



软件分析

程序合成：枚举

熊英飞
北京大学

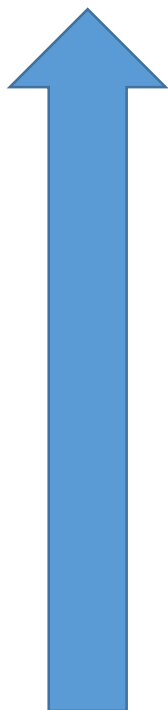


外祖母可以编程吗？

- 程序设计语言的发展历史就是提高抽象级别



抽象级别



外祖母编程语言？

Haskell (1990), Prolog (1972)

Java

C

Assembly



为什么外祖母还不能编程？

- 程序设计语言默认保证很多属性
 - 类型正确的程序一定能通过编译
 - 通过编译的程序有清晰定义的语义
- 很难再进一步提升抽象级别





程序合成——外祖母的希望

- 从规约中自动生成程序
 - 规约可能是模糊的
 - 生成是不保证成功的



“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



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

典型应用——Data Wrangling

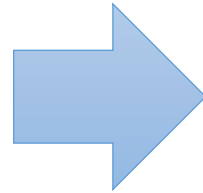


	A	B
1	Email	Column 2
2	Nancy.FreeHafer@fourthcoffee.com	nancy freehafer
3	Andrew.Cencici@northwindtraders.com	andrew cencici
4	Jan.Kotas@litwareinc.com	jan kotas
5	Mariya.Sergienko@gradicdesigninstitute.com	mariya sergienko
6	Steven.Thorpe@northwindtraders.com	steven thorpe
7	Michael.Neipper@northwindtraders.com	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

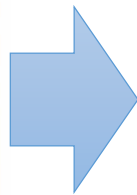


```
i=round(i);
```



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

典型应用-自动编写重复程序



```
class AcidicSwampOoze(MinionCard):  
    def __init__(self):  
        super().__init__("Acidic Swamp Ooze", 2,  
                          CHARACTER_CLASS.ALL, CARD_RARITY.COMMON,  
                          battlecry=Battlecry(Destroy(),  
                                              WeaponSelector(EnemyPlayer())))  
  
    def create_minion(self, player):  
        return Minion(3, 2)
```




典型应用-缺陷修复

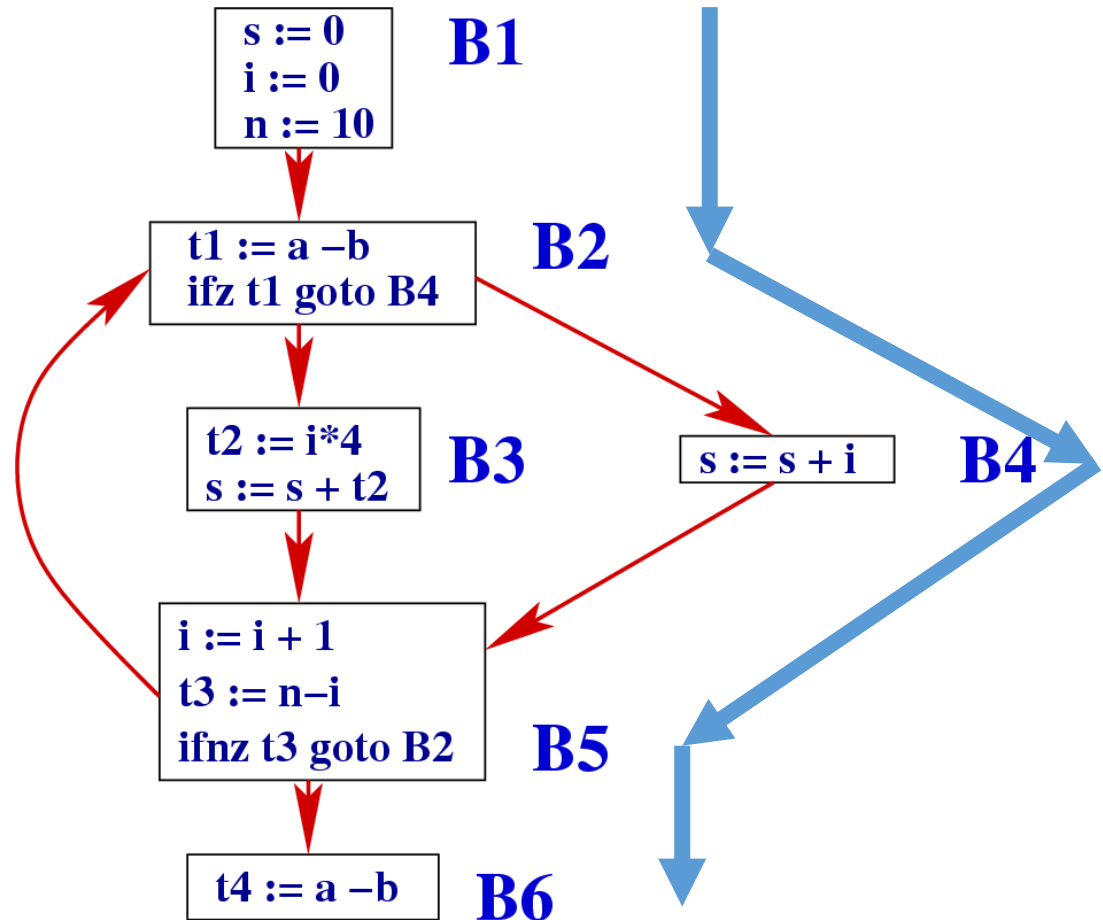
```
/** Compute the maximum of two values
 * @param a first value
 * @param b second value
 * @return b if a is lesser or equal to b, a otherwise
 */
public static int max(final int a, final int b) {
    return (a <= b) ? a : b;
}
```

合成出新的表达式来替换掉旧的



典型应用-生成测试

合成单元测试
来覆盖某路径





典型应用-加速程序分析

SMT Solver

```
Apply Tactic 1
If formula is long
  Apply Tactic 2
Else
  Apply Tactic 3
```

策略

针对一组问题合成最佳策略



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
 - $prog \in Prog \wedge prog \vdash Spec$
- 局限性:
 - 当规约是模糊或者不完整的时候, 正确性就完全无保障了



例子：max问题

- 语法：

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

- 规约：

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

- 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：Synth-Lib
 - <http://sygus.seas.upenn.edu/files/SyGuS-IF.pdf>



Sync-Lib: 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。



Sync-Lib: 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



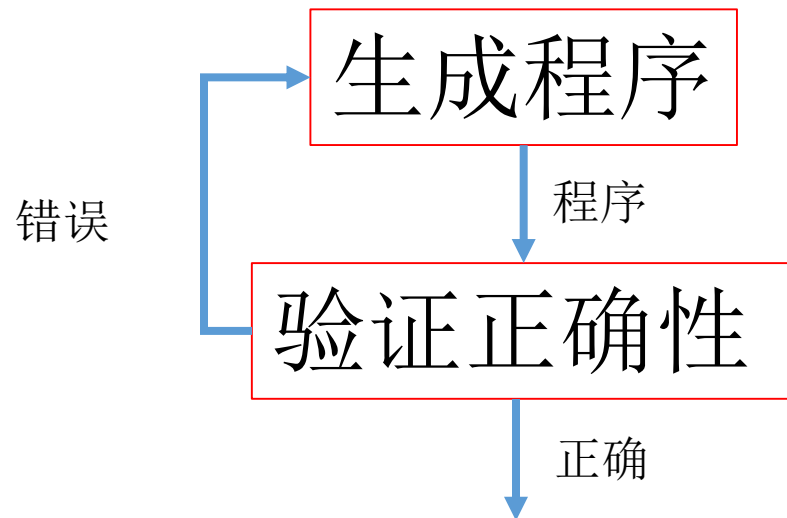
感谢曾沐焱、刘鑫远、吉如一同学准备课程项目！

课程项目2

- 编写程序求解SyGuS问题
- 每小组提交：
 - 一个SyGuS求解器
 - 一个测试样例，至少用自己的求解器2分钟可以解出
- 组队要求：2-3名同学一队，队友须和上一项目不同
- 测试限制：只允许采用样例中出现过的操作符
- 截止日期：12月19日提交，12月21日报告
- 程序包包括：
 - 完整的测试环境，其中题目分为CLIA和BV两部分
 - CLIA的规约为逻辑约束，测试包包含全部题目
 - BV的规约为输入输出样例，其中部分仅在测试时使用，考察程序的泛化能力。测试包包含60%的题目。
 - 最基本的自顶向下枚举solver（只能解1道题）
 - 最基本的solver修改两行就至少能解出3道题
- 评分：根据解出来的样例个数评分（每个时限5分钟）
 - 能解出3道题获得60分

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?



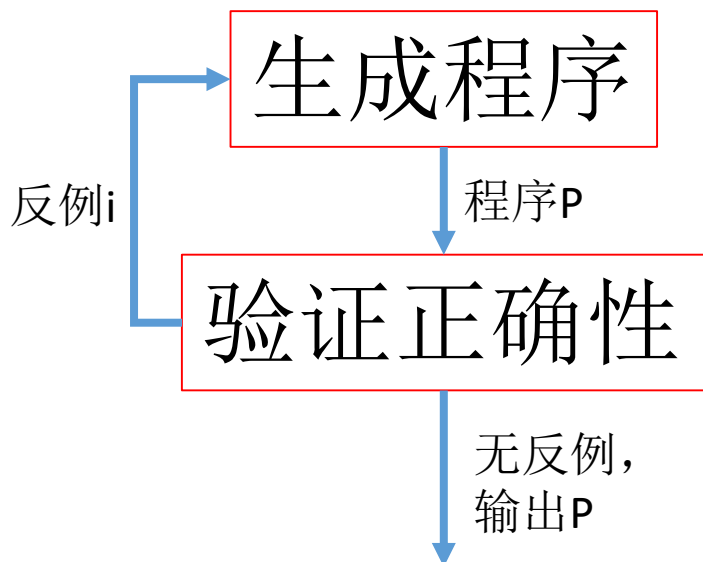
如何验证程序的正确性?

- 采用本课程学习的技术
 - 抽象解释
 - 符号执行

- 目前大多数程序合成技术都只处理表达式
 - 可直接转成约束让SMT求解
 - Synth-lib直接提供支持



CEGIS——基于反例的优化



- 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输入保存
- 验证的时候首先采用测试验证



Armando Solar-Lezama
麻省理工大学教授

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
 - $x, y, \text{Expr}+\text{Expr}, \text{Expr}-\text{Expr}, (\text{ite BoolExpr}, \text{Expr}, \text{Expr})$
 - $y, \text{Expr}+\text{Expr}, \text{Expr}-\text{Expr}, (\text{ite BoolExpr}, \text{Expr}, \text{Expr})$
 - $\text{Expr}+\text{Expr}, \text{Expr}-\text{Expr}, (\text{ite BoolExpr}, \text{Expr}, \text{Expr})$
 - $x+\text{Expr}, y+\text{Expr}, \text{Expr}+\text{Expr}+\text{Expr}, \text{Expr}-\text{Expr}+\text{Expr}, (\text{ite BoolExpr}, \text{Expr}, \text{Expr})+\text{Expr}, \text{Expr}-\text{Expr}, (\text{ite BoolExpr}, \text{Expr}, \text{Expr})$
 - ...



自顶向下遍历

```
function ENUMTOPDOWNSEARCH(grammar  $G$ , spec  $\phi$ )  
   $\tilde{P} \leftarrow [S]$  // An ordered list of partial derivations in  $G$   
   $\tilde{P}_v \leftarrow \{S\}$  // A set of programs  
  while  $\tilde{P} \neq \emptyset$  do  
     $p \leftarrow \text{REMOVEFIRST}(\tilde{P})$   
    if  $\phi(p)$  then // Specification  $\phi$  is satisfied  
      return  $p$   
     $\tilde{\alpha} \leftarrow \text{NONTERMINALS}(p)$   
    foreach  $\alpha \in \text{RANKNONTERMINALS}(\tilde{\alpha}, \phi)$  do  
       $\tilde{\beta} \leftarrow \{\beta \mid (\alpha, \beta) \in R\}$   
      foreach  $\beta \in \text{RANKPRODUCTIONRULE}(\tilde{\beta}, \phi)$  do  
         $p' \leftarrow p[\alpha \rightarrow \beta]$   
        if  $\neg \text{SUBSUMED}(p', \tilde{P}_v, \phi)$  then  
           $\tilde{P}.\text{INSERT}(p')$   
           $\tilde{P}_v \leftarrow \tilde{P}_v \cup p'$ 
```

对非终结符排序

对产生式排序

检查程序是否和之前的等价，比如 $x+S$ 和 $S+x$ 为什么 ϕ 也是参数？



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
 - $(\text{ite } x \leq y, x, y), \dots$



自底向上遍历

function ENUMBOTTOMUPSEARCH(grammar G , spec ϕ)

$\tilde{E} \leftarrow \{\Phi\}$ // Set of expressions in G

progSize $\leftarrow 1$

while True **do**

$\tilde{C} \leftarrow$ **ENUMERATEEXPRS**($G, \tilde{E}, \text{progSize}$)

foreach $c \in \tilde{C}$ **do**

if $\phi(c)$ **then** // Specification ϕ is satisfied

return c

if $\neg \exists e \in \tilde{E} :$ **EQUIV**(e, c, ϕ) **then**

$\tilde{E}.$ **INSERT**(c)

progSize \leftarrow progSize + 1

根据语法 G ，用 \tilde{E} 组合出大小等于progSize的所有表达式

判断 e 和 c 是否等价。



优化

- 等价性削减
 - 如果等价于一个之前的程序，则删掉
 - $\text{Expr}+x, \text{x+Expr}$
- 剪枝
 - 如果所有对应完整程序都不能满足约束，则删掉
 - $\text{Ite BoolExpr } x \ x$



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
- 通过测试判断（又叫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
 - 但一般没有问题，因为我们目标是找到一个满足所有样例的程序而非找到所有这样的程序
 - 对于不完整程序不能运行测试
- 通过预定义规则判断
 - 如 $S+x$ 和 $x+S$ 的等价性




剪枝

- 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

- 从部分程序中生成约束
 - 针对每个组件预定义约束
 - 该约束可以不充分但必须必要
 - 根据语法树将约束连接在一起

Ite BoolExpr x x  (declare-fun boolExpr () Int)
(declare-fun max2 ((x Int) (y Int)) Int
(ite boolExpr x x))

- 从测试中生成约束

max2(1,2)=2  (assert (= (max2 1 2) 2))
(check-sat)



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 $x=1, y=0, \max2(x,y)=1$
- 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？
- 复习：SAT
 - 什么是DPLL？
 - 什么是CDCL？



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： $x=[49, 62, 82, 54, 76]$

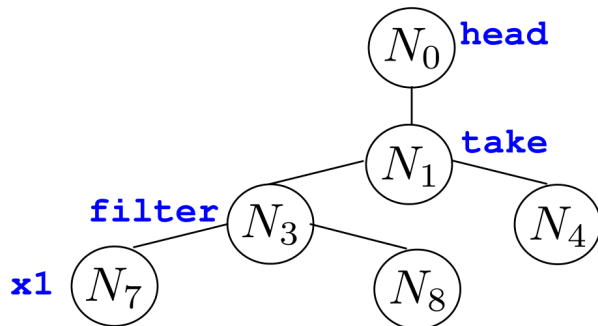
输出： $y=158$



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

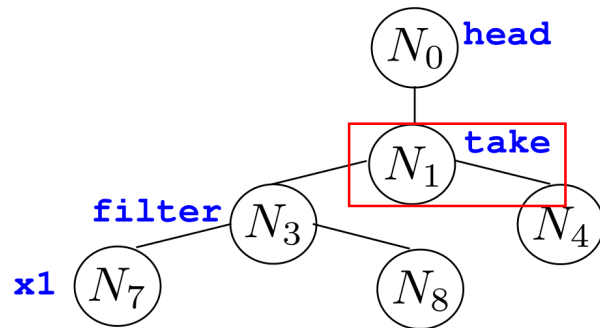


从冲突中学习

- 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



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

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

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

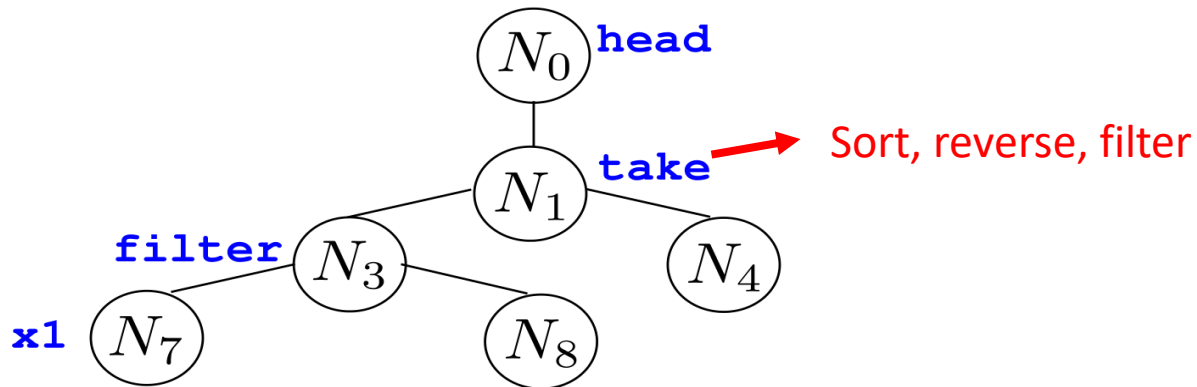
$$\phi_{N_7} = \underline{x_1 = v_7}$$

- 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



吉如一
北京大学博士生
PolyGen作者



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	x	false	true
3	3	3	x,y	true	false
5	2	5	y	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	x	false	true
3	3	3	x,y	true	false
5	2	5	y	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	x	false	true
3	3	3	x,y	true	false
5	2	5	y	false	false



决策树 vs 程序合成

- 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很大的程序
 - 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
- 解决方案: 设计新算法控制条件总大小
 - 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)
- 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.