



算法合成

自动应用算法模式合成高效程序

熊英飞
北京大学



知乎问题

程序员 软件开发 编程 计算机 计算机经典课程

作为计算机专业学生，最应该学习的课程前五位是什么？

不论当前大学是否开设这些课程。

关注问题

写回答

邀请回答

好问题 371 13 条评论

1、数据结构与算法

2、计算机组成原理

3、操作系统

4、计算机网络

5、数据库

赞同 1461



124 条评论

软件工程 计算机科学 计算机专业

你认为计算机专业最难学的课程是什么？

关注问题

写回答

邀请回答

好问题 16

课程名	点赞数
算法数据结构	47
大学语文	23
马克思主义	12
汇编语言	5
体系结构	1



算法为什么难：示例

- 最大后缀和：
 - 输入一个整数的列表
 - 对列表上所有后缀求和
 - 返回最大的和
 - $\text{mts}[1, -2, \underline{3}, -1, 2] = 4$
- 穷举算法非常容易实现

```
mts xs = maximum (map sum (tails xs))
```

- 算法复杂度为 $O(n^2)$
- 能否应用算法课的知识来进行优化？



算法为什么难：示例

- 算法课教了一系列算法设计模式，尝试应用迭代

```
mts' xs = (tails' xs).1  
tails' Nil = (0, 0)  
tails' Cons(h, t) =  
    let (tmts, tsum) = tails' t in  
    (max tmts (tsum + h), tsum + h)
```

- 算法复杂度: $O(n)$
- 应用算法设计模式并不容易
 - 迭代：维护一组变量并反复更新，直到得到结果
 - 但对具体问题并没有给出找变量方法和更新方法，主要靠程序员智慧
 - 算法设计仍然是艺术不是科学
 - 优化后的程序长度、理解难度均远超原来的程序



MTS高效程序的运行过程

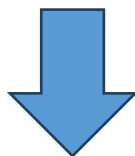
[1, -2, 3, -1, 2]	tmts=0 tsum=0
[1, -2, 3, -1, <u>2</u>]	tmts=2 tsum=2
[1, -2, 3, <u>-1</u> , 2]	tmts=2 tsum=1
[1, -2, <u>3</u> , -1, 2]	tmts=3 tsum=4

```
mts' xs = (tails' xs).1
tails' Nil = (0, 0)
tails' Cons(h, t) =
  let (tmts, tsum) = tails' t in
  (max tmts (tsum + h), tsum + h)
```

研究目标：自动完成程序优化



```
mts xs = maximum (map sum (tails xs))
```



```
mts' xs = (tails' xs).1  
tails' Nil = (0, 0)  
tails' Cons(h, t) =  
  let (tmts, tsum) = tails' t in  
  (max tmts (tsum + h), tsum + h)
```



降低难度： 算法设计是困难耗时的步骤

为什么算法这么难? 知乎

为什么大多人都学不会算法? 播报文章

算法怎么就这么难 Bai 知道

Why does it feel so difficult to learn algorithms and data structures and come up with solutions for real

I work very hard to learn algorithms but still can't grasp it, what should I do?

Answer Follow · 41 Request 1

6 Answers Quora



节省成本： 会算法的程序员需要更高人力成本

招聘中

算法研究员 30-60K

深圳 经验不限 本科

感兴趣 立即沟通

职位描述

机器学习 深度学习 算法

AI算法工程师岗位要求：

- 1、扎实的数学基础，能够利用数学方法进行建模分析，发
 - 2、熟悉大数据分析复杂算法和模型的选型，精通各种算法应
 - 3、掌握常用机器学习、自然语言处理、最优化算法，有较深
 - 4、扎实的编程功底，熟悉JAVA,PYTHON,SCALA, C (一种
 - 5、熟悉大数据分析技术，Hadoop ecosystem, Mahout, Spar
- 专业知识要求：
- 1、熟练掌握Java、C++、python等主流开发语言的一种；
 - 2、了解网络知识、Linux配置和Shell使用；
 - 3、了解常用的软件架构模式、基本的编程工具

招聘中

华为云JAVA开发工程师... 15-30K

深圳 经验不限 本科

感兴趣 立即沟通

职位描述

Java JavaScript Python Shell

【工作职责】

- 1.参与华为云客户轻松上云平台、服务、工具的探索及设计开发，为华为云带来一起把华为云上云服务做成业界第一
- 2.采用真正Devops的开发模式完成特性开发。从需求分析、设计、开发、测试，all都是由全功能团队自治决策，有好的想法可轻松落地。
- 3.完成客户应用、数据上云的自动化能力建设，能力包括服务、工具、脚本、弹，
- 4.除了设计开发例外，进行日常现网的SRE、CI/CD、oncall等现网服务的升级、

【任职要求】

业务技能要求：

- 1.有某方面技术擅长者优先，可以是以下任何一个技术，如操作系统架构OS（例如ES、Hbase、Hive等）、容器、EI等，。
- 2.能够采用Shell脚本、Python脚本进行自动化部署、升级、运维优先。
- 3.主动接受挑战，不推诿，积极和善沟通。

同一时间、同一地点、同一公司、同一部门的两个招聘岗位



提高质量：算法优化是缺陷来源

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

- SeL4: 著名经验证操作系统内核
 - 第一阶段：采用Haskell的算法设计
 - 验证导致500+修改
 - 第二阶段：采用C的系统实现
 - 验证导致50+修改



提升效率：自动优化程序

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。自动应用算法模式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

大型语言模型是否解决这个问题？



你能优化下面这个程序，将其变成 $O(n)$ 时间复杂度的程序吗？

```
mts xs = maximum (map sum (tails xs))
```

由 DeepSeek-Coder-V2 生成



当然可以。原始程序的时间复杂度是 $O(n^2)$ ，因为 `tails` 函数会生成所有后缀列表，而 `map sum` 会对每个后缀列表求和。我们可以通过一次遍历来优化这个程序，使其时间复杂度变为 $O(n)$ 。

以下是优化后的代码：

haskell

复制

```
maximumSubarraySum :: [Int] -> Int
maximumSubarraySum xs = go xs 0 0
  where
    go [] maxSoFar currentSum = maxSoFar
    go (x:xs) maxSoFar currentSum =
      let currentSum' = max 0 (currentSum + x)
          maxSoFar' = max maxSoFar currentSum'
      in go xs maxSoFar' currentSum'
```

这个函数使用了一个辅助函数 `go`，它维护了两个变量：`maxSoFar` 表示到目前为止找到的最大子数组和，`currentSum` 表示当前子数组的和。通过一次遍历，我们可以在 $O(n)$ 时间内找到最大子数组和。

你能优化下面这个mts程序，将其变成 $O(n)$ 时间复杂度的程序吗？

```
cal xs = sum (map (+1) xs) - length xs
mts xs = maximum (map cal (tails xs))
```



由 DeepSeek-Coder-V2 生成

要将给定的 `mts` 程序优化到 $O(n)$ 时间复杂度，我们需要避免重复计算。原始程序中的 `cal` 函数和 `mts` 函数都涉及对列表的多次遍历，导致时间复杂度为 $O(n^2)$ 。我们可以通过一次遍历来计算所需的所有信息。

以下是优化后的代码：

haskell

复制

```
import Data.List (tails)

-- 优化后的 cal 函数
cal :: [Int] -> Int
cal xs = sum (map (+1) xs) - length xs

-- 优化后的 mts 函数
mts :: [Int] -> Int
mts xs = maximum $ scanl (\acc x -> acc + x + 1) 0 xs

-- 测试
main :: IO ()
main = do
    let testList = [1, 2, 3, 4]
    12 print $ mts testList -- 输出应该是 14
```

对约束进行
小等价变换

开始胡说并
给出完全不
对的代码



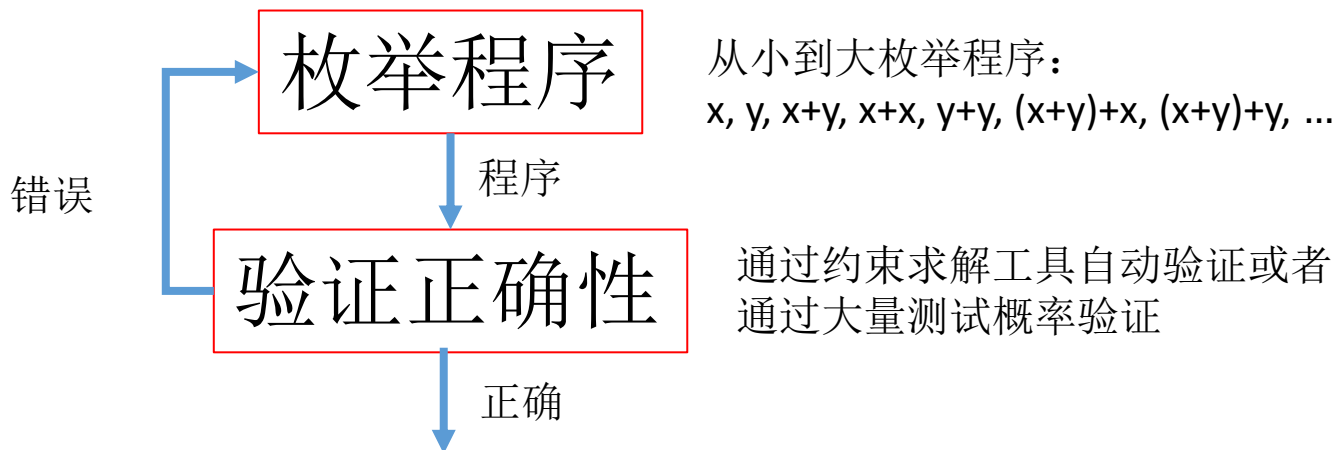
传统求解方法：程序演算

- 通过一系列程序变换半自动得到优化后的程序
- 但算法设计往往需要“根据具体问题设计关键模块”这样的操作
 - 通用程序变换规则难以完成问题特定的设计
- 因此，程序演算保持半自动求解



我们的路线： 基于搜索的归纳程序合成

- 基本方法：枚举+验证



- 其他高级算法可以看做是这个基本形式的优化
- 可以搜索到问题特定的模块，只要包含在空间中
- 可以利用文法控制算法复杂度
 - $O(1)$: 程序空间中只包括 $O(1)$ 操作
 - $O(n)$: 程序空间中只包括一个遍历输入数据的循环



我们的路线： 基于搜索的归纳程序合成

```
mts xs = maximum (map sum (tails xs))
```



合成 $O(n)$ 的等价程序

```
mts' xs = (tails' xs).1  
tails' Nil = (0, 0)  
tails' Cons(h, t) =  
  let (tmts, tsum) = tails' t in  
  (max tmts (tsum + h), tsum + h)
```

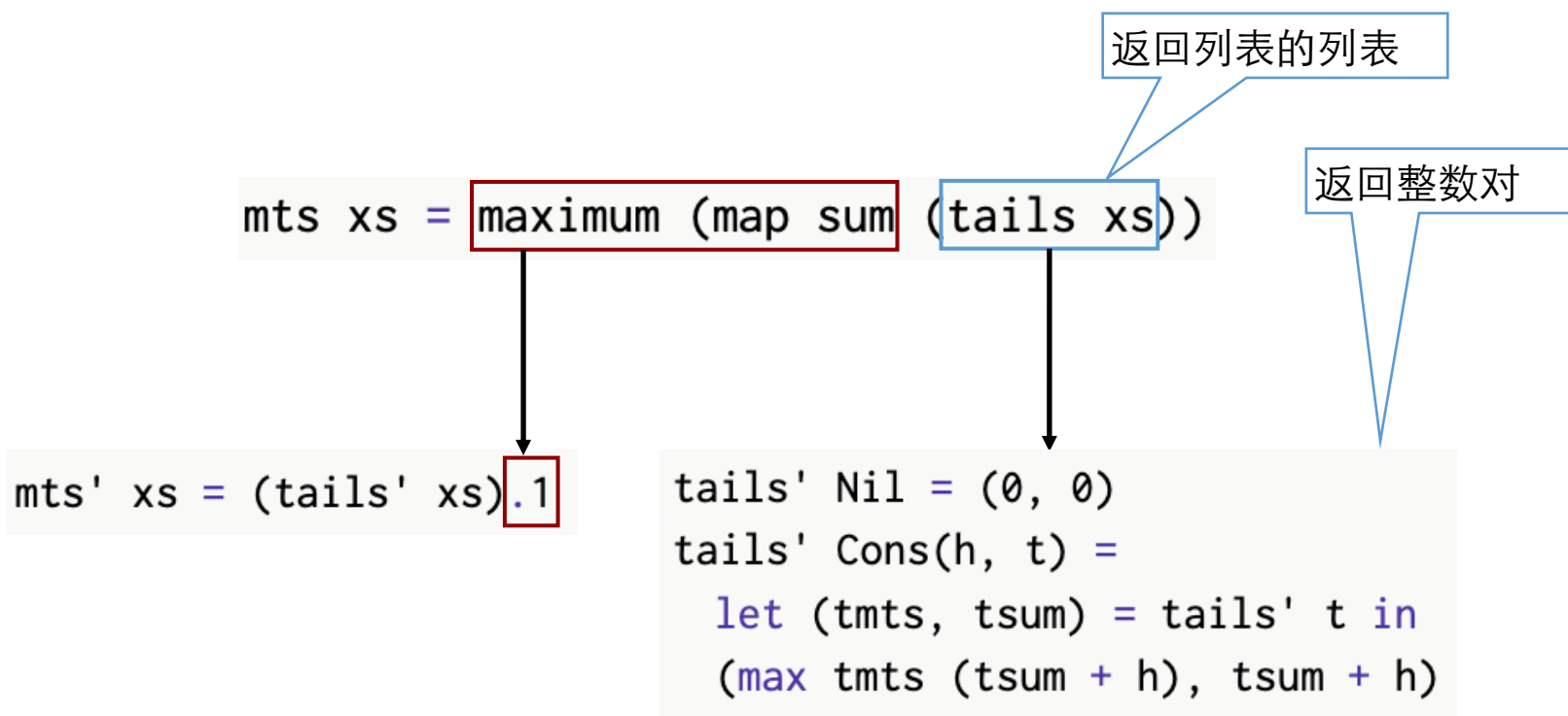
基于搜索程序合成的瓶颈： 可伸缩性



- 从小到大的枚举不可能枚举到很大的程序
 - 程序空间随程序大小呈指数增长
- 当目标程序使用超过5个运算符的时候，现代程序合成工具就经常失效
- 实用算法远远超过这个规模



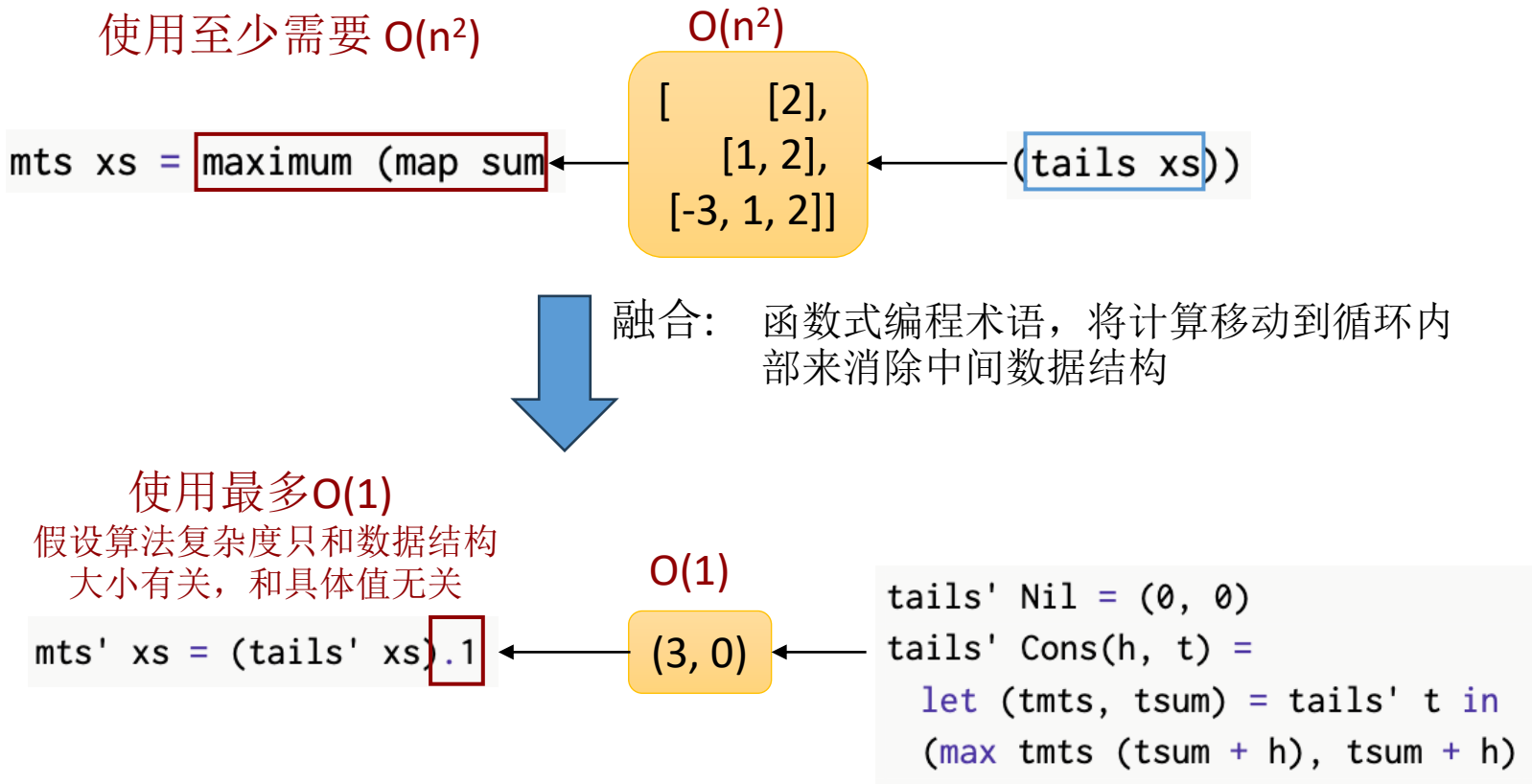
改进思路：独立优化片段



问题1：什么片段需要被重写？

问题2：片段之间并不独立，如何独立优化？

针对问题2分析优化关键： 中间数据结构

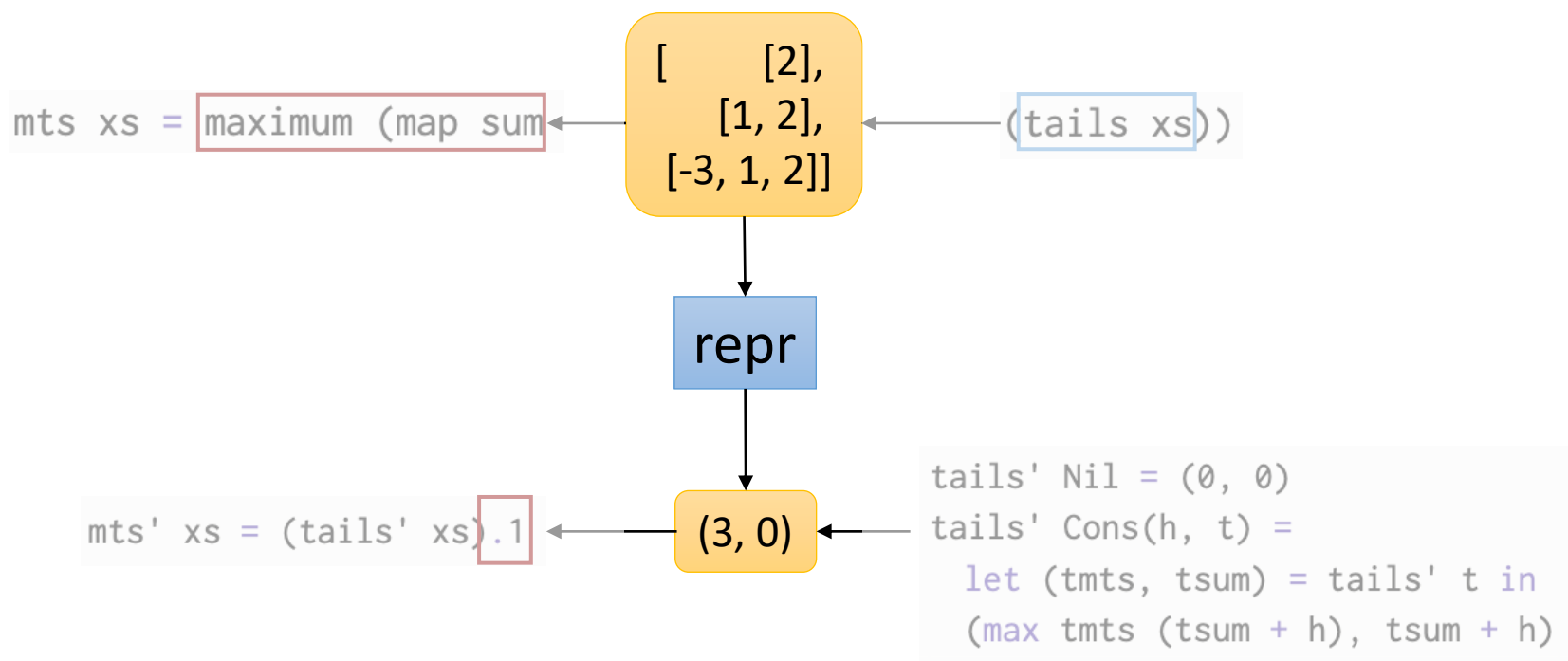


青方系统



青方=SuFu=SuperFusion=自动融合

- 首先合成一个表示函数，在优化前后的数据间做转换。
 - *repr*: *tails*返回值 → *tails'*返回值
- 每个片段成为独立的合成问题





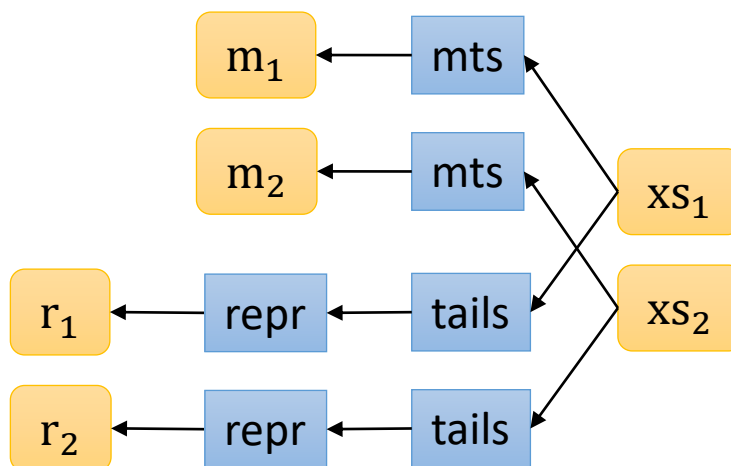
如何合成repr函数?

repr需要为最终返回值提供足够多的信息

```
mts xs = maximum (map sum (tails xs))
```

If $m_1 \neq m_2$

then $r_1 \neq r_2$



基于以上规约，寻找一个输出尽可能小的repr



问题1： 什么部分需要被重写？

- 设计语言扩展让用户标记要压缩的数据结构
- 系统推出所有读写该数据结构的片段

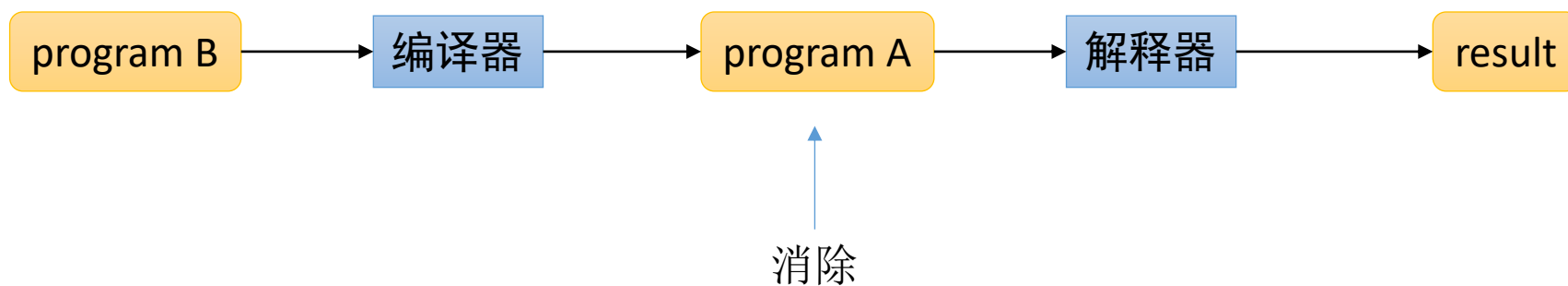
```
tails :: List -> Packed NList  
mts xs = maximum (map sum (tails  
    xs))
```



```
tails Nil =  
    NCons(Nil, NNil)  
tails Cons(_, t)@xs =  
    let ts = tails t in  
    NCons(xs, ts)  
  
mts xs =  
    let ts = tails xs in  
    maximum (map sum ts)
```



应用：合成解释器





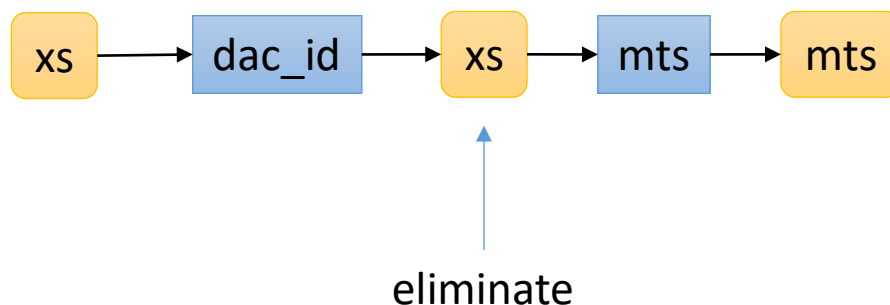
应用：应用其他算法设计模式

- 很多算法设计模式采用特定模式遍历输入
- 捕获这种模式作为模版

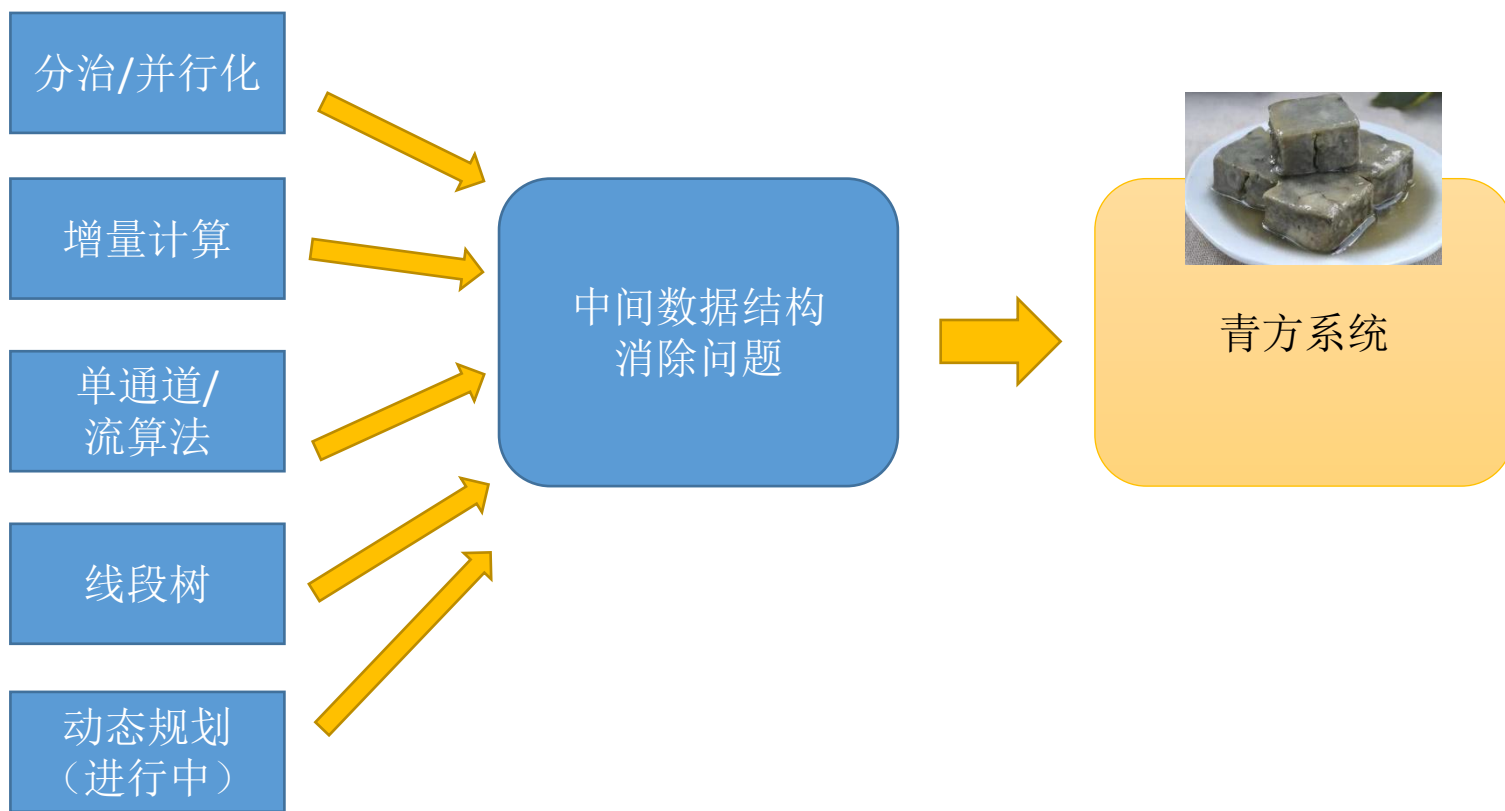
```
dac_id Elt(_)@xs = xs
dac_id Cons(_, _)@xs =
  let (ls, rs) = split xs in
  concat (dac_id ls) (dac_id rs)
```

```
mts xs = maximum (map sum (tails
  xs))
```

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



青方系统 支持多种算法设计模式





方法效果

- 290个已有数据集、算法论文和codeforces.com中的融合和其他问题
 - 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）

Source	#Solved		Time
Total	264/290	91%	24.4

在线演示

<http://8.140.207.65/new-demo>



SuperFusion

Eliminating Intermediate Data Structures via Inductive Synthesis

Reference Program

01Knapsack ▾

AutoLabel ☒

NonScalar ☐

Optimize

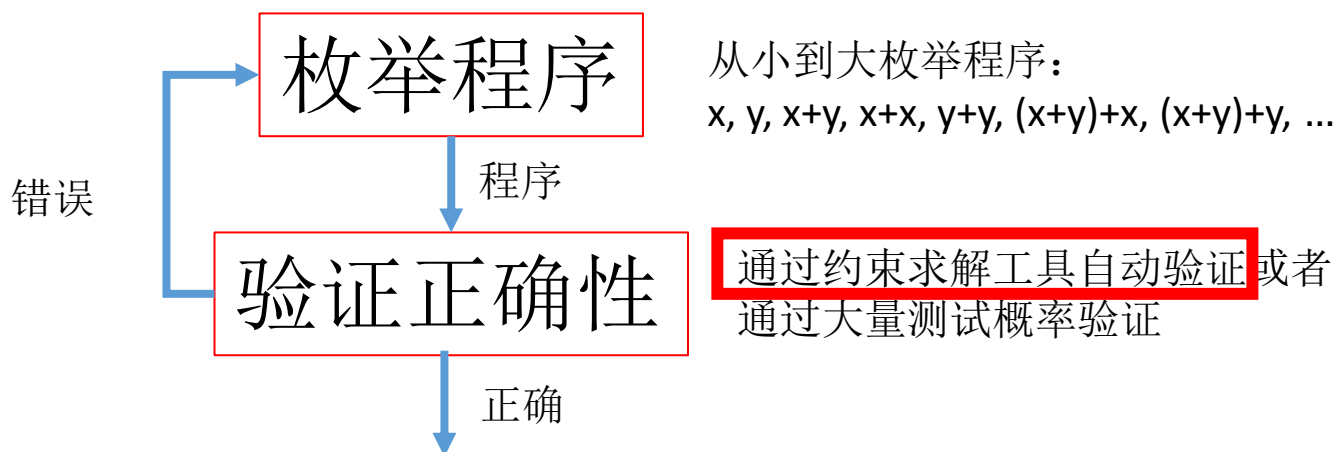
```
1 import "compress";
2 import "list";
3
4 Item = Int * Int;
5 ItemList = List (Item);
6 Plan = Reframe (ItemList);
7
8 fun sumw items = sum (map fst items);
9 fun sumv items = sum (map snd items);
10
11 step :: Int -> Item -> Plan -> List (Plan);
12 fun step lim item plan =
13   if sumw plan + fst item <= lim then
14     Cons {Cons {item, plan}, Cons {plan, Nil}}
15   else Cons {plan, Nil};
16
17 gen :: Int -> ItemList -> List (Plan);
18 fun gen lim = function
19   | Nil -> Cons {Nil, Nil}
```

Optimized Program

```
1 import "list";
2
3 fun sumw items = sum (map fst items);
4
5 fun sumv items = sum (map snd items);
6
7 step :: Int -> (Int * Int) -> (Int * Int) -> List (Int * Int);
8 fun step lim item plan =
9   if (fst plan) + fst item <= lim
10  then Cons {
11    (let c0 = (fst item) in
12     let c1 = (snd item) in
13     {(fst plan) + c0, (snd plan) + c1}),
14    Cons {plan, Nil unit}
15  }
16  else Cons {plan, Nil unit};
17
18 gen :: Int -> (List (Int * Int)) -> List (Int * Int);
19 fun gen lim = function
```

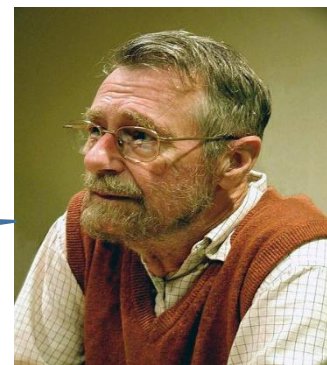


看起来很好?



- 合成程序包含大量循环，现有系统无法验证
- 目前通过测试来保证正确性

Testing shows the presence,
not the absence of bugs.



Edsger W. Dijkstra



如何证明程序优化是正确的？

- 优化后的程序应该等价于优化前的程序
- 即：程序等价性证明
- 本质上，需要证明类似这样的命题

$$\forall xs : \text{List}. \quad \text{sum} (\text{rev } xs) = \text{sum } xs$$

$$\forall xs : \text{List}. \quad \text{sum} (\text{rev } xs) = \text{sum} (\text{sort } xs)$$



程序等价性证明为什么难?

- 传统一阶逻辑的证明已经有较好的自动化方法
- 涉及归纳定义的数据结构和结构递归函数
- 证明通常需要采用归纳证明

```
Inductive List = nil | cons Int List;
```

```
Let rev (l:List) =  
match l with  
| nil → nil  
| cons h t → snoc h (rev t)  
end;
```

```
Let sort (l:List) =  
match l with  
| nil → nil  
| cons h t → ins h (sort t)  
end;
```

```
Let sum (l:List) =  
match y with  
| nil → 0  
| cons h t → h + (sum t)  
end;
```



归纳证明为什么难？

- 在原命题上直接归纳不一定会成功
- 示例 $\forall xs : \text{List}. \text{sum}(\text{rev } xs) = \text{sum}(\text{sort } xs)$
 - 对xs做归纳，分为两种情况讨论
 - 情况1: $xs = []$
 - 直接计算，等式左右两边都为[], 成功证明
 - 情况2: $xs = \text{cons } h \ t$
 - 有归纳假设 $\text{sum}(\text{rev } t) = \text{sum}(\text{sort } t)$
 - 原命题变形得到 $\text{sum}(\text{snoc } h (\text{rev } t)) = \text{sum}(\text{ins } h (\text{sort } t))$
 - 无法应用归纳假设，也无法化简，证明卡住
 - 可以证明，该命题无法通过直接归纳证明
- 需要引入引理，并归纳证明引理
- 如何知道需要什么引理？



现有工作

- 启发式搜索引理，然后尝试自动证明
 - 从小到大枚举所有引理
 - 基于机器学习生成引理
 - 基于和原命题的文本相似性搜索引理
- 因为无保证，大量的时间都浪费在无效引理上



归纳友好命题

- 归纳友好命题一定保证能应用归纳假设
- 归纳友好命题形式1:
 - 等式的一边只有一个结构递归函数调用
- 之前的例子不具备该特点

$$\forall xs : \text{List}. \quad \text{sum} (\text{rev } xs) = \text{sum} (\text{sort } xs)$$

- 能否将该命题转换成归纳友好命题?



转换为归纳友好命题

- 原命题: $\forall xs : \text{List}. \text{sum} (\text{rev } xs) = \text{sum} (\text{sort } xs)$
- 合成结构递归函数f
 - $f \text{ xs} = \text{sum} (\text{sort } xs)$
- 将原命题改写为两个新命题
 - $\text{sum} (\text{rev } xs) = f \text{ x}$
 - $f \text{ xs} = \text{sum} (\text{sort } xs)$
- 两个新命题都符合归纳友好命题形式1, 可以直接归纳证明



如何得到函数f?

- $f\ xs = \text{sum} (\text{sort}\ xs)$
- 标准的程序合成问题
- 实现中调用青方系统解决
- 即：青方系统自己被用来证明自己合成程序的正确性



自动证明系统AutoProof

- 提出三种归纳友好命题形式
- 提出两种转换策略，将命题转换到归纳友好形式
- 转换策略的执行需要求解程序合成问题
 - 由青方系统解决



实验结果

来自经典测试集CLAM, Isaplanner, TIP等的270个证明问题

	#Solved (Standard)	#Solved (Extension)	#Solved (Total)	#Fails (Timeout)	AvgTime
AUTO _{PROOF}	140 (↑ 16.67%)	21 (↑ 600%)	161 (↑ 30.89%)	109	3.64s (↓ 95.47%)
CVC4 _{IND}	120	3	123	147	80.36s



小结

- 算法设计是重要的
 - 算法是高效软件开发的基础
- 算法设计是困难的
 - 研究只是整理出算法设计模式
 - 能否应用全凭程序员能力
- 算法合成是可能的
 - 算法设计的一个关键是消除中间数据结构
 - 通过找合适的repr函数，算法设计可以转成多个小的程序合成问题
- 算法合成还有很多要解决的问题
 - 如何支持更多算法设计模型
 - 如何自动优化大型程序
 - 如何更好集成合成和验证
- 期待更多人投入算法合成的研究！



相关论文

- Ruyi Ji, Yuwei Zhao, Nadia Polikarpova, Yingfei Xiong*, Zhenjiang Hu. Superfusion: Eliminating Intermediate Data Structures via Inductive Synthesis. PLDI'24: ACM SIGPLAN Conference on Programming Language Design and Implementation, June 2024.
- Yican Sun, Ruyi Ji, Jian Fang, Xuanlin Jiang, Mingshuai Chen, Yingfei Xiong*. Proving Functional Program Equivalence via Directed Lemma Synthesis. FM'24: 26th International Symposium on Formal Methods, September 2024.
- Yican Sun, Xuanyu Peng, Yingfei Xiong*. Synthesizing Efficient Memoization Algorithms. OOPSLA'23: Object-Oriented Programming, Systems, Languages and Applications, October 2023.
- Ruyi Ji, Yuwei Zhao, Yingfei Xiong*, Di Wang, Lu Zhang, Zhenjiang Hu. Decomposition-Based Synthesis for Applying D&C-Like Algorithmic Paradigms. TOPLAS: ACM Transactions on Programming Languages and Systems, Accepted, January 2024