NASAC青年软件创新奖论坛

程序合成

从补丁合成到算法合成

汇报人：熊英飞

北京大学计算机学院

程序合成

程序合成研究如何自动从规约中生成程序，一直被认为是程序设计理论上最重要的问题之一。

“One of the most central problems in the theory of programming.”

----Amir Pneuli

图灵奖获得者

“（软件自动化）提升软件生产率的根本途径”

----徐家福先生

中国软件先驱

程序合成的应用

	A	B
1	Email	Column 2
2	Nancy.FreeHafer@fourthcoffee.com	nancy freehafer
3	Andrew.Cencici@northwindtraders.com	andrew cencici
4	Jan.Kotas@litwareinc.com	jan kotas
5	Mariya.Sergienko@gradicdesigninstitute.com	mariya sergienko
6	Steven.Thorpe@northwindtraders.com	steven thorpe
7	Michael.Neipper@northwindtraders.com	michael neipper
8	Robert.Zare@northwindtraders.com	robert zare
9	Laura.Giussani@adventure-works.com	laura giussani
10	Anne.HL@northwindtraders.com	anne hl
11	Alexander.David@contoso.com	alexander david

最终用户编程

```
i=round(i);
```

↓

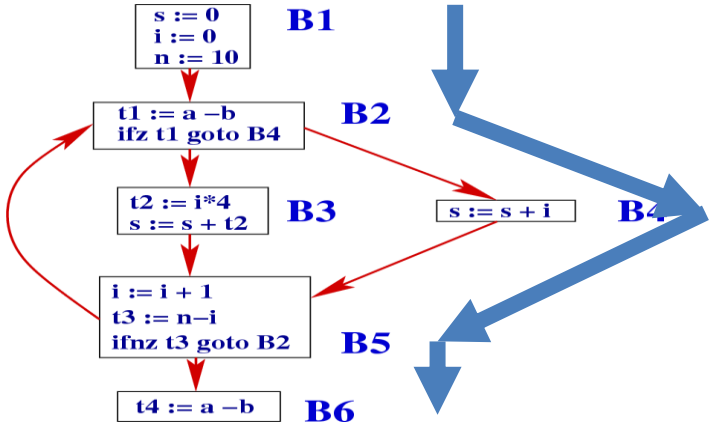
```
a = 6755399441055744.0;
i=(i+a)-a;
```

优化代码

```
/** Compute the maximum of two values
 * @param a first value
 * @param b second value
 * @return b if a is lesser or equal to b, a otherwise
 */
public static int max(final int a, final int b) {
    return (a <= b) ? b : a;
}
```

合成出新的表达式来替换掉旧的

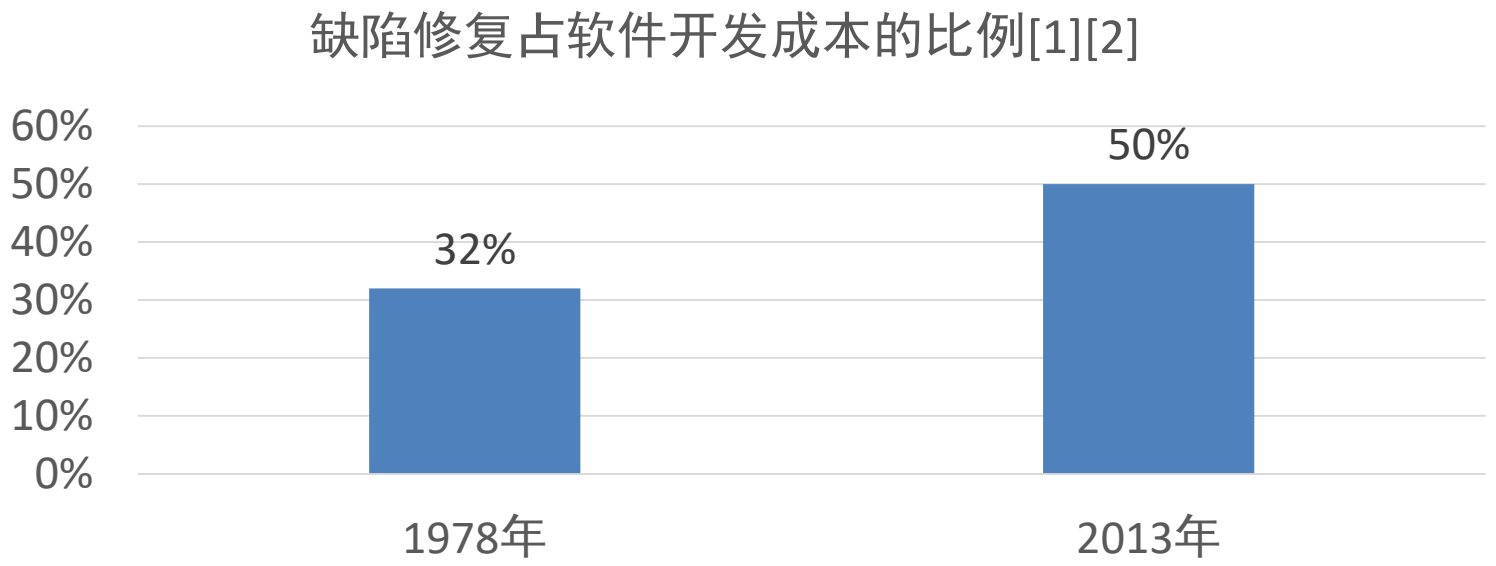
缺陷修复



测试生成

缺陷修复成本

缺陷修复成本不断增加，已成为软件开发的主要成本支出



现代开发团队常没有足够资源修复所有发现的缺陷[3]

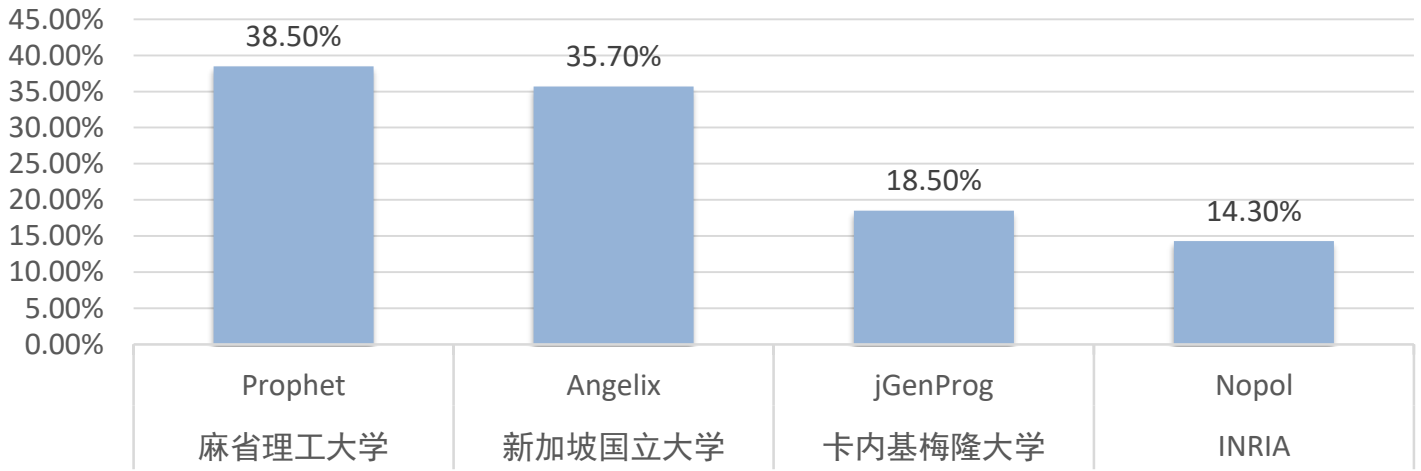
[1] B. P. Lientz, E. B. Swanson, and G. E. Tompkins, “Characteristics of application software maintenance,” Commun. ACM, vol. 21, no. 6, pp. 466–471, 1978
[2] Britton et al. Quantify the time and cost saved using reversible debuggers. Cambridge report, 2013
[3] J. Anvik, L. Hiew, and G. C. Murphy, “Coping with an open bug repository,” eXchange, 2005, pp. 35–39

程序合成应用到修复的问题

传统程序合成面向规约合成程序。

实际软件中规约往往不完备，导致修复正确率低。

已有方法在真实系统上正确率低



“First open challenge.”

-- Claire Le Goues (CMU), ESEC/FSE, 2015

“Key discussion topic”

-- Dagstuhl Report 17022 “Automated Program Repair”

本团队贡献——数据驱动的缺陷修复

基于差别的修复表示模型

基于补丁前后的差别表示补丁空间，解决之前基于状态的空间表示低效和信息丢失问题。

正确修复数量最高可达SOTA的237.5%

MODEL最有影响论文奖、ASE最有影响论文入围

概率和逻辑结合的程序合成框架

提出玲珑框架，采用全新扩展文法引导搜索概率最大的程序，并结合抽象解释剪枝，避免不完备规约影响

缺陷修复正确率从38.5%提升到85%

3篇ACM SIGSOFT杰出论文、1篇SCI高引论文、ISSTA18和ICSE17引用第一和第二

交互式程序合成理论和方法

统一程序合成和最优决策树构建，采用交互式问答辅助程序员复查并过滤掉潜在错误补丁，降低补丁审阅的开销

提高程序员修复成功率62.5%

IEEE TSE封面论文
IEEE TCSE杰出论文

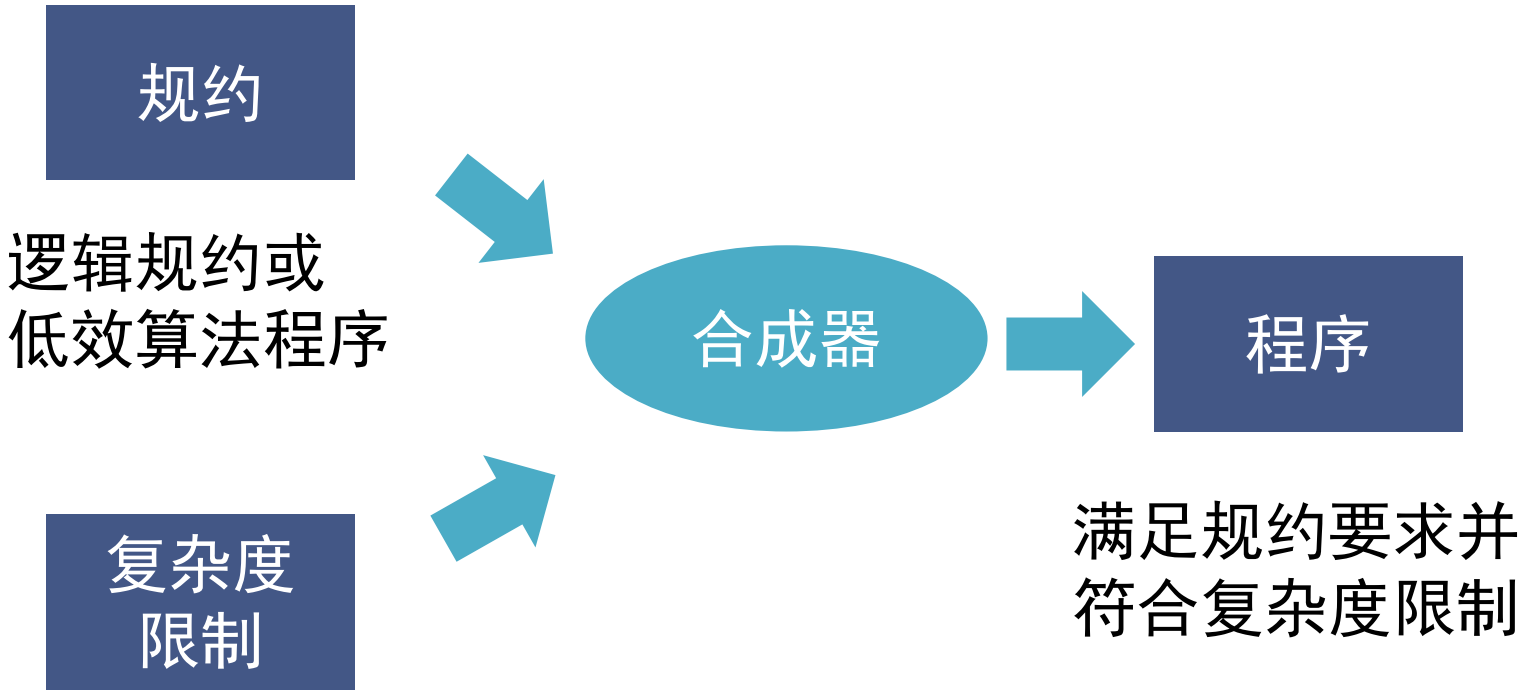
数据驱动的缺陷修复逐渐成为主流

多家企业开发和部署数据驱动的缺陷修复工具。

部分企业直接转换本团队成果。

算法合成——问题定义

合成满足复杂度限制的程序，通常需要应用特定类型的算法



如：O(n^2)或多项式复杂度或
在O(n)个处理器上是常数时间

算法合成意义——从程序合成角度

现有程序合成技术通常只能完成表达式级别的合成，对程序员帮助有限，合成算法才能真正帮助程序员。

掌握和设计算法对多数程序员都是巨大挑战

算法优化往往是程序的错误来源

优化前程序
简洁、优雅、模块化

优化后程序
冗长、繁杂、破坏边界

算法合成意义——从编译优化的角度

目前的编译优化技术只能进行局部小替换，通常很难显著降低程序复杂度。
算法合成有望对程序进行整体大幅优化。

数据流敏感重写 flow-sensitive rewrites

- 条件常量传播 conditional constant propagation
- 主导测试检测 dominating test detection
- 基于流承载的类型缩减转换 flow-carried type narrowing
- 无用代码消除 dead code elimination

语言相关的优化技术 language-specific techniques

- 类型继承关系优化 class hierarchy analysis
- 去虚拟机化 devirtualization
- 符号常量传播 symbolic constant propagation
- 自动装箱消除 autobox elimination
- 逃逸分析 escape analysis
- 锁消除 lock elision
- 锁膨胀 lock fusion
- 消除反射 de-reflection

内存及代码位置变换 memory and placement transformation

- 表达式提升 expression hoisting
- 表达式下沉 expression sinking
- 冗余存储消除 redundant store elimination
- 相邻存储合并 adjacent store fusion
- 卡痕消除 card-mark elimination
- 交汇点分离 merge-point splitting

循环变换 loop transformations

- 循环展开 loop unrolling
- 循环剥离 loop peeling
- 安全点消除 safepoint elimination
- 迭代范围分离 iteration range splitting
- 范围检查消除 range check elimination
- 循环向量化 loop vectorization

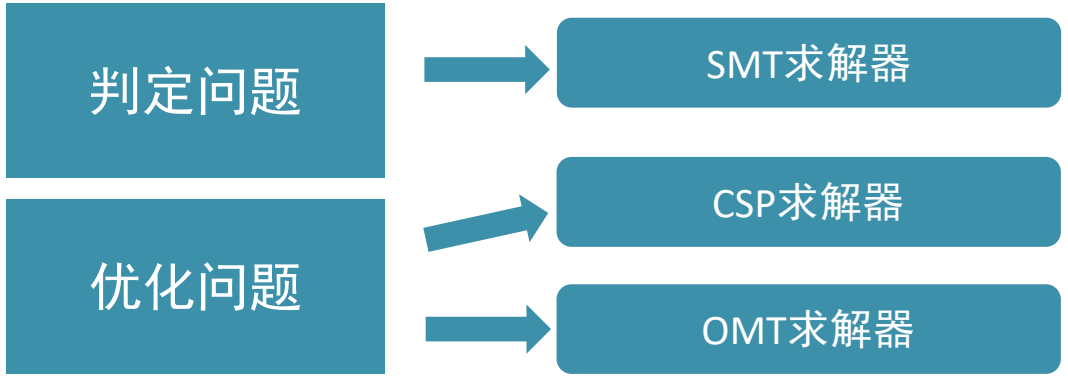
全局代码调整 global code shaping

- 内联 inlining (graph integration)
- 全局代码外提 global code motion
- 基于热度的代码布局 heat-based code layout
- switch 调整 switch balancing
- 抛出内联 throw inlining

常见编译优化技术

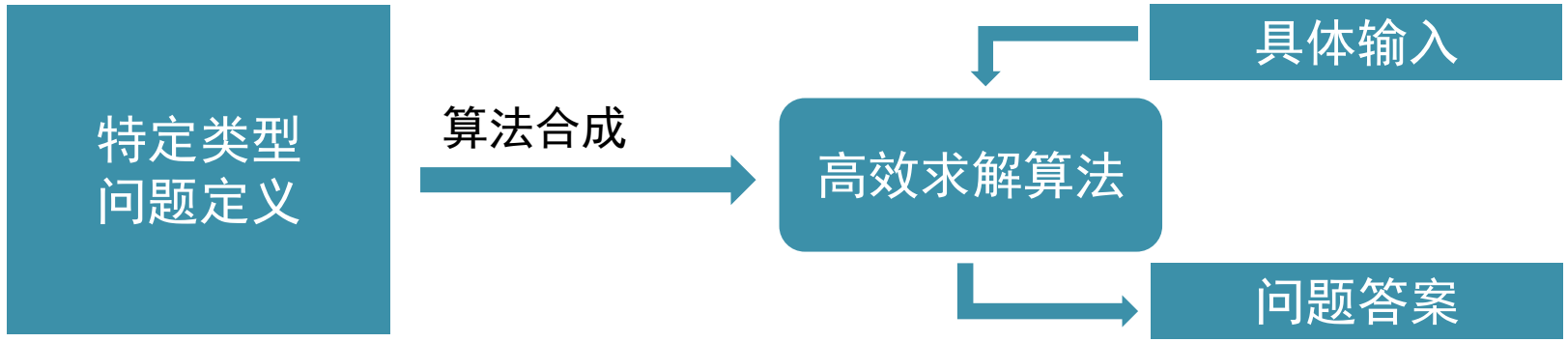
算法合成意义——从约束求解/规划的角度

约束求解/规划试图采用通用算法解决任意判定/优化问题



实际应用中往往只需要求解特定类型的问题，采用通用算法效率不高

算法合成可以对这些问题进行离线优化



算法合成——从程序验证的角度

验证往往只能在协议/模型层面完成，而模型驱动的方法生成的实现往往效率低下。算法合成从验证后的模型生成高效实现，有望解决该问题。

算法合成——例子

最大子段和问题：输入一个整数的列表，求列表上连续一段的最大和。

可用穷举解决：遍历子段开头和列表结尾，选出和最大的子段。

```
p l = maximum (map sum [t | i <- inits l, t <- tails i])
```

该程序作为算法合成的规约。

当前算法复杂度为$O(n^3)$，期望复杂度为$O(n/m)$，其中m是处理器的数量。

算法合成——例子

可能的解决方案：采用分治求解，每次将输入列表平均分成两部分，对每一部分计算最大子段和、最大前缀和、最大后缀和，再用这些结果组合出完整列表的结果。

```
f l = (mps l, mts l, sum l)
  where
  mts = maximum.(scanr (+) 0)
  mps = maximum.(scanl (+) 0)
c (mss1, (mps1, mts1, sum1))
  (mss2, (mps2, mts2, sum2)) =
                  (mss3, (mps3, mts3, sum3))
  where
  mss3 = maximum [mss1, mss2, mts1 + mps2]
  mps3 = max mps1 (sum1 + mps2)
  mts3 = max (mts1 + sum2) mts2
  sum3 = sum1 + sum2
```

```
dc' l =
  if length l <= 1 then (p l, f l)
  else c (dc' (take m l)) (dc' (drop m l))
  where m = div (length l) 2
dc = fst.dc'
```

书写难度和验证难度均远大于前页穷举程序

算法合成——挑战

01 如何得到这样的程序？

- 现有合成算法以枚举为基础，无法合成这么大的程序
- 现有合成算法对递归支持有限，无法合成这样复杂的递归

02 如何知道该程序满足规约的要求？

- 现有合成算法通过SMT求解器验证程序正确性，但SMT求解器的可伸缩性和对递归程序的支持都不足

03 如何知道该程序满足复杂度的要求？

- 虽然存在静态分析程序复杂度的方法，但在递归程序上收敛性不足

程序演算

程序演算：旨在建立一个形式化系统来帮助人们推导出高效的程序。

算法策略：程序演算领域整理出的算法结构表示。

并行分治算法策略

算法模板

```
dc' l =
  if length l <= 1 then (p l, f l)
  else c (dc' (take m l)) (dc' (drop m l))
  where m = div (length l) 2
dc = fst.dc'
```

参数：f, c

应用条件

$$(p\ (l_1 +\!\!+\ l_2), f\ (l_1 +\!\!+\ l_2)) = c\left((p\ l_1, f\ l_1), (p\ l_2, f\ l_2)\right)$$

算法效果

如果c是常数时间，f应用在常数长度的list上是常数时间，那么算法在$m = O(n/logn)$个处理器上的运行时间为$O(n/m)$。

采用算法策略显著降低算法合成的难度

01 如何得到这样的程序？

- 只需要找到模板的参数
- 模板的参数通常不包含递归

显著降低

02 如何知道该程序满足规约的要求？

- 只需要验证应用条件
- 但应用条件仍然涉及带递归的原程序

少量降低

03 如何知道该程序满足复杂度的要求？

- 根据复杂度选择合适的算法策略
- 参数的复杂度要求可以通过语法保证

已解决

挑战1——如何得到模板的参数

问题

- 模板参数的规模仍然太大，超出现有程序合成算法的能力

思路

- 不同于一般的程序合成，这里能拿到应用条件的形式
- 可以针对应用条件设计专门算法

成果

- 目前已经针对三类应用条件设计了合成算法
  - Lifting：用于撰写分治算法和其他相关算法
  - Thinning：用于撰写动态规划算法
  - Incrementalization：用于优化程序中传递的数据结构

挑战2——如何验证程序的正确性

问题

- 原程序带递归，难以形式化验证

思路

- 考虑概率正确性

概率正确性是否可接受？
实践中计算机系统的安全保障都是概率性的
- 软件很难完全验证
- 硬件总有概率损坏

成果

- 基于机器学习领域奥卡姆学习理论，提出奥卡姆求解器的概念和实现算法
- 对任意概率正确级别，给出对应数量的测试用例

目前实现与效果

目前已经实现两个算法合成工具。

- AutoLifter：用于求解并行分治和有相似应用条件的问题
- MetHyl：用于求解动态规划问题

AutoLifter验证

- 57个已有数据集、论文和codeforces.com中的分治和其他问题
  - 最大子段和
  - 字符串转换成数字
  - 检查字符串中括号是否匹配
  - 线段树问题
  - Petrozavodsk冬令营题目（全球243支队伍只有26支解出）
- AutoLifter在时间复杂度O(n/m)下解出56题，平均用时8秒左右，所有答案完全正确
- 解题成功率和速度均高于之前的半自动方法

MetHyl验证

- 37个构造自《算法导论》的动态规划问题
  - 0-1背包问题
  - 木材切割问题（木材切成不同长度有不同价值，求最大价值和）
- MetHyl在97.3%的问题上带来指数级提速
- 在70.3%的问题和标准答案复杂度相同
- 所有答案均正确，平均求解时间不到一分钟

其他进行中的工作

数据集构建

- 包含CSP-S/J、NOI、IOI近十年左右的比赛题目
- 所有题目用MiniZinc形式化描述
  - 可以直接采用现有CSP/OMT求解器求解
- 所有题目同时包含自然语言描述、测试用例

其他算法合成途径

- 约束求解/规划算法中包含了通用的问题求解算法
- 如果针对问题定义尽量将该算法离线化增量化，就有可能得到高效算法

算法合成——总结与展望

- 算法合成——软件自动化下一阶段的重要问题
- 目前研究结果表明发展前景广阔
- 欢迎各位老师/同学一起进行算法合成的研究
  - 合成算法：约束求解/规划/逻辑编程、增量计算、神经网络、程序验证/分析，多条不同的路径可以尝试
  - 语言设计：如何让用户更容易地描述问题？需要程序设计语言、人机交互、可视化背景的研究人员
  - 实际应用：各种软件工程、系统和形式化应用都能用到算法合成
- 本报告相关论文和实现
  - Ruyi Ji, Jingtao Xia, Yingfei Xiong, Zhenjiang Hu. Generalizable Synthesis Through Unification. OOPSLA'21.
  - Ruyi Ji, Yingfei Xiong, Zhenjiang Hu. Black-Box Algorithm Synthesis — Divide-and-Conquer and More. Available at: https://jiry17.github.io/pistool/papers/AutoLifter.pdf
  - Ruyi Ji, Tianran Zhu, Yingfei Xiong, Zhenjiang Hu. Synthesizing Efficient Dynamic Programming Algorithms. Available at: https://jiry17.github.io/pistool/papers/MetHyl.pdf
  - PISTool合成工具集：https://jiry17.github.io/pistool/

吉如一 博士二年级

孙奕灿 博士一年级

李思源 本科四年级

胡振江 讲席教授

感谢各位老师同学！敬请批评指正！