



软件分析

程序合成：枚举

熊英飞
北京大学



能否躺着把程序编了？

- 程序合成：从规约中自动生成程序
 - 规约一般为形式化规约
 - 规约也可以是自然语言描述



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

----Amir Pnueli
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



典型应用——自动编程

文心大模型3.5

你好！最大子段和问题是指给定一个整数的列表，然后求这个列表中连续的一段，其中这一段的整数的和最大，求这个最大的和。你能写一个程序求解该问题吗？



当然可以，以下是一个Python程序，用于求解最大子段和问题：

python

复制代码

```
1 def maxSubarraySum(nums):
2     if not nums:
3         return 0
4
5     max_sum = nums[0]
6     for i in range(1, len(nums)):
7         max_sum = max(nums[i], max_sum + nums[i])
8
9     return max_sum
10
11 # 示例用法
12 nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
13 print(maxSubarraySum(nums)) # 输出 15
```

在这个程序中，我们定义了一个名为 `maxSubarraySum` 的函数，它接受一个整数列表作为输入，并返回这个列表中连续的一段，其中这一段的整数的和最大。

在函数中，我们首先检查输入列表是否为空，如果是，则返回0。否则，我们初始化一个变量 `max_sum` 为列表中的第一个元素，然后遍历列表中的所有元素。对于每个元素，我们将其与 `max_sum` 进行比较，并更新 `max_sum` 的值。最后，我们返回 `max_sum` 作为函数的结果。

在示例用法中，我们定义了一个整数列表 `nums`，并调用 `maxSubarraySum` 函数来求解最大子段和问题。最后，我

典型应用——Data Wrangling

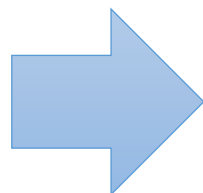


	A	B
1	Email	Column 2
2	Nancy.FreeHafer@fourthcoffee.com	nancy freehafer
3	Andrew.Cencici@northwindtraders.com	andrew cencici
4	Jan.Kotas@litwareinc.com	jan kotas
5	Mariya.Sergienko@gradicdesigninstitute.com	mariya sergienko
6	Steven.Thorpe@northwindtraders.com	steven thorpe
7	Michael.Neipper@northwindtraders.com	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
12	Kim.Shane@northwindtraders.com	kim shane
13	Manish.Chopra@northwindtraders.com	manish chopra
14	Gerwald.Oberleitner@northwindtraders.com	gerwald oberleitner
15	Amr.Zaki@northwindtraders.com	amr zaki
16	Yvonne.McKay@northwindtraders.com	yvonne mckay
17	Amanda.Pinto@northwindtraders.com	amanda pinto

典型应用– Superoptimization



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



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



经典程序合成定义

- 输入:
 - 一个程序空间 $Prog$, 通常用文法表示
 - 一条规约 $Spec$, 通常为逻辑表达式
- 输出:
 - 一个程序 $prog$, 满足
 - $prog \in Prog \wedge prog \mapsto Spec$



例子：max问题

- 语法：

$$\begin{array}{lcl} \text{Expr} & ::= & 0 \mid 1 \mid x \mid y \\ & & | \text{Expr} + \text{Expr} \\ & & | \text{Expr} - \text{Expr} \\ & & | (\text{ite } \text{BoolExpr } \text{Expr } \text{Expr}) \\ \text{BoolExpr} & ::= & \text{BoolExpr} \wedge \text{BoolExpr} \\ & & | \neg \text{BoolExpr} \\ & & | \text{Expr} \leq \text{Expr} \end{array}$$

- 规约：

$$\begin{array}{l} \forall x, y : \mathbb{Z}, \quad \text{max}_2(x, y) \geq x \wedge \text{max}_2(x, y) \geq y \\ \quad \wedge (\text{max}_2(x, y) = x \vee \text{max}_2(x, y) = y) \end{array}$$

- 期望答案： $\text{ite } (x \leq y) \ y \ x$



程序合成是软件分析问题

```
Expr ::= 0 | 1 | x | y
      | Expr + Expr
      | Expr - Expr
      | (ite BoolExpr Expr Expr)
BoolExpr ::= BoolExpr ∧ BoolExpr
          | ¬BoolExpr
          | Expr ≤ Expr
```



```
int x, y;
int Main(Tree[int] prog, int _x, int _y) {
    x=_x; y=_y;
    return Expr(prog);
}

int Expr(Tree[int] prog) {
    switch(prog.value) {
        case 0: return 0; break;
        case 1: return 1; break;
        case 2: return x; break; ...
        case 4:
            return Expr(prog.child[0])+Expr(prog.child[1]);
            break;
        ...
        case 6:
            return BoolExpr(prog.child[0]) ?
                Expr(prog.child[1]) : Expr(prog.child[2]);
            break;
        ...
    }
}
```

程序是否满足性质：

$\exists x. prog = x \rightarrow Spec$



SyGuS: 程序合成问题的标准化

- 输入：语法 G ，约束 C
- 输出：程序 P ， P 符合语法 G 并且满足 C
- 输入输出格式：SyGuS-IF
 - <https://sygus-org.github.io/language/>



SyGuS: 定义逻辑

- 和SMT-Lib完全一致
- (set-logic LIA)
- 该逻辑定义了我们后续可以用的符号以及这些符号的语法/语义，程序的语法应该是该逻辑语法的子集。



SyGuS : 语法

```
(synth-fun max2 ((x Int) (y Int)) Int
  ((Start Int (x
    y
    0
    1
    (+ Start Start)
    (- Start Start)
    (ite StartBool Start Start)))
  (StartBool Bool ((and StartBool StartBool)
    (or StartBool StartBool)
    (not StartBool)
    (<= Start Start)
    (= Start Start)
    (>= Start Start)))))
```



约束

```
(declare-var x Int)  
(declare-var y Int)
```

约束表示方式和SMTLib一致

```
(constraint (>= (max2 x y) x))  
(constraint (>= (max2 x y) y))  
(constraint(or (= x (max2 x y))  
                (= y (max2 x y))))
```

```
(check-synth)
```



期望输出

输出:

```
(define-fun max2 ((x Int) (y Int)) Int (ite (<= x y) y x))
```

输出必须:

- 满足语法要求
 - 即, 语法和SMTLib/Logic不一致就合成不出正确的程序
- 满足约束要求
 - 一般要求可以通过SMT验证



程序合成的历史

1957

- 程序合成的开端
- Alonzo Church: 电路合成问题

2000前

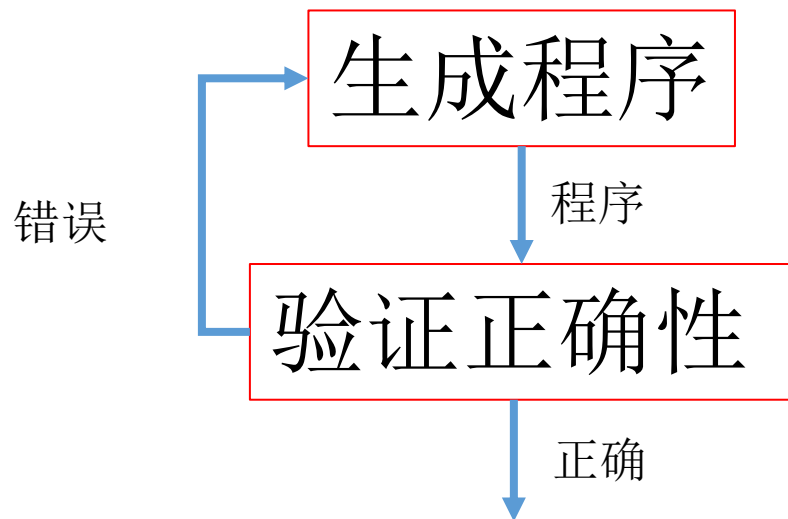
- 演绎合成

2000后

- 归纳合成

归纳程序合成

——程序空间上搜索



Q1:如何产生下一个被搜索的程序?

Q2:如何验证程序的正确性?

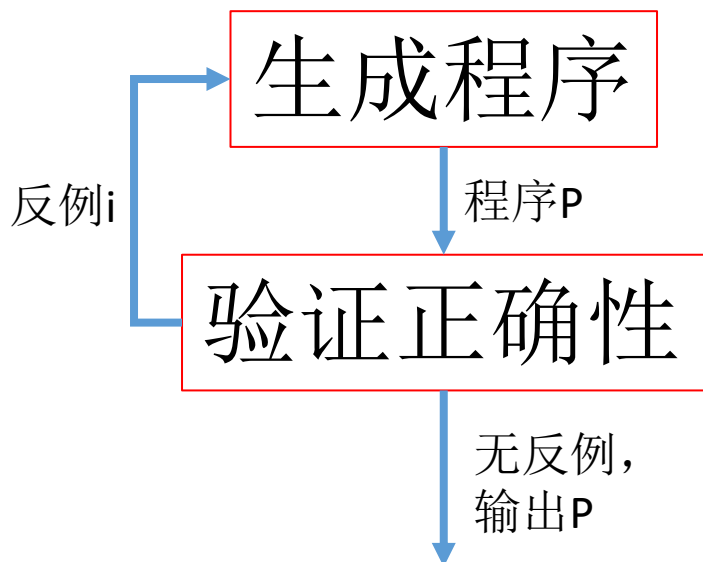


如何验证程序的正确性?

- 采用本课程学习的技术
 - 抽象解释
 - 符号执行
- 传统程序合成技术通常只处理表达式
 - 可直接转成约束让SMT求解
 - SyGuS直接提供支持



CEGIS——基于反例的优化



Armando Solar-Lezama
麻省理工大学教授

- 采用约束求解验证程序的正确性较慢
- 执行测试较快
 - 大多数错误被一两个测试过滤掉
- 将约束求解器返回的反例作为测试保存
- 验证的时候首先采用测试验证
- 如果程序空间是有限的（如程序大小有上限），该方法保证最后收敛到正确程序



选择添加的反例

- 极限情况下，SMT返回的反例每次只从程序空间中排除一个程序
 - 调用SMT求解器次数和程序空间大小相同
- 改进：多次询问SMT求解器，然后选择最好的加入测试集
 - 为什么不都加入？因为程序合成的效率和测试集的大小有关
 - 测试越多，运行测试越慢
 - 生成程序的算法也会和测试集大小有关



什么样例最好？

- 在剩余的语法空间上排除掉尽可能多的程序
 - 剩余的语法空间：满足当前测试集的程序集合
 - 难以快速计算
- LearnSy：通过近似输出相同的程序对的个数来估计被排除的程序
 - 输出相同的程序对越多，被一个输入输出对排除的程序期望数量就越少
 - 原本为交互式程序合成设计，但因为速度快，可直接用来加速非交互式程序合成



吉如一
北京大学博士生
LearnSy提出者



基于语法估算相同对的数量

- 对每条规则 r 赋一个参数 $p[r]$, 表示:
 - 两个从该规则展开的程序, 当子程序的输出不同时, 输入相同的概率
 - 例如: $E \rightarrow E + E$, 参数 $p[E \rightarrow E + E]$ 表示如果 $(e_1, e_2) \neq (e'_1, e'_2)$, 那么 $e_1 + e_2 = e'_1 + e'_2$ 的概率
- 对每对规则赋一个参数, 表示:
 - 从这两个规则 r_1, r_2 分别展开的两个程序 $p[r_1, r_2]$, 输出相同的概率
 - 例如: $E \rightarrow E + E$ 和 $E \rightarrow E * E$, 参数 $p[E \rightarrow E + E, E \rightarrow E * E]$ 表示任意两个程序 $e_1 + e_2$ 和 $e_3 * e_4$ 的概率



基于语法估算相同对的数量

- 对于语法递归计算，令 $S(A)$ 为从 A 展开的程序数量， $S(r)$ 为从规则 r 展开的程序数
 - $E(r) = 1$
 - 如果 r 的右边没有非终结符
 - $E(r) = E(B) + (S(B) - E(B)) * p[r]$
 - 如果 r 形如 $A \rightarrow \dots B \dots$ ，即右边只有一个非终结符
 - $E(r) = E(B) * E(C) + (S(B) * S(C) - E(B) * E(C)) * p[r]$
 - 如果 r_3 形如 $A \rightarrow \dots B \dots C \dots$ ，即右边有两个非终结符
 - $E(A) = E(r_1) + E(r_2) + p[r_1, r_2] * S(r_1) * S(r_2)$
 - 如果 A 被两条规则 r_1 和 r_2 展开



估算剩余空间相同对的数量

- 前述计算是关于整个程序空间的
- 实际需要计算剩余空间相同对的数量
- 如何知道哪些程序在剩余空间？
 - 上一轮产生了一个程序 p_c ，该程序一定满足当前测试集
- 可以用类似的方式估算每个程序对和该程序 p_c 等价的概率
 - 程序 p_1 和 p_2 等价且满足测试的概率 = p_1 和 p_2 等价的概率 * p_1 和 p_c 等价的概率
- 这样等价对的数量也可以基于非终结符递归计算



如何得到概率模型的参数？

- 对于每个输入和每个参数，随机采样程序，然后统计概率
 - 可以在选择输入的时候快速按需计算
 - 因为程序大小很小，所以能快速计算

如何产生下一个被搜索的程序？



- 多种不同方法
 - 枚举法 —— 按照固定格式搜索
 - 约束求解法 —— 转成约束求解问题
 - 空间表示法 —— 一次考虑一组程序而非单个程序
 - 基于概率的方法 —— 基于概率模型查找最有可能的程序



枚举法



自顶向下遍历

- 按语法依次展开
 - Expr
 - $x, y, \text{Expr} + \text{Expr}, \text{Expr} - \text{Expr}, (\text{ite BoolExpr}, \text{Expr}, \text{Expr})$
 - $y, \text{Expr} + \text{Expr}, \text{Expr} - \text{Expr}, (\text{ite BoolExpr}, \text{Expr}, \text{Expr})$
 - $\text{Expr} + \text{Expr}, \text{Expr} - \text{Expr}, (\text{ite BoolExpr}, \text{Expr}, \text{Expr})$
 - $x + \text{Expr}, y + \text{Expr}, \text{Expr} + \text{Expr} + \text{Expr}, \text{Expr} - \text{Expr} + \text{Expr}, (\text{ite BoolExpr}, \text{Expr}, \text{Expr}) + \text{Expr}, \text{Expr} - \text{Expr}, (\text{ite BoolExpr}, \text{Expr}, \text{Expr})$
 - ...



自底向上遍历

- 从小到大组合表达式
 - size=1
 - x, y
 - size=2
 - size=3
 - $x+y, x-y$
 - size=4
 - size=5
 - $x+(x+y), x-(x+y), \dots$
 - size=6
 - $(\text{ite } x \leq y, x, y), \dots$



优化

- 等价性削减
 - 如果等价于一个之前的程序，则删掉
 - $\text{Expr} + x, \cancel{x + \text{Expr}}$
 - 自底向上的枚举中两个程序还需要是从同一个非终结符展开的
- 剪枝
 - 如果所有对应完整程序都不能满足约束，则删掉
 - $\cancel{\text{Ite BoolExpr } x \ x}$



判断程序是否等价

- 通过SMT求解器可以判断
 - 判断 $f(x, y) \neq f'(x, y)$ 是否可以满足
 - 开销较大，不一定划算
- 通过预定义规则判断（常用于自顶向下）
 - 如 $S+x$ 和 $x+S$ 的等价性



判断程序是否等价

- 通过测试判断（又叫可观察等价observational equivalence）
 - 运行所有测试检测 $f = f'$
 - 认为等价的程序有可能不等价
 - 但在CEGIS框架中没有问题，因为我们目标是找到一个满足所有样例的程序而非满足完整规约的程序
 - 用于自底向上，是目前效果最好的等价性削减策略之一
 - 对于不完整程序不能运行测试，所以不能用于自顶向下



剪枝

- 通常用于自顶向下
- 搜索过程中剪枝
 - 语义: `ite BoolExpr x x`
 - 类型: `Expr + substring(Expr, 2)`
 - 大小: 假设AST树的大小（节点数）限定为4，那么 `ite BoolExpr x x`肯定无法满足
- 剪枝的条件
 - 所有可展开的程序都无法满足约束

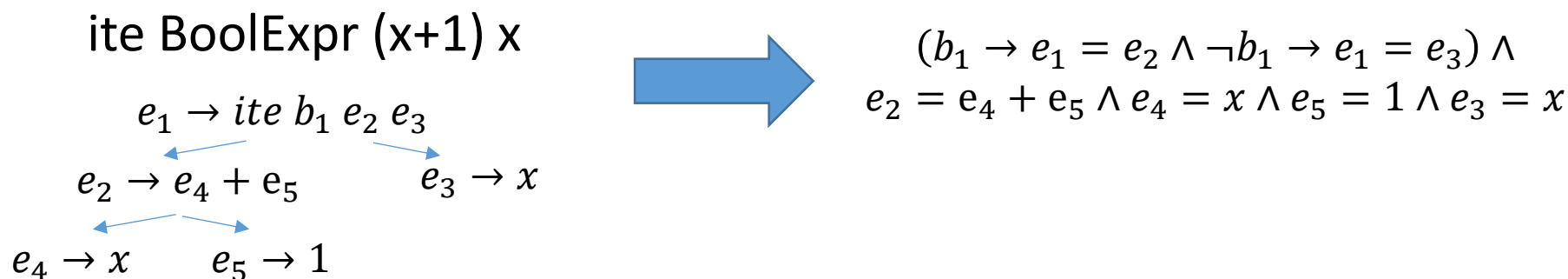


剪枝基本方法：约束求解

- 针对每个组件预定义约束
 - 该约束可以不充分但必须必要

$Expr_1 \rightarrow ite\ BoolExpr\ Expr_2\ Expr_3$	$BoolExpr \rightarrow Expr_1 = Expr_2 \wedge \neg BoolExpr \rightarrow Expr_1 = Expr_3$
$Expr_1 \rightarrow Expr_2 + Expr_3$	$Expr_1 = Expr_2 + Expr_3$
$Expr_1 \rightarrow Expr_2 \% Expr_3$	$Expr_1 \leq Expr_2 \wedge Expr_1 < Expr_3$
$Expr \rightarrow x$	$Expr = x$
$Expr \rightarrow 1$	$Expr = 1$

- 从部分程序生成约束
 - 根据语法树对其中变量进行替换





剪枝基本方法：约束求解

- 从测试生成约束

$$\max2(1,3)=3$$



$$x = 1 \wedge y = 3 \wedge e_1 = 3$$

- 检查约束可满足性
 - 如果不可满足，则剪枝



剪枝的优化

- 剪枝起作用的条件
 - 剪枝的分析时间 $<$ 被去掉程序的分析时间
 - 约束求解的开销通常较大
- 如何快速分析出程序不满足约束?
- 方法1: 预分析
 - 在语法上离线做静态分析
 - 根据静态分析的结果快速在线剪枝
- 方法2: 在线学习
 - 根据冲突学习约束
 - 根据约束快速排除



语法上的静态预分析

- 假设所有约束都是 $\text{Pred}(\text{Prop}(N))$ 的形式
 - N : 非终结符
 - Prop : 以 N 为根节点的子树所具有的属性值
 - Pred : 该属性值所应该满足的谓词
- 如:
 - 语义约束: Prop 为表达式取值
 - 类型约束: Prop 为表达式的可能类型
 - 大小约束: Prop 为表达式的大小
- 通过静态分析获得 Prop 的所有可能取值
 - 要求上近似
- 如果所有可能取值都不能满足 Pred , 则该部分程序可以减掉



语法上静态分析示例：语义

- 抽象域：由0, 1, 2, 3, >3, <0, true, false构成的集合
- 容易定义出抽象域上的计算
- 给定输入输出样例 $x=1, y=2, \max2(x,y)=2$
- 从语法规则产生方程
- $E \rightarrow E+E \mid 0 \mid 1 \mid x \mid \dots$
 - $V[E] = (V[E]+V[E]) \cup \{0\} \cup \{1\} \cup \{1\} \dots$
- 求解方程得到每一个非终结符可能的取值（在开始时做一次）
- 根据当前的部分程序产生计算式

ite BoolExpr x x



$$V[E] = V[x] \cup V[x]$$



语法上静态分析示例：类型

- 抽象域：由Int, String, Boolean构成的集合

- 从语法规则产生方程

- $E \rightarrow E + E \mid 0 \mid 1 \mid x \mid \dots$

- $T[E] = (T[E] + T[E]) \cup \{Int\} \cup \{Int\} \cup \{Int\} \dots$

- 其中

- $$t_1 + t_2 = \begin{cases} \{Int\}, & Int \in t_1 \wedge Int \in t_2 \\ \emptyset, & \text{否则} \end{cases}$$



语法上静态分析示例：大小

- 抽象域：整数
- 从语法规则产生方程
- $E \rightarrow E + E \mid 0 \mid 1 \mid x \mid \dots$
 - $S[E] = \min(2S[E], 1, 1, 1, \dots)$



冲突制导的在线学习

- 之前的剪枝方法主要依赖离线分析
- 在线搜索的过程中如果遇到不满足规约的程序，能否从中学习到更好的剪枝方法？



冯煜
UCSB助理教授



例子：序列操作合成

$$N \rightarrow \emptyset \mid \dots \mid 1\emptyset \mid x_i \mid \text{last}(L) \mid \text{head}(L) \mid \text{sum}(L) \\ \mid \text{maximum}(L) \mid \text{minimum}(L)$$
$$L \rightarrow \text{take}(L, N) \mid \text{filter}(L, T) \mid \text{sort}(L) \mid \text{reverse}(L) \mid x_i$$
$$T \rightarrow \text{geqz} \mid \text{leqz} \mid \text{eqz}$$

输入： $x=[49, 62, 82, 54, 76]$

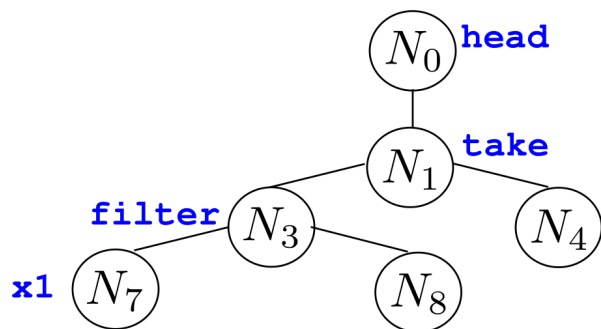
输出： $y=158$



用基本方法剪枝部分程序

Component	Specification
head	$x_1.size > 1 \wedge y.size = 1 \wedge y.max \leq x_1.max$
take	$y.size < x_1.size \wedge y.max \leq x_1.max \wedge$ $x_2 > 0 \wedge x_1.size > x_2$
filter	$y.size < x_1.size \wedge y.max \leq x_1.max$

head(take(filter(x1, T), N)



$$x_1 = [49, 62, 82, 54, 76] \wedge y = 158$$

$$\phi_{N_0} = \underline{y \leq v_1.max} \wedge v_1.size > 1 \wedge y.size = 1$$

$$\phi_{N_1} = \underline{v_1.max \leq v_3.max} \wedge v_1.size < v_3.size \wedge$$

$$v_4 > 0 \wedge v_3.size > v_4$$

$$\phi_{N_3} = v_3.size < v_7.size \wedge \underline{v_3.max \leq v_7.max}$$

$$\phi_{N_7} = \underline{x_1 = v_7}$$

加下划线的为极小矛盾集 ψ

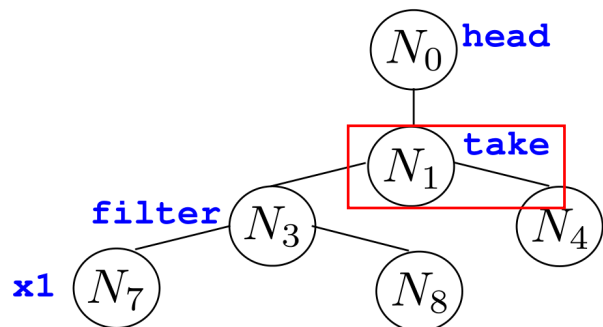


从冲突中学习

- 如果
 - 从新程序P导出的规约 \Rightarrow 该矛盾集 ψ
- 则
 - P也是无效程序
- 用约束求解器判断上述条件不一定比直接判断新程序是否满足规约快
 - 如何快速排除部分程序?



对当前冲突等价



$$\phi_{N_0} = \underline{y \leq v_1.max} \wedge v_1.size > 1 \wedge y.size = 1$$

$$\phi_{N_1} = \underline{v_1.max \leq v_3.max} \wedge v_1.size < v_3.size \wedge v_4 > 0 \wedge v_3.size > v_4$$

$$\phi_{N_3} = v_3.size < v_7.size \wedge \underline{v_3.max \leq v_7.max}$$

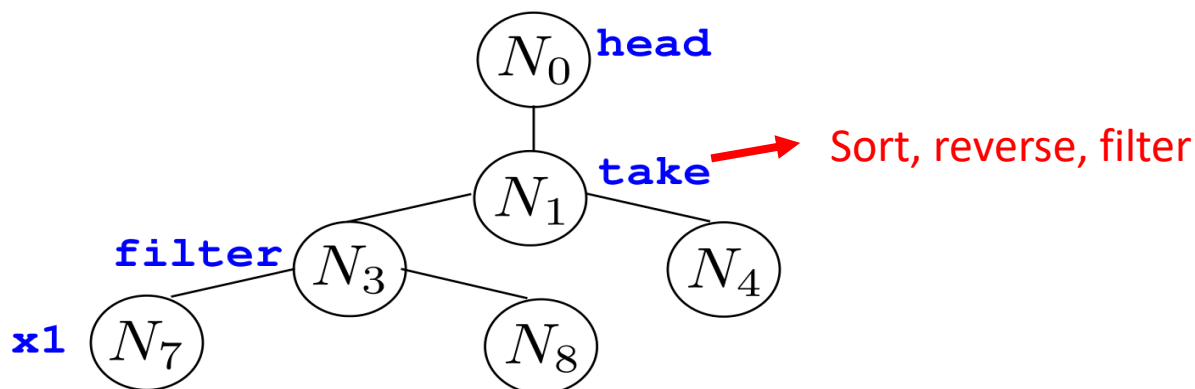
$$\phi_{N_7} = \underline{x_1 = v_7}$$

- N_1 位置的take因为 $y.max \leq x_1.max$ 而冲突
- 任意组件f, 如果
 - f 的规约 $\Rightarrow y.max \leq x_1.max$
- 则
 - f 在 N_1 位置和当前冲突等价
- 因为只涉及到组件的固定规约, 可以用SMT solver离线验证



排除程序

- 遍历组件可以发现，sort, reverse, filter都在 N_1 位置和当前冲突等价



- 所有将 N_1 替换这些组件的程序都无效
- 考虑其他位置的等价以及这些等价关系的组合，能排除较多等价程序。

合一化程序合成STUN

Synthesis through Unification



- 针对If表达式的特殊合成方法
 - STUN系列求解器Eusolver, Euphony, PolyGen在SyGuS比赛的CLIA Track中表现最优
- 假设合成的程序具有如下形式
 - **if** boolExpr₁ **then** expr₁
 else if boolExpr₂ **then** expr₂
 ...
 - **else** expr_n
 - Expr_i是符合上面形式的表达式或不带if的原子表达式



Rajeev Alur
Upenn教授
STUN发明人，领导
Eusolver和Euphony开发



基本流程(1/3)

- 首先枚举一组boolExpr和expr
 - 对于expr，记录所覆盖的example
 - 基于boolExpr，记录将example分成的两个组
 - 从小到大枚举，按概率枚举或者用专门的算法产生
- 假设枚举到x, y, 1三个表达式和x=3, x<y两个条件

x	y	ret	covering expr	boolExpr	
				x=3	x<y
1	2	2	y	false	true
3	3	3	x,y	true	false
5	2	5	x	false	false



基本流程(2/3)

- 选择覆盖所有例子的expr集合
 - 选择标准：限制expr大小的最大值、控制总大小等
- 在这里选择x和y

x	y	ret	covering expr	boolExpr	
				x=3	x<y
1	2	2	y	false	true
3	3	3	x,y	true	false
5	2	5	x	false	false



基本流程(3/3)

- 用决策树算法选择boolExpr构造最终程序
 - 选择boolExpr对example分类，让每个分类都能被一个所选expr覆盖
 - 选择标准：减少信息熵，或者控制总大小
- 选择 $x < y$ 可以把原来的example分成两类，第一类包含第一个example，被x覆盖，第二类包含后两个example，被y覆盖
- 得到if($x < y$) then x else y

x	y	ret	covering expr	boolExpr	
				x=3	x<y
1	2	2	y	false	true
3	3	3	x,y	true	false
5	2	5	x	false	false



吉如一
北京大学博士生
奥卡姆程序合成
和PolyGen提出人

奥卡姆程序合成

- 如何使用较少样例就得到正确程序？
 - 奥卡姆剃刀原则：如非必要，勿增实体
 - 映射到程序合成：合成尽量小的程序
- 严格要求最小程序导致合成算法效率低下
- 奥卡姆程序合成：高概率合成大小不超过最小程序多项式倍的程序
- 奥卡姆程序合成的用处
 - 提升CEGIS框架下的合成速度
 - 对于难以验证的规约，提供较高的概率错误保障



Eusolver/Euphony 不是奥卡姆程序合成

- Eusolver/Euphony控制了expr和boolExpr的大小，但没有控制他们的数量
- Eusolver/Euphony从小到大枚举expr
 - 可能导致用大量的小expr来覆盖一组样例
- 传统决策树算法是为模糊分类设计，用在程序合成上可能生成大量无用boolExpr
 - 假设三个表达式a, b, c可以覆盖所有样例且没有重叠
 - 某个boolExpr可以把样例分成如下两类
 - 类1: 98个样例可以被a覆盖，1个样例被b覆盖，1个样例被c覆盖
 - 类2: 98个样例被b覆盖，1个样例被a覆盖，1个样例被c覆盖
 - 该条件有很好的信息熵增益
 - 但对生成程序几乎没有帮助，因为每类还需要继续区分三个表达式



PolyGen

- 通过迭代加深算法控制expr的个数
 - 从一个较小的expr个数上限开始，如果合成失败，则给上限增加一个常量
- 通过生成决策列表而不是决策树来控制boolExpr的个数
 - **if** boolExpr₁ **then** expr₁
else if boolExpr₂ **then** expr₂
...
else expr_n
- 在SyGuS的CLIA Track上，PolyGen的速度达到Eusolver和Euphony的6-15倍
- 详见PolyGen论文：
 - Ruyi Ji, Jingtao Xia, Yingfei Xiong, Zhenjiang Hu. Generalizable Synthesis Through Unification. OOPSLA'21: Object Oriented Programming Languages, Systems and Applications, October 2021.



参考资料

- Armando Solar-Lezama. Lecture Notes on Program Synthesis.
<https://people.csail.mit.edu/asolar/SynthesisCourse/TOC.htm>
- Syntax-Guided Synthesis. R. Alur, R. Bodik, G. Juniwal, P. Madusudan, M. Martin, M. Raghothman, S. Seshia, R. Singh, A. Solar-Lezama, E. Torlak and A. Udupa. In 13th International Conference on Formal Methods in Computer-Aided Design, 2013.
- Sumit Gulwani, Oleksandr Polozov, Rishabh Singh: Program Synthesis. Foundations and Trends in Programming Languages 4(1-2): 1-119 (2017)
- Ruyi Ji, Chaozhe Kong, Yingfei Xiong, Zhenjiang Hu. Improving Oracle-Guided Inductive Synthesis by Efficient Question Selection. OOPSLA'23: Object-Oriented Programming, Systems, Languages and Applications, October 2023.
- Feng Y , Martins R , Bastani O , et al. Program Synthesis using Conflict-Driven Learning[J]. PLDI 2018.
- Ruyi Ji, Jingtao Xia, Yingfei Xiong, Zhenjiang Hu. Generalizable Synthesis Through Unification. OOPSLA'21: Object Oriented Programming Languages, Systems and Applications, October 2021.