不超时。

(2)、优化精度和性能提升

优化沙箱与验证器的性能，确保评估程序正确性和形式化验证的速度与精度。

## 2.4 相关复杂工程问题总结

1、需要将超级优化子模块融合进华为的LLVM大框架中。需要进行架构设计、模块接口定义和算法优化，以确保子模块与LLVM的无缝协作。

2、将搜索模块、验证模块结合，并与大语言模型高效交互。搜索模块需要实时得到验证模块的反例反馈。同时，需要设计合适的接口和通信机制，以便在超级优化子模块和大型语言模型之间进行高效的数据交换和协作。解决这个问题需要综合考虑多个因素，并进行深入的研究和实践。

# 第三章 针对复杂工程问题的方案设计与实现

## 3.1 硬件的设计与实现

此课题项目不涉及到硬件。

## 3.2 工具软件的设计及实现

1、Z3 是一个高性能的定理证明器，微软研究院开发的SMT求解器。它是一个高性能的约束求解器，能够解决布尔逻辑、整数、实数、位向量等多种数学理论的可满足性问题，可使用它编写自动化测试和验证工具。

2、Antlr是一个强大的解析器生成器，用于构建词法分析器和语法解析器，可以解析特定语言或汇编语言。

3、Asmjit是一个跨平台的汇编代码生成框架，旨在提供一个高性能、轻量级的汇编库，用于运行时生成即时编译（Just-In-Time Compilation，JIT）的代码。Asmjit 被用作沙箱，辅助在搜索阶段计算rewrite程序的cost和performance cost。

4、Argparse：命令行参数的解析由 argparse 库处理，它可以方便地定义和解析命令行参数。

5、Tomlplusplus：配置文件的解析则采用 tomlplusplus，这是一个专门用于处理TOML（Tom’s Obvious Minimal Language）语言撰写的配置文件的库，使得用户能够以简洁、易于理解的格式定义配置。

6、Spdlog：日志记录功能通过 spdlog 实现，spdlog 是一个快速的日志记录库，支持多种输出目标和格式，会方便开发和调试。

7、Nlohmann：处理 JSON（JavaScript Object Notation）的任务交给了 nlohmann JSON 库，它可以将 C++ 中的数据结构序列化成 JSON 数据，或者是将 JSON数据反序列化成 C++ 中的数据结构，这样就可以通过 REST API（Represen-tational State Transfer Application Programming Interface）与大语言模型交互。

## 3.3 应用软件的设计与实现

### 3.3.1 应用软件架构图

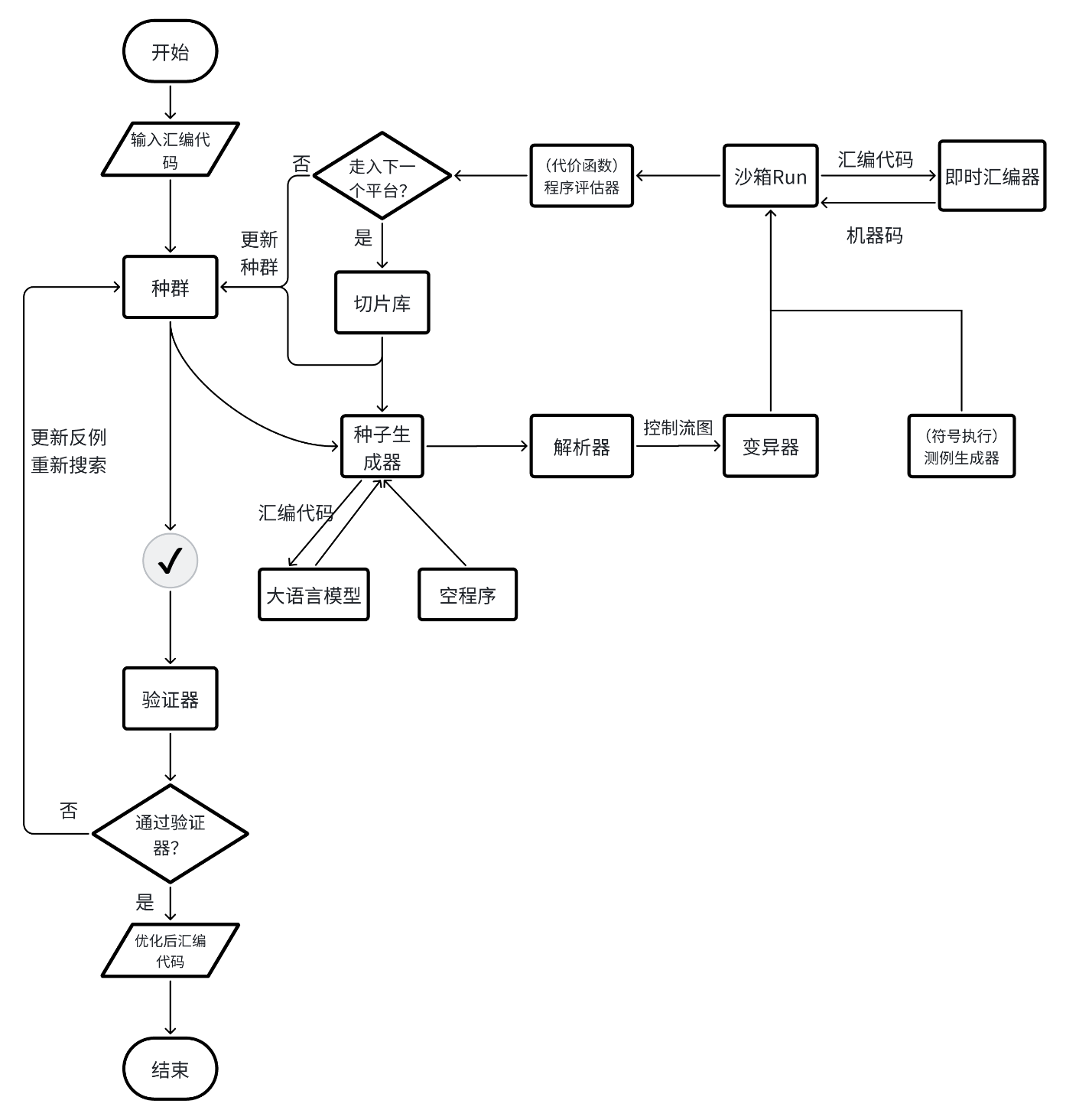


图3-1 应用方案架构图

### 3.3.2 子系统和功能点流程图

项目整体模块示意图如下：

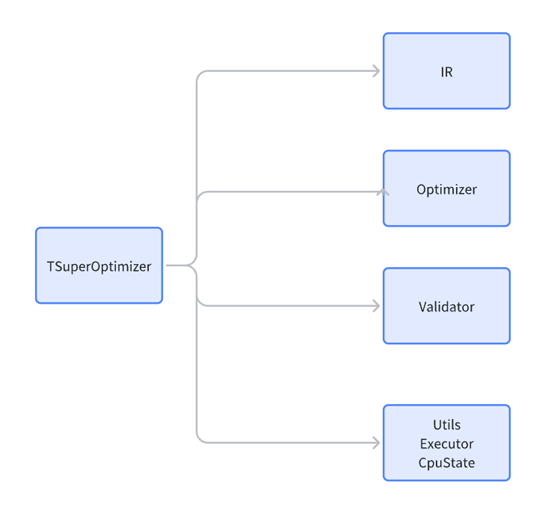


图3-2 项目模块示意图

1、IR部分：记录了用于存储汇编指令的相关ir结构

(1) instruction.hpp和instruction.cpp：一条指令由class TInstNodeHandler表示，主要由InstNode和InstType组成。其中指令本身由asmjit的数据结构InstNode表示，指令的每个操作数的类型由InstType表示。

(2) program.hpp和program.cpp：class Program代表一个由一个或多个BasicBlock组成的汇编程序（类似于一个CFG），其中一个基本块由std::vector<TInstNodeHandler> insts;组成。

(3) parser.hpp和parser.cpp：从文件读入一个汇编，解析并返回一个Program

(4) type.hpp和type.cpp：将操作数类型分为MemType、NumType、RegisterType等等，并根据不同的属性分出更多子类（比如寄存器分出了64位、32位、向量寄存器几个子类）。一条指令的所有操作数的type放在InstType类型中。有一些指令（如bfc、casp）的操作数范围无法在静态时定下来，又派生出BfInstType等子类。通过几个InstType中可重写的方法（如下）可随机生成一个操作数，这些在mutate的时候被调用：

virtual std::tuple<bool, asmjit::Operand,RegisterWidth> generate\_gp\_reg(int i, RegisterWidth rw, std::shared\_ptr<asmjit::InstNode> inst);

virtual std::tuple<bool, asmjit::Operand> generate\_vec\_reg(int i, RegisterWidth &rw, ReinterpretWidth &re, std::shared\_ptr<asmjit::InstNode> inst);

virtual asmjit::Operand generate\_shift(int i);

virtual asmjit::Operand generate\_cond(int i);

virtual asmjit::Operand generate\_imm(int i, std::shared\_ptr<asmjit::InstNode> inst);

2、Optimizer部分

(1) mutator.cpp和mutator.hpp：调用mutate可进行一次变异：

std::pair<bool, Program> mutate(Program program,

bool preserving\_semantics = false);

(2) optimizer.hpp和optimizer.cpp：管理所有搜索优化器，运行过程中选择一种配置文件指定的优化器override的search()方法搜索运行。

目前的搜索优化器：

(1) stoke.cpp：同stoke工具的MCMC随机搜索

(2) cryptopt.cpp：语义等价的mutate，同PLDI 23，对一个基本块内的指令进行 reorder

(3) restartopt.cpp：在stoke的基础上实现了PLDI 21'《adaptive restarts for stochastic synthesis》的重启restart方法

(4) funsearch.cpp：完全按照FunSearch论文实现的一版

(5) new\_genetic.cpp：在funsearch的基础上增加了交叉变异，但是目前没有加对整个种群的选择更新操作，所以收敛速度较慢

(6) genetic.cpp：基本的遗传算法

3、Validator验证器部分

调用validator.hpp的函数可检验两个program是否等价：

std::pair<z3::check\_result, std::optional<CPUState>>

check\_equivalence(Program target, Program rewrite);

4、其他文件

(1) config.cpp和config.hpp：保存从.toml文件中读到的各个信息

(2) cpustate.cpp和cpustate.hpp：用`class HeapSegment`和class CPUState拟出cpu寄存器、堆的状态。并有CPUState gen\_rand7();等可指令testcase的生成方法

(3) executor.cpp和executor.hpp：通过execute可运行一个program并得到一个程序的cost值：

pair<double, double> execute(Program &program,

const vector<CPUState> &testcases,

const vector<CPUState> &target\_results);

### 3.3.3 应用软件的运行流程及效果

以简单的p21.s为例，展现软件的运行流程及完成后的界面：

eor x4, x3, x1

cmp x0, x3

csel x4, x4, xzr, eq

eor x2, x3, x2

cmp x0, x1

csel x2, x2, xzr, eq

eor x0, x4, x2

eor x0, x0, x3

这段代码的目的是循环遍历三个值，并比较交换将需要的值存入x0中。

1、搜索阶段：

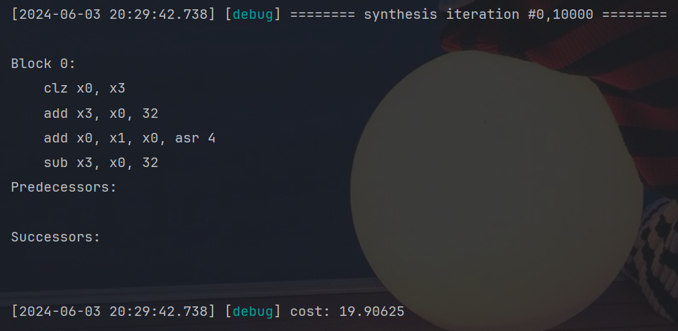


图3-3 搜索阶段截图

2、搜索器搜到一个通过所有测例的程序：

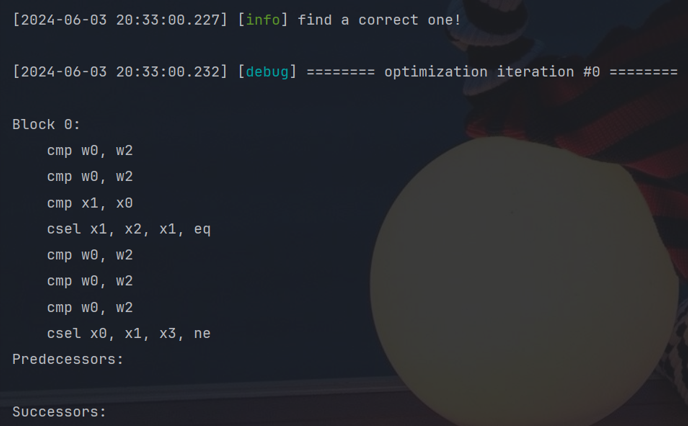


图3-4 搜索到正确程序的截图

并不断对这段程序进行进一步搜索优化：

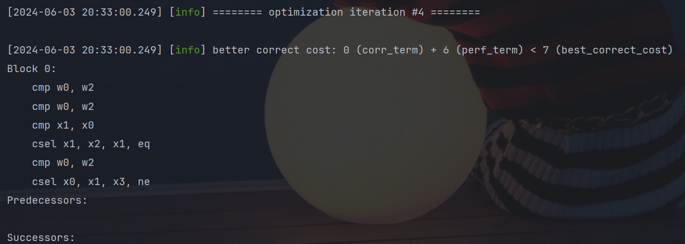


图3-5 搜索优化程序截图

3、验证器进一步对rewrite程序的等价性进行验证，跑出一个反例或得到一个通过验证的结果：

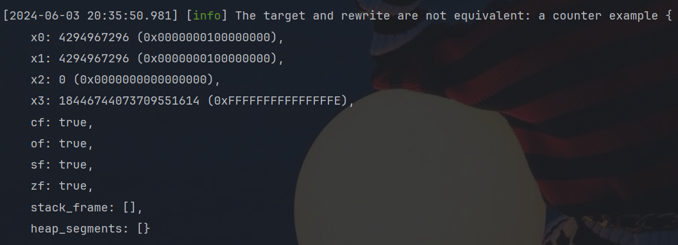


图3-6 验证器返回一个反例

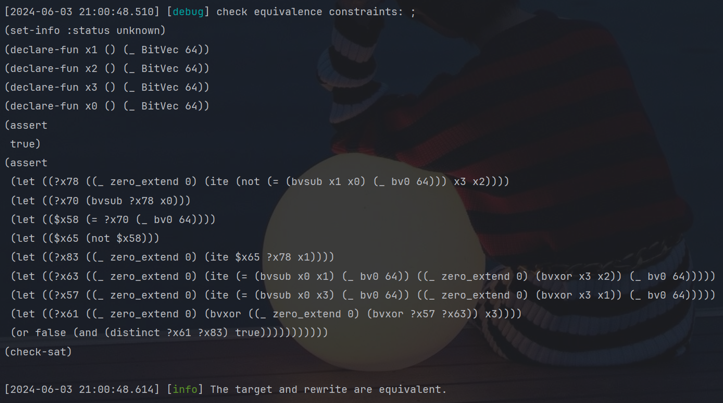


图3-7 验证器验证出等价

4、得到最终优化后程序：

cmp x1, x0

csel x2, x3, x2, ne

cmp x2, x0

csel x0, x2, x1, ne

## 3.4 应用软件测试

我们实现了所提出的系统方案，并与既有方案实施了初步的对比实验，以检验所提改进的效果。

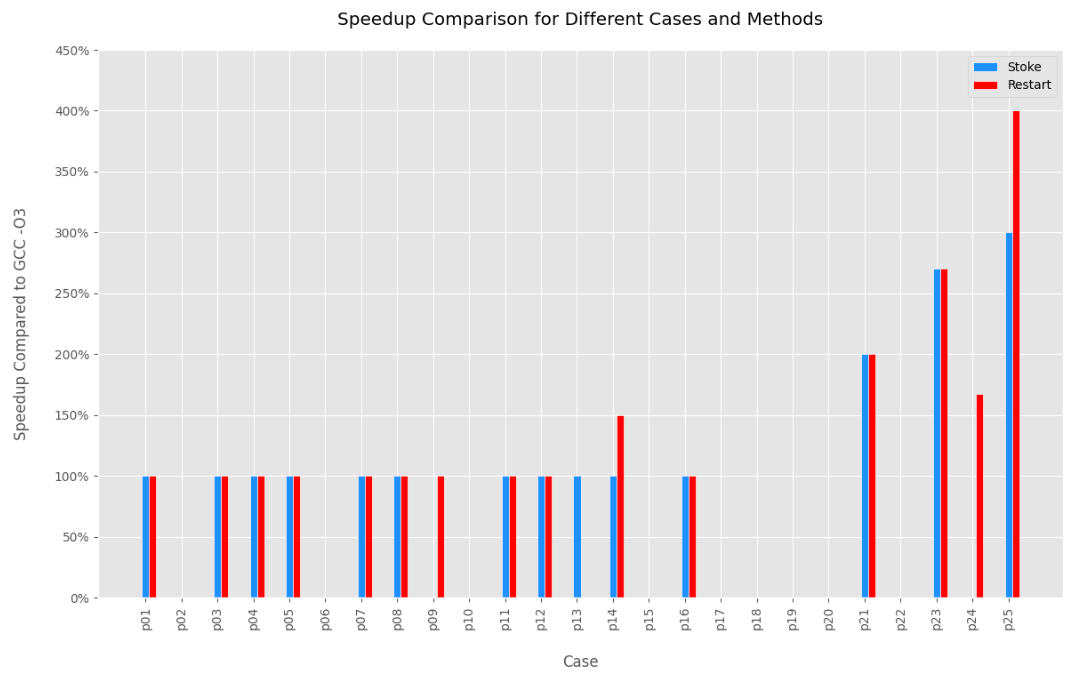


图3-8 初步对比测试结果

我们会对比现有的基于个体的随机化搜索方法（Stoke）和基于种群的进化算法，这里我们实现了我们的进化算法（Restart）。对于每个待优化的程序，我们会给每个算法40 分钟的时间，去在程序空间中探索。随机化搜索方法（Stoke）从空程序出发，进化算法从算法生成的种子出发。

评估所使用的目标程序集合是来自教科书《高效程序的奥秘（Hacker's Delight）》中的25个程序，这是一个检验超级优化工具的经典基准数据集，最初由Gulwani所收集。这 25 个程序均为位运算和算术指令序列，不包含分支结构，比如“将字中最右边的 1 置 0”和“计算两个 32 位无符号数乘积的高 32 位”。我们所使用的版本来自STOKE的实验，我们首先使用GCC -O1从C代码编译出AArch64汇编代码，将这 25 份汇编代码作为我们的目标程序集。

实验结果如下图所示，其中横轴为25 个目标程序，纵轴为相对于工业级编译器的最高级别优化（GCC -O3）的加速比率，这里我们用指令条数来估计程序的运行时间。

总得来讲，在 4/25=16% 的目标程序（p09,p14,p24,p25）上，进化算法表现出了比现有的随机化搜索算法更好的效果；在 1/25=4% 的目标程序（p13）上，随机化搜索算法比进化算法表现了更好的效果。其中p09与p24分别是求绝对值和计算离输入数最近的2次幂数，大语言模型都提供了较符合语义的种子，加速了合成的进行。

在华为的benchmark上评估是否达到需求，目前还正在进行，暂不能提供更详细的测试数据。

# 第四章 知识技能学习情况

## 4.1 开发环境和工具

本实验的实验环境分为两部分：用于运行本工具的环境，由于需要直接执行AArch64 汇编代码，所以需要一个 ARM64 架构的设备；用于运行大语言模型的环境，由于需要大量的计算资源，所以需要一个 GPU。两者之间通过 REST API（representational state transfer application programming interface）来通信。

运行本工具的环境如下：

• 操作系统：Ubuntu 22.04

• CPU 型号：ARM Cortex-A76（时钟频率为 2GHz）

• 内存大小：1.7GiB

• 编程语言：C++17

• 集成开发环境（IDE）: Visual Studio Code (VSCode)

本实验所使用的大语言模型为 Mixtral-7B，运行大语言模型的环境如下：

• 操作系统：Ubuntu 20.04

• GPU 型号：NVIDIA GeForce RTX 4090

• 显存大小：24 GiB

## 4.2 预备知识

1、C++语法：C++是一种通用的高级编程语言，用于开发各种类型的应用程序。它提供了丰富的语法和功能，包括面向对象编程、泛型编程和低级编程支持。熟悉C++语法意味着你能够理解和编写C++代码，包括变量和数据类型、控制流语句、函数和类等。掌握C++语法是进行高效编程的基础，更有利于构建模块化、可扩展和可维护的软件系统。

2、Aarch64汇编：Aarch64是ARM架构的64位汇编语言，广泛应用于移动设备和嵌入式系统。它是一种低级编程语言，用于直接控制计算机硬件。需要熟悉寄存器的使用和约束、内存访问指令、分支和跳转指令，以及操作数和指令格式。Aarch64架构提供了一组通用寄存器，用于存储数据和执行计算；aarch64提供了多种指令用于读取和写入内存中的数据。这些指令包括加载（load）和存储（store）指令，用于从内存中加载数据到寄存器或将数据存储到内存中，需要理解这些指令并了解内存对齐和数据访问模式。此外，分支、跳转指令、向量寄存器等也是需要了解的内容。

3、编译原理与LLVM框架：编译原理研究如何将高级编程语言转换为机器语言，它涉及词法分析、语法分析、语义分析、优化和代码生成等方面的技术。LLVM（Low-Level Virtual Machine）是一个广泛使用的编译器基础设施，它提供了一套用于构建编译器和工具的开发框架。LLVM的设计目标是提供灵活、可扩展且高性能的编译器技术，适用于各种不同的编程语言和目标架构。

LLVM采用了一种中间表示语言称为LLVM IR（LLVM Intermediate Representation）。LLVM IR是一种低级的、静态单赋值形式（Static Single Assignment，SSA）的抽象语言，它在编译过程中作为中间表达式的表示形式。LLVM IR是一种面向对象的表示形式，通过使用类型、操作符和指令，描述了程序的数据流和控制流。

LLVM IR具有许多优点。首先，它是与机器无关的，因此可以在不同的目标架构上进行优化和代码生成。这意味着你可以使用相同的LLVM IR表示形式来编写针对不同平台的代码，并利用LLVM的优化和代码生成功能来生成高效的目标代码。

其次，LLVM IR提供了丰富的优化机会。LLVM框架内置了多种优化技术，包括常量折叠、死代码消除、循环优化、内联函数等。通过在LLVM IR层面上应用这些优化，可以在编译过程中显著改善代码的性能和效率。

## 4.3 新知识点学习和掌握情况

在实习过程中，根据课题应用场景，选择并学习了多种开发环境、工具和技术标准。具体情况如下：

1、符号执行（Symbolic Execution）是一种静态分析技术，用于推导程序在所有可能输入情况下的行为。它是一种自动化的程序分析方法，通过以符号形式代替具体的输入值来执行程序，并跟踪程序中的符号约束，以推导出程序的各种路径、条件和状态。在符号执行中，程序的输入被表示为符号值，而不是具体的数值。这些符号值可以代表变量、表达式或内存位置等。通过符号执行引擎，程序可以在不实际运行的情况下，通过处理符号值和符号约束来探索不同的路径和可能的行为。

符号执行的主要目标是发现程序中的漏洞、错误、不变量和不一致性。通过执行所有可能的输入路径，符号执行可以发现潜在的错误情况，如空指针解引用、整数溢出、数组越界、死锁等，并生成相应的测试用例或错误报告。具体而言，维护两个公式：

 Path condition，路径条件——执行到当前分支(路径）需要满足的条件

 Symbolic Memory，符号内存——变量当前的值

在执行到如assert(x-y != 0)时调用smt solver，返回验证结果。

2、BMC：符号执行的挑战之一是路径爆炸问题，多层嵌套if语句、无上界循环都可能给程序路径数目带来指数级、甚至是无界地增长。引用著名的停机问题理论，想要判定无上界循环的终止性是不可判定的，即不存在一个行之有效的算法能够对无上界循环进行推理。因此，为了有效解决路径爆炸问题带来的挑战，采用限界模型检测技术将无界模型转化为有界模型，以在有限时间内高效地进行等价性验证。限界模型检测的基本思想是对程序的执行进行建模，直到达到有限步数为止。从技术上而言，这个过程基本上可以等效于循环展开的过程。循环展开的概念可用以下通用例子来说明：

int main(int argc, char \*\*argv) {

while(cond) {

BODY CODE

}

}

一个包含最多五次迭代循环的限界模型检测实例为上述源代码包含五个循环体的副本，并基本上等价于对以下无循环程序进行检查：

int main(int argc, char \*\*argv) {

if(cond) {

BODY CODE COPY 1

if(cond) {

BODY CODE COPY 2

if(cond) {

BODY CODE COPY 3

if(cond) {

BODY CODE COPY 4

if(cond) {

BODY CODE COPY 5

}

}

}

}

}

}

上述代码基于if语句来对循环进行展开，使得循环在前五次迭代前终止。进行循环展开后，产生了在限界内程序执行轨迹和原程序等价的新程序。采用限界模型检测后，无界的原程序被转化为有界的新程序进行验证，让有限资源下对于无界程序的符号执行成为可能。

# 第五章 工程协作交流情况

工程协作交流情况表详见表“5-1”。

表5-1 工程协作交流表

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 协作人 | 协作工作 | 完成情况 |
| 1 | 谢兴宇（实验室同学） | 继续研究搜索算法，对框架进一步进行支持 | 搜索算法最后基本由我一人完成 |
| 2 | 闫润邦（实验室同学） | 完成向量指令支持 | 完美完成 |
| 3 | 梁航宇（实验室同学） | 沟通搜索端与验证端的benchmark | 完美完成 |
| 4 | 华为工作人员 | 使用华为提供的沙箱和接口迁移算法 | 延期1个月完成 |

# 第六章 工程计划管控与执行情况

工程计划管控与执行情况详见表“6-1”。

表6-1 课题实施计划表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 工作内容 | 计划工作开始时间 | 计划工作结束时间 | 实际工作开始时间 | 实际工作结束时间 |
| 1 | 对超级优化项目进行基础调研 | 1.10 | 1.20 | 1.10 | 1.20 |
| 2 | 调研遗传算法并得到初步demo | 1.21 | 2.18 | 1.21 | 2.20 |
| 3 | 调研FunSearch并得到初步demo | 2.21 | 3.15 | 2.28 | 3.30 |
| 4 | 学习PLDI 21并产生新想法 | 4.5 | 4.20 | 4.5 | 4.20 |
| 5 | 探索cost函数改进方法 | 4.21 | 4.30 | 4.21 | 4.30 |
| 6 | 为框架补全支持的指令集 | 5.1 | 5.25 | 5.1 | 5.30 |
| 7 | 将新算法迁移至华为内部 | 6.1 | 6.30 | 6.1 | 8.1 |
| 8 | 根据华为数据集调整算法 | 7.1 | 7.15 | 8.2 | 8.23 |

# 第七章 职业素养与工程伦理的学习与培养

作为一名软件工程专业的学生,我深刻认识到职业素养和工程伦理在我的学习和未来职业发展中的重要性。在完成超级优化项目中,我尤其注重以下几个方面:

1、 系统质量意识:

(1) 我了解到软件系统质量的关键指标,并在项目实践中努力提升系统的功能性、可靠性、可用性等方面的表现。

(2) 我建立了完善的测试体系,通过系统验证确保所开发的软件能够满足用户需求。

(3) 我主动参与代码审查,并利用静态分析工具持续改进代码质量,减少系统缺陷。

2、 服务意识:

(1) 我时刻以用户需求为中心,努力提供优质的软件服务,确保用户体验良好。

(2) 我积极倾听用户反馈,及时响应并解决用户问题,不断优化系统功能和性能。

(3) 我致力于与用户建立良好的合作关系,为用户创造价值,提高企业的社会信誉。

3、 职业道德和规范:

(1) 我遵守国家相关法律法规,诚实守信,维护企业利益和社会公共利益。

(2) 我尊重知识产权,杜绝抄袭剽窃行为,保护自身及他人的创作成果。

(3) 我保护用户隐私和数据安全,不泄露或非法使用用户信息。

通过在编译超级优化项目中的实践,我更加深入地理解和认识到软件工程的职业素养和工程伦理的重要性。我将继续努力学习,在今后的工作中践行这些专业标准,为用户和社会创造更大的价值。

# 第八章 对软件工程实践以及软件工程领域发展的认识

在完成这次编译优化项目的过程中,我对软件工程实践以及软件工程领域的发展有了更深入的认识。

1、 软件工程实践对社会可持续发展的影响：软件系统的优化能够提升计算资源的利用效率,为社会各领域的可持续发展提供有力支撑。例如编译优化可以让软件在有限的硬件条件下发挥更大作用，减少了对新硬件的需求,从而降低了社会资源的消耗。

2、 软件工程实践促进创新：软件工程实践的不断发展,推动了计算机科学技术的进步。编译优化等技术的进步，为更复杂、更先进的软件系统的开发提供了可能，引发了一系列的技术创新。

3、 软件工程实践推动社会发展：高性能、低能耗的软件系统,为社会各个领域的发展注入了新动力。编译优化项目的成果,将为未来社会的可持续发展贡献自己的一份力量。

总的来说,软件工程实践对社会可持续发展具有重要影响。作为一名学生,我将继续深入学习软件工程相关知识，为构建更加高效鲁棒的软件系统贡献自己的力量。

# 第九章 结束语

## 9.1 课题完成情况、有待进一步解决的问题及方向

1、框架的沙箱对于栈和data段的实现还未完成

2、算法的测试还不充分，还有很多内存测试用例需要测试

3、在华为的工作还没有结束，还需要在工业界的benchmark上继续加强算法。

## 9.2 本人对于企业实习的收获及体会

学会了很多写大型项目的技巧，也感受了科研项目的流程，明白了科研的不易，明白了从问题调研到编写算法的全过程。但是实在是太累了，每天早9晚11，以后要学会自我调节才行。

# 参考文献

[1] H. Massalin, “Superoptimizer: a look at the smallest program,” ACM SIGARCH

Comput. Archit. News, vol. 15, no. 5, pp. 122–126, Nov. 1987, doi: 10.1145/36177.36194.

[2] R. Joshi, G. Nelson, and K. H. Randall. Denali: A goal-directed superoptimizer. In PLDI, pages 304–314, 2002.

[3] Raimondas Sasnauskas, Yang Chen, Peter Collingbourne, Jeroen Ketema, Jubi Taneja, and John Regehr. 2017. Souper: A Synthesizing Superoptimizer. arXiv preprint arXiv:1711.04422 (2017).

[4] Cabrera Arteaga, J., Donde, S., Gu, J., Floros, O., Satabin, L., Baudry, B., Monperrus, M.: Superoptimization of webassembly bytecode. In: Companion Proceedings of the 4th International Conference on Art, Science, and Engineering of Programming. pp. 36–40 (2020)

[5] T. Granlund and R. Kenner. Eliminating branches using a superoptimizer and the gnu c compiler. In PLDI, 1992.

[6] P. M. Phothilimthana, A. Thakur, R. Bodik, and D. Dhurjati. Scaling up superoptimization. In Proceedings of the TwentyFirst International Conference on Architectural Support for Programming Languages and Operating Systems, pages 297–310. ACM, 2016.

[7] Berkeley Churchill, Oded Padon, Rahul Sharma, and Alex Aiken. 2019. Semantic Program Alignment for Equivalence Checking. In PLDI. <https://doi.org/10.1145/3314221.3314596>

[8] Sahar Badihi, Faridah Akinotcho, Yi Li, and Julia Rubin. 2020. ARDiff: Scaling program equivalence checking via iterative abstraction and refinement of common code. In Proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 13–24

[9] Shubhani Gupta, Abhishek Rose, and Sorav Bansal. 2020. Counterexample-Guided Correlation Algorithm for Translation Validation. In OOPSLA. https://doi.org/10.1145/3428289

[10] Nagasamudram R, Naumann D A. Alignment Completeness for Relational Hoare Logics

[M/OL]. arXiv, 2021[2022-11-23]. http://arxiv.org/abs/2101.11730.

[11] Shypula A, Yin P, Lacomis J, et al. Learning to Superoptimize Real-world Programs:

arXiv:2109.13498[R/OL]. arXiv, 2022[2022-05-30]. http://arxiv.org/abs/2109.13498.

[12] Williams R J. Simple statistical gradient-following algorithms for connectionist reinforcement

learning[Z].

[13] OpenAI, Achiam J, Adler S, et al. GPT-4 Technical Report[M/OL]. arXiv, 2024[2024-03-20].

<http://arxiv.org/abs/2303.08774>.

[14] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374 (2021).

[15] Cummins C, Seeker V, Grubisic D, et al. Large Language Models for Compiler Optimization

[M/OL]. arXiv, 2023[2023-09-26]. http://arxiv.org/abs/2309.07062.

[16] Grubisic D, Cummins C, Seeker V, et al. Compiler generated feedback for large language models

[A]. 2024.

[17] Alexander Shypula, Aman Madaan, Yimeng Zeng, Uri Alon, Jacob Gardner, Milad Hashemi, Graham Neubig, Parthasarathy Ranganathan, Osbert Bastani, and Amir Yazdanbakhsh. 2023. Learning performance-improving code edits. arXiv preprint arXiv:2302.07867 (2023).

[18] Jason R. Koenig, Oded Padon, and Alex Aiken. 2021. Adaptive restarts

for stochastic synthesis. In PLDI ’21: 42nd ACM SIGPLAN International

Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021, Stephen N. Freund and Eran Yahav

(Eds.). ACM, 696ś709. https://doi.org/10.1145/3453483.3454071

[19] B. Romera-Paredes, M. Barekatain, A. Novikov, M. Balog, M. P.

Kumar, E. Dupont, F. J. Ruiz, J. S. Ellenberg, P. Wang, O. Fawzi et al.,

“Mathematical discoveries from program search with large language

models,” Nature, pp. 1–3, 2023.

# 致谢

感谢实验室的同学们与带我的老师，教会了我许多编程和基础的知识，教会了我很多大型项目开发的技巧。也感谢大家对我的照顾，在女生节全实验室送了我一个Casual的牛牛蛋糕，在我生病的时候老师给我送来了感冒药，也感谢华为的同事陪我一起加班，实习这半年过得很辛苦但是也幸福！