大规模代码遗产系统的重构与优化——软件工程难题与挑战

熊英飞（北京大学）
刘辉（北京理工大学）
李宣东（南京大学）

背景：软件维护

- 软件维护成本占软件开发的90%
- 大规模遗产系统的维护尤其困难
  - 文档缺失
  - 结构复杂
  - 技术债高
- 美国社会保障管理局的信息管理系统
  - 存在6000万行用COBOL编写的代码
  - 重构预计需投入近7亿美元和5年时间

全自动软件维护

- 能否自动完成软件维护?
  - 大模型出现之后，自动化的希望大大增加
- 软件维护包括
  - 功能性维护
  - 非功能性

全自动软件维护

- 能否自动完成软件维护?
  - 大模型出现之后，自动化的希望大大增加
- 软件维护包括
  - 功能性维护：增加软件功能，修复缺陷等
    - 必须有人给出功能规约
    - 本质上无法全自动
      - 李宣东老师：在创造过程中制造，在制造过程中创造
  - 非功能性维护

全自动软件维护

- 能否自动完成软件维护?
  - 大模型出现之后，自动化的希望大大增加
- 软件维护包括
  - 功能性维护
  - 非功能性维护：提高性能、安全性、可理解性等非功能性指标
    - 程序功能（输入输出行为）保持不变
    - 在非功能性指标上有所提升
      - 性能：通过性能测试的执行时间衡量
      - 安全性：通过静态分析工具检查没有发现漏洞
      - 可理解性：代码复杂度指标或由大模型评判
    - 李宣东老师：Hard to solve, easy to verify
    - 可以做到全自动

大规模代码遗产系统的重构与优化

给定软件项目代码，在保证软件功能不变的前提下，自动修改代码，提升软件性能、安全性、可移植性等非功能属性

研究意义——国家需求

- 高性能芯片对大陆禁运
- 软件性能重构可缓解硬件性能问题
  - 为国产硬件争取时间窗口
- 软件可移植性重构可更好支持国产芯片和操作系统
  - 提升产业自动可控能力

研究意义——应用价值

- 软件性能是核心竞争力，其他非功能属性也很重要
- 传统优化人力成本很高，只能针对关键热点代码进行
  - 阿姆达尔定律
- 如果能做到自动优化，就能大幅降低人力成本，对软件进行全面的优化

研究意义——学术意义

- 涉及多个基础核心问题
- 如何验证功能属性不变?
  - 涉及两个程序之间的关系——关系霍尔逻辑
    - 几乎无工具支持和决策过程
  - 如何定义功能属性不变?
- 如何度量非功能属性?

研究意义——学术意义

- 如何实现自动优化?
  - 大模型优化
    - 如何解决长上下文问题
    - 如何解决训练数据问题
    - 如何规划和进行大规模优化
    - 如何保证正确性
  - 传统程序合成
    - 如何解决搜索空间爆炸的问题
    - 如何通用地支持多种不同的优化

已有工作

- 编译优化：
  - 只能产生局部小优化，很少改变时间复杂度
  - 结果人类不可读
- 自动算法优化：
  - 只针对特定编程语言、特定算法范式
  - 难以对复杂多语言项目进行
- 大模型的程序优化：
  - 目前主要针对算法竞赛等封闭环境
  - 在遗产系统上自动优化缺乏系统研究
- 其他非功能属性优化：
  - 针对特定属性、特定缺陷类型
  - 没有形成通用的方法

已有工作：基于传统程序合成的算法优化


熊英飞


吉如一


孙奕灿


张钊


张云帆

数据集：

[FSE24DEMO] ASAC: A Benchmark for Algorithm Synthesis

正确性验证：

[FM24] Proving Functional Program Equivalence via Directed Lemma Synthesis

程序合成：

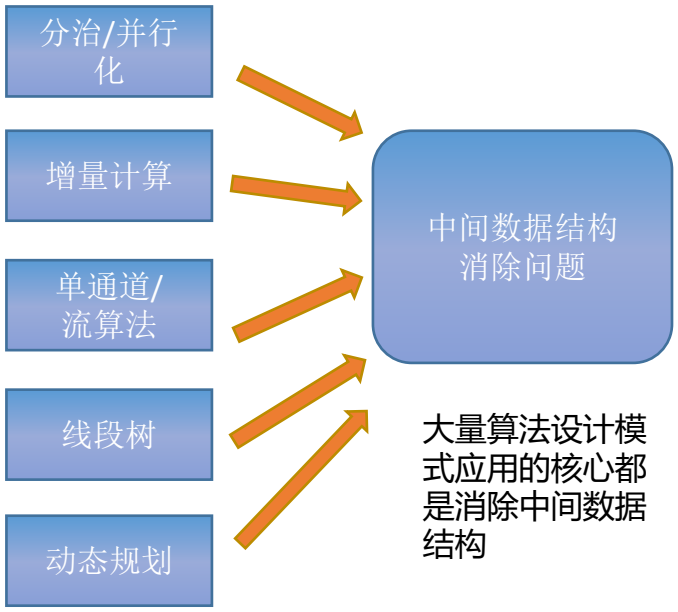
[PLDI24] Superfusion: Eliminating Intermediate Data Structures via Inductive Synthesis

[TOPLAS24] Decomposition-Based Synthesis for Applying D&C-Like Algorithmic Paradigms

[OOPSLA23] Synthesizing Efficient Memoization Algorithms

核心问题：消除中间数据结构

发现高效算法设计的核心是消除中间数据结构，以该发现为核心设计SuFu语言描述算法问题和算法设计模式，可自动优化模型生成或用户编写的代码。



大量算法设计模式应用的核心都是消除中间数据结构

```
tails :: List -> Packed NList
mts xs = maximum (map sum (tails
      xs))
```

算法设计问题描述为对应问题的穷举算法

- 简短易书写
- 不容易出错

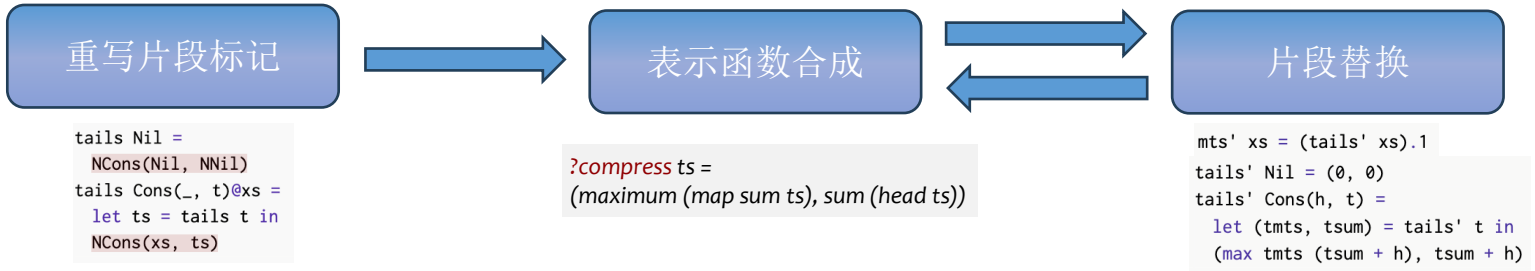
```
dac_id Elt(_)@xs = xs
dac_id Cons(_, _)@xs =
  let (ls, rs) = split xs in
  concat (dac_id ls) (dac_id rs)
```

```
dac_id :: List -> Packed List
dac_mts xs = mts (dac_id xs)
```

算法设计模式描述为具有对应递归结构的等价变换

青方系统：
求解中间数据结构消除问题

提出青方系统自动求解中间数据结构消除问题，通过表示函数合成，将规模较大的算法设计问题分解为独立小程序片段合成问题。



- 290个来源于现有数据集和经典教材的算法问题
  - 涉及分治、增量计算、流计算、线段树等多个算法。
  - 包括最大子段和等经典问题
  - 包括世界顶级算法赛事的难题

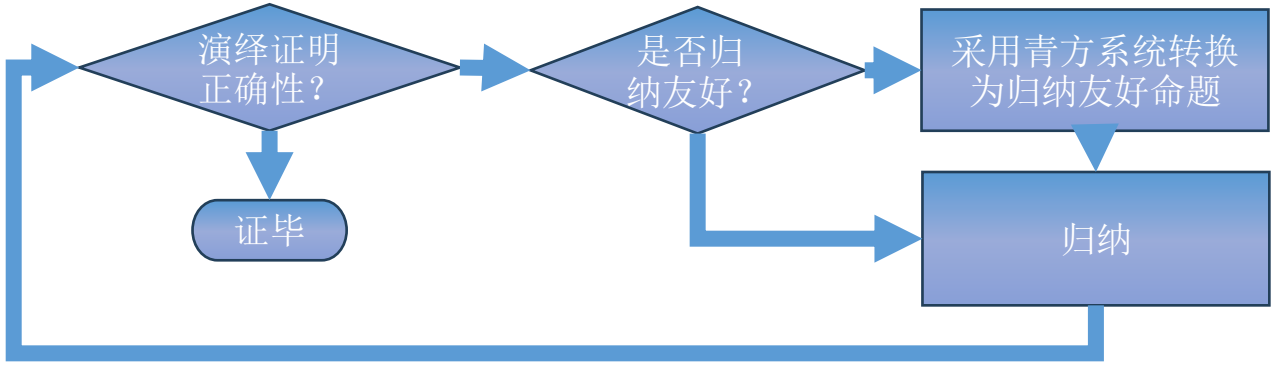
Source	#Solved		Time
Total	264/290	91%	24.4

AutoProof：证明优化结果的正确性

提出Autoproof方法证明青方系统合成程序的正确性

核心创新

归纳友好命题：能保证利用归纳假设的命题



	#Solved (Standard)	#Solved (Extension)	#Solved (Total)	#Fails (Timeout)	AvgTime
AutoProof	**140** (↑ **16.67%**)	**21** (↑ **600%**)	**161** (↑ **30.89%**)	**109**	**3.64s** (↓ **95.47%**)
Cvc4Ind	120	3	123	147	80.36s

青方系统问题

- 基于专用函数式语言，难以直接应用到大型遗产系统
- 虽然已有覆盖大量算法设计范式，但并不能覆盖所有范式

已有工作
基于大模型的自动代码优化

- 大模型可以理解
  - 多种编程语言
  - 多种算法设计范式/程序优化手段
- 直接让大模型来做代码优化
  - 输入代码，提示模型进行优化
  - 效果不好，常常只进行风格优化
- 已有工作：小样本学习[高翠芸等]
  - 收集大量历史优化提交作为样例
  - 给定代码，提示合适的样例引导模型
  - 效果显著提升
- 问题：如何知道哪块代码适合哪个优化?
  - 已有方法采用bm25等信息检索算法，效果有限

赵雨薇

熊英飞

基于规则的优化模式匹配

- 首先对历史优化提交进行聚类
  - 每一类代表一个优化策略
- 从每个类中生成静态分析规则
  - 规则代表应用该优化的前置条件
  - 可用于匹配适用于该优化的代码
  - 规则由大模型生成
- 在软件项目代码上运行分析规则，并提示大模型优化

实验结果

Approach	DeepSeek-V3		GPT-4.1		Gemini-2.5-Pro (on 40 problems)	
	EM	SemEqv	EM	SemEqv	EM	SemEqv
RAPGen	0 (0.0%)	3 (2.0%)	1 (0.7%)	2 (1.3%)	-	-
RAPGen+	3 (2.0%)	6 (4.0%)	3 (2.0%)	7 (4.6%)	-	-
Direct	2 (1.3%)	9 (6.0%)	7 (4.6%)	13 (8.6%)	0 (0.0%)	3 (7.5 %)
RAG	25 (16.6%)	36 (23.8%)	17 (11.3%)	32 (21.2%)	6 (15.0%)	9 (22.5%)
SemOpt	42 (27.8%)	64 (42.4%)	28 (18.5%)	44 (29.1%)	12 (30.0%)	18 (45.0%)
SemOpt + RAG	49 (32.5%)	75 (49.7%)	37 (24.5%)	58 (38.4%)	16 (40.0%)	21 (52.5%)

Repo	Perf Improvement ↑	
	Max	Avg
RocksDB	5.04%	2.41%
Redis	6.14%	1.68%
gRPC	35.48%	3.10%
LevelDB	218.07%	10.40%
spdlog	20.00%	3.27%

小结

- 大型遗产系统的优化和重构是重要问题
- 结合大模型和符号方法有望在该问题上取得显著进展
  - 初步探索已经展现出良好结果
- 相当多的未被解决的问题
  - 性能以外的非功能属性
  - 证明优化前后功能等价
  - 大尺度的代码重构和优化