软件分析

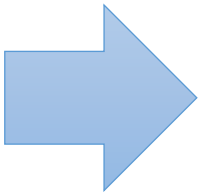
程序合成：枚举

熊英飞
北京大学

动机——超优化

针对不同的函数，常常有特定的优化模式。
模式数量多，难以在编译器中预置。

```
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：

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

- 规约：

$$
\begin{aligned}
\forall x, y : \mathbb{Z}, \quad & max_2(x, y) \geq x \wedge max_2(x, y) \geq y \\
& \wedge (max_2(x, y) = x \vee max_2(x, y) = y)
\end{aligned}
$$

- 期望答案： ite (x <= y) y x

程序合成是软件分析问题

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

```
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)

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

(check-synth)
```

约束表示方式和SMTLib一致

期望输出

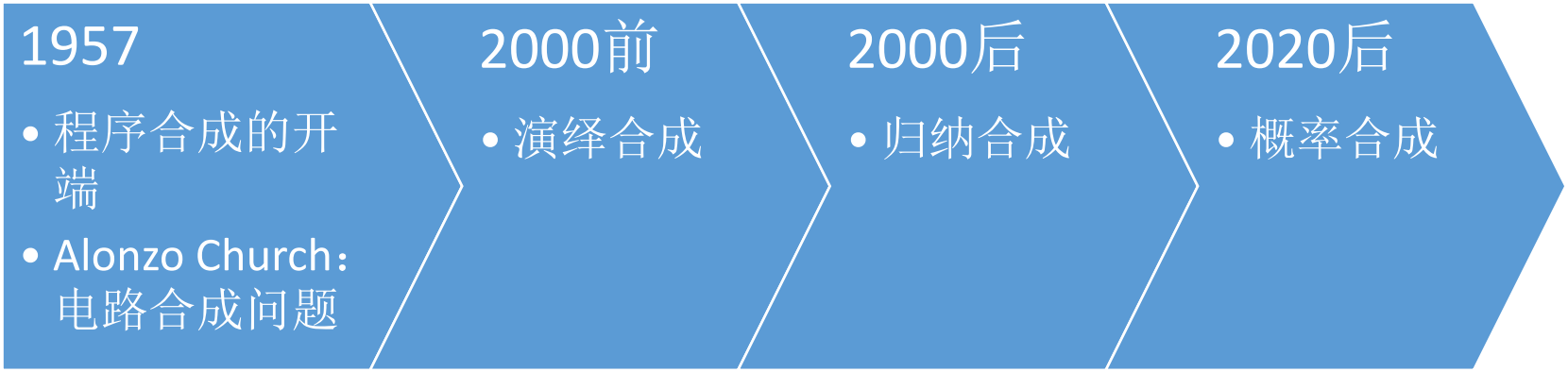
输出：

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

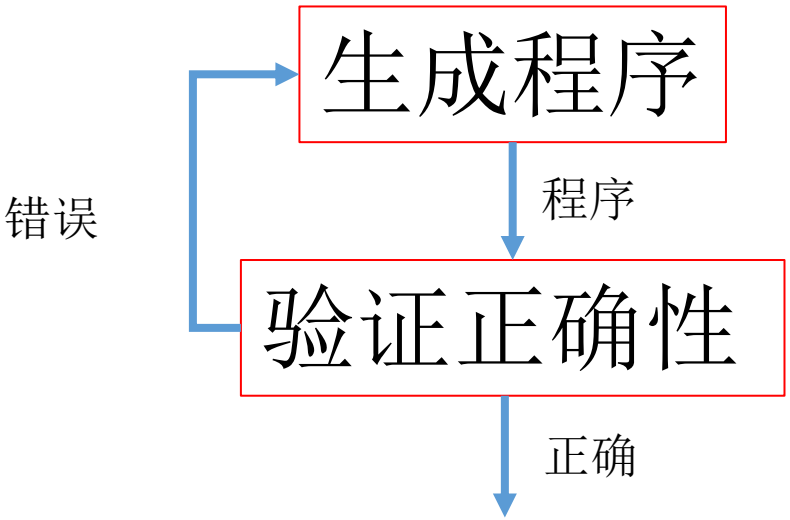
输出必须：

- 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
2020后
• 概率合成

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？

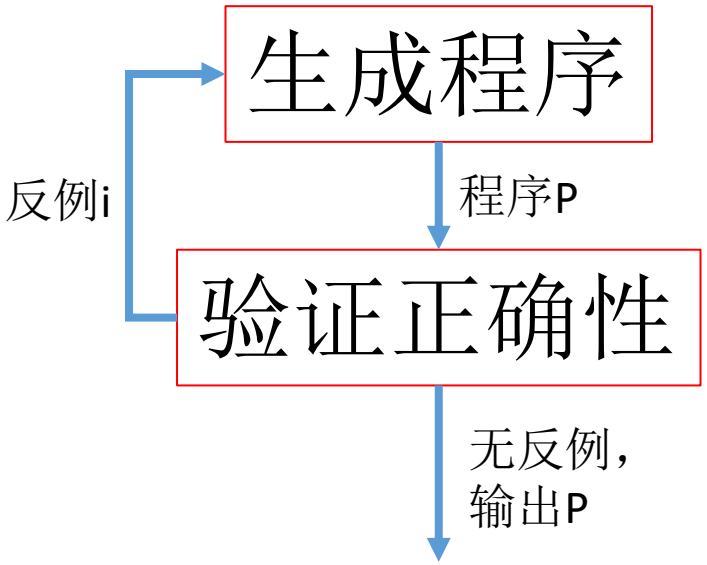
Q2:如何验证程序的正确性？

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

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, Expr+Expr, Expr-Expr, (ite BoolExpr, Expr, Expr)
  - y, Expr+Expr, Expr-Expr, (ite BoolExpr, Expr, Expr)
  - Expr+Expr, Expr-Expr, (ite BoolExpr, Expr, Expr)
  - x+Expr, y+Expr, Expr+Expr+Expr, Expr-Expr+Expr, (ite BoolExpr, Expr, Expr)+Expr, Expr-Expr, (ite BoolExpr, Expr, 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), …
  - size=6
    - (ite x<=y, x, y), …

优化

- 等价性削减
  - 如果等价于一个之前的程序，则删掉
  - Expr+x, ~~x+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
  - ~~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

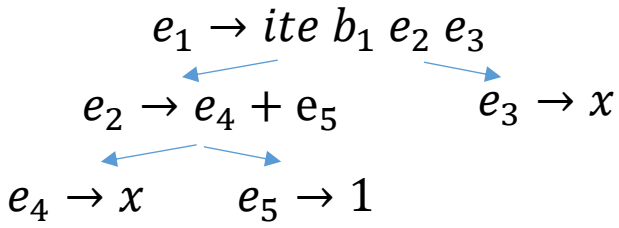
剪枝基本方法：约束求解

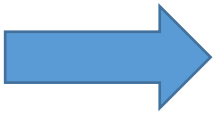
- 针对每个组件预定义约束
  - 该约束可以不充分但必须必要

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

- 从部分程序生成约束
  - 根据语法树对其中变量进行替换

ite BoolExpr (x+1) x





$$(b_1 \to e_1 = e_2 \wedge \neg b_1 \to e_1 = e_3) \wedge e_2 = e_4 + e_5 \wedge e_4 = x \wedge e_5 = 1 \wedge e_3 = x$$

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

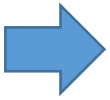
语法上的静态预分析

- 假设所有约束都是Pred(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->E+E | 0 | 1 | x |…
  - V[E]=(V[E]+V[E]) ∪ {0} ∪ {1} ∪ {1} …
- 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->E+E | 0 | 1 | x |…
  - T[E]=(T[E]+T[E]) ∪ {Int} ∪ {Int} ∪ {Int} …
- 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->E+E | 0 | 1 | x |…
  - S[E]=min(2S[E], 1, 1, 1, …)

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

$N \rightarrow$ 0 | ... | 10 | x_i | last(L) | head(L) |sum(L)
| maximum(L) | minimum(L)

$L \rightarrow$ take(L, N) | filter(L, T)| sort(L) | reverse(L) | x_i

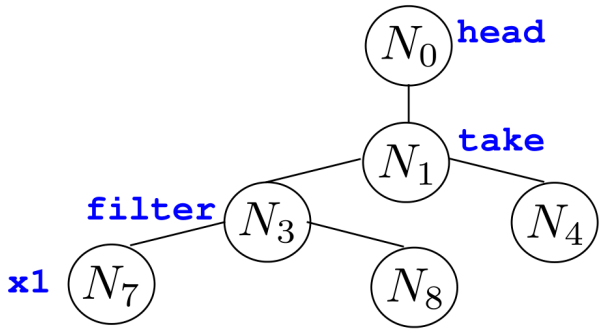
$T \rightarrow$ geqz | leqz | eqz

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)



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

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

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

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

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

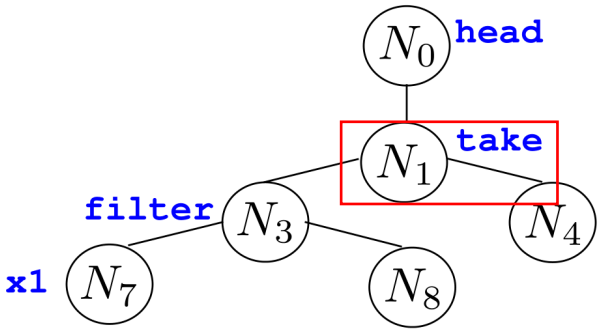
加下划线的为极小矛盾集ψ

从冲突中学习

- 如果
  - 从新程序P导出的规约 ⇒ 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



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

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

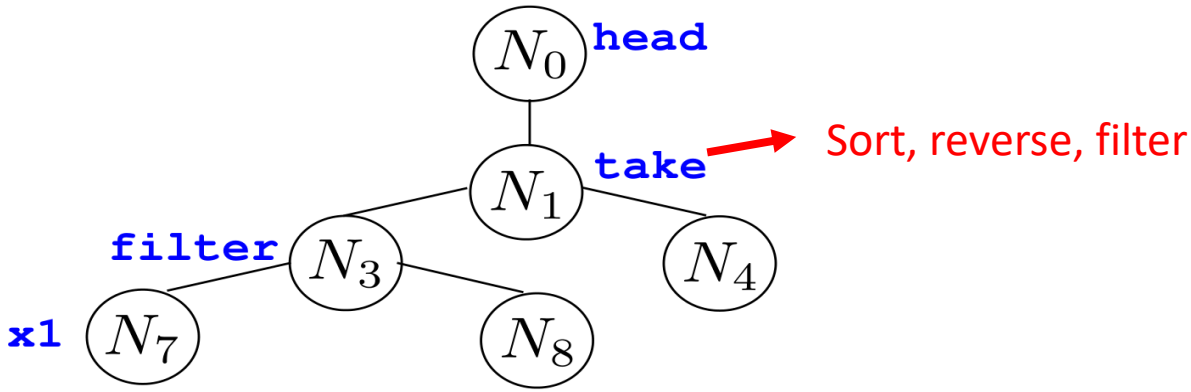
$$\phi_{N_3} = v_3.size < v_7.size \ \wedge \ v_3.max \le v_7.max$$

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

- N_1位置的take因为$y.max \le x_1.max$而冲突
- 任意组件f，如果
  - f的规约⇒ $y.max \le x_1.max$
- 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。

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.