软件分析

程序综合：枚举

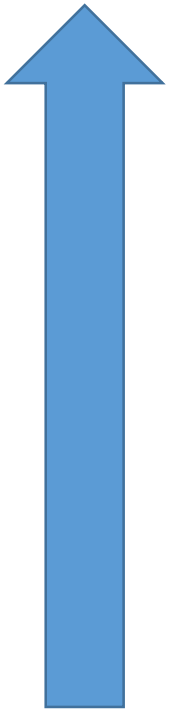
熊英飞
北京大学

软件分析课程内容

- 基于抽象解释的分析
  - 数据流分析
  - 过程间分析
  - 指向分析
  - 控制流分析
  - 抽象解释理论
  - 符号抽象
- 基于约束求解的分析
  - SAT求解算法
  - SMT求解算法
  - 符号执行
  - 霍尔逻辑和谓词变换
- 分析技术应用
  - 程序综合
  - 错误定位
  - 错误修复

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



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

程序综合——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 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

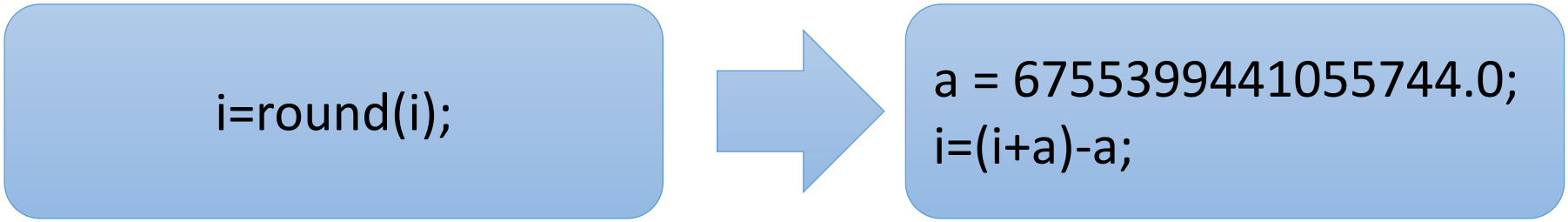
中国软件先驱

程序综合的历史

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



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)
```

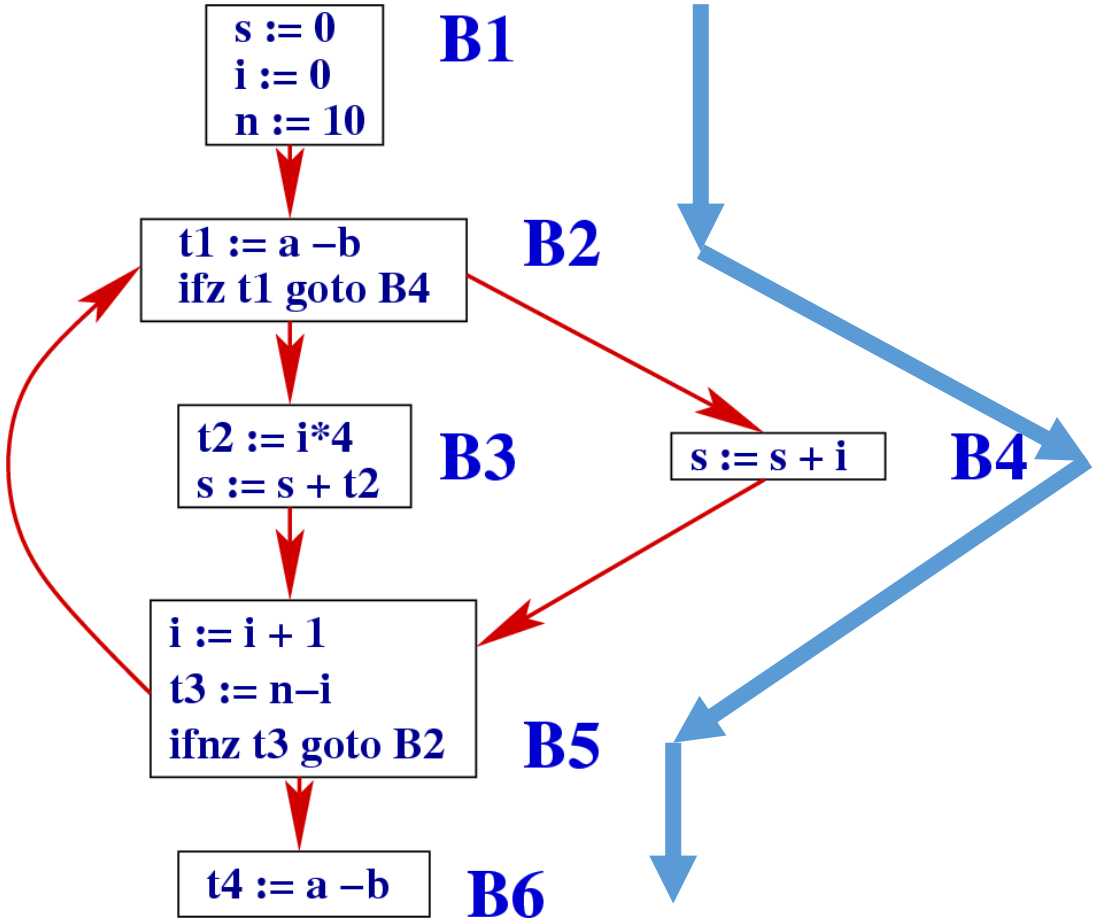
典型应用–缺陷修复

```
/** 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;
}
```

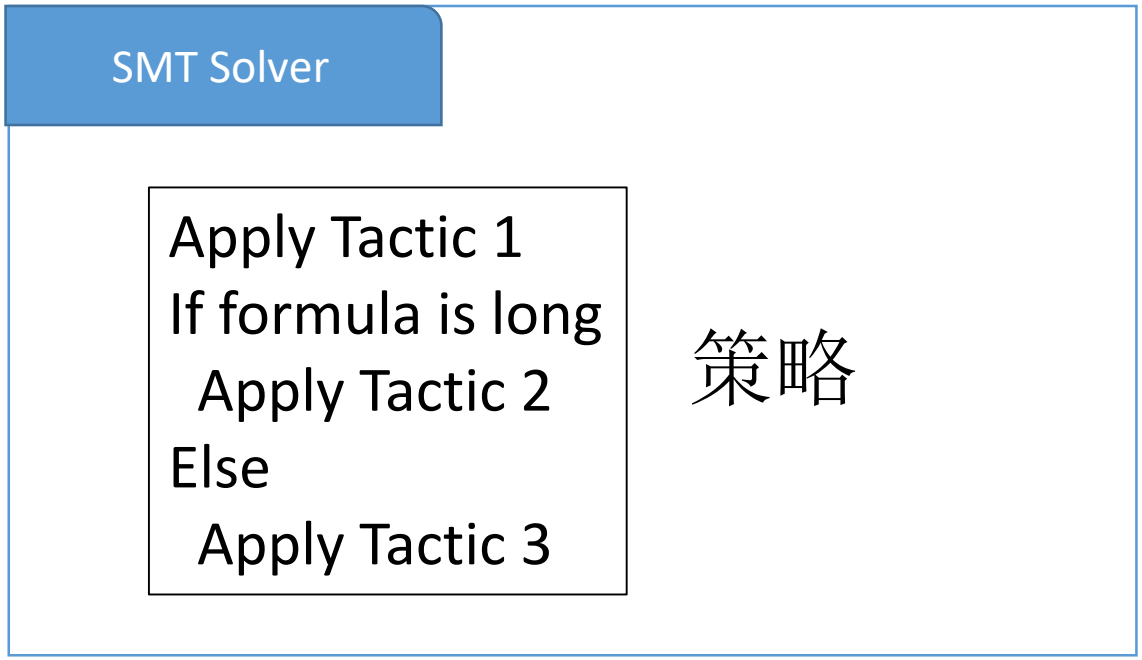
综合出新的表达式来替换掉旧的

典型应用–缺陷修复

综合单元测试
来覆盖某路径



典型应用-加速程序分析



针对一组问题综合最佳策略

程序综合定义

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

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

- 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

程序综合是软件分析问题

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

```
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:
程序综合问题的标准化

- 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)

(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

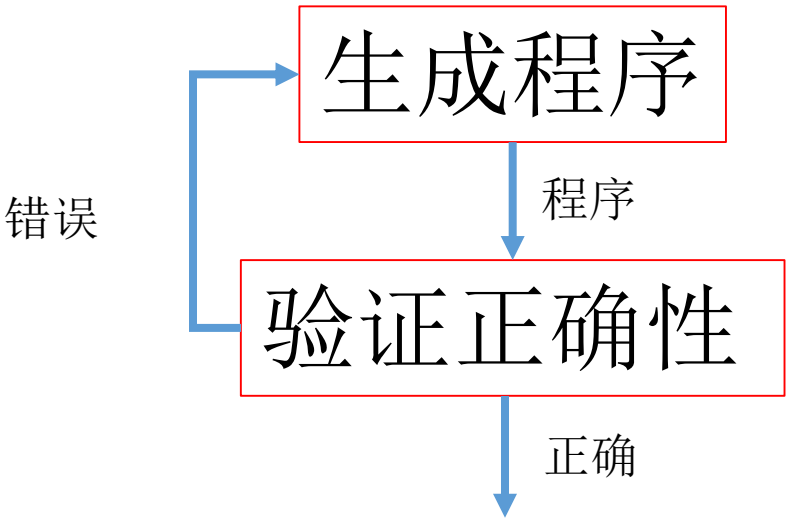
输出：

```
(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

程序综合作为搜索问题



Q1:如何产生下一个被搜索的程序？

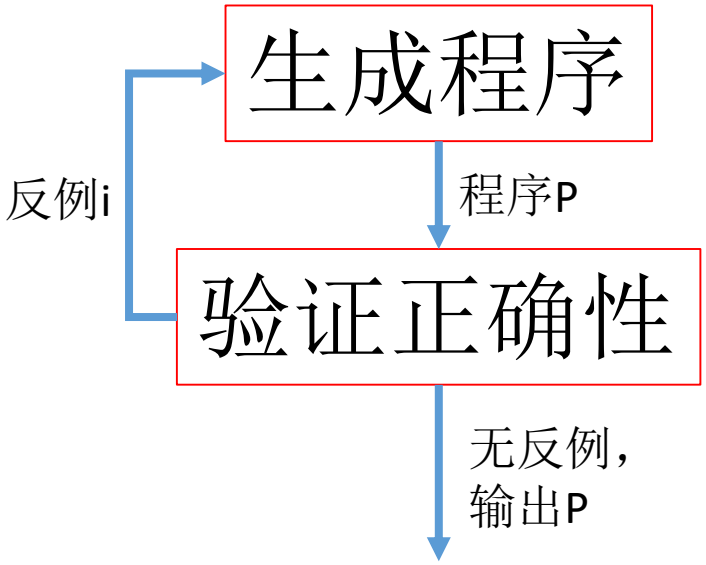
Q2:如何验证程序的正确性？

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

- 目前大多数程序综合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

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
  - 基于表示的方法——一次考虑一组程序而非单个程序
  - 约束求解法 —— 将程序搜索问题整体转成约束求解问题

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)
  - …

自顶向下遍历


```
function EnumTopDownSearch(grammar G, spec φ)
    P̃ ← [S] // An ordered list of partial derivations in G
    P̃_v ← {S} // A set of programs
    while P̃ ≠ ∅ do
        p ← RemoveFirst(P̃)
        if φ(p) then // Specification φ is satisfied
            return p
        α̃ ← NonTerminals(p)
        foreach α ∈ RankNonTerminals(α̃, φ) do
            β̃ ← {β|(α, β) ∈ R}
            foreach β ∈ RankProductionRule(β̃, φ) do
                p′ ← p[α → β]
                if ¬Subsumed(p′, P̃_v, φ) then
                    P̃.Insert(p′)
                    P̃_v ← P̃_v ∪ 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), …
  - size=6
    - (ite x<=y, x, y), …

自底向上遍历


```
function EnumBottomUpSearch(grammar G, spec φ)
    Ẽ ← {Φ} // Set of expressions in G
    progSize ← 1
    while True do
        C̃ ← EnumerateExprs(G, Ẽ, progSize)
        foreach c ∈ C̃ do
            if φ(c) then // Specification φ is satisfied
                return c
            if ¬∃ e ∈ Ẽ : Equiv(e, c, φ) then
                Ẽ.Insert(c)
        progSize ← 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。

双向搜索

- 自底向上遍历可以看做是从输入开始搜索
- 自顶向下遍历可以看做是从输出开始搜索
- 也可以从输入输出同时开始搜索
- 要求能计算最强后条件或者最弱前条件
- 通常用于pipeline程序或者系统状态固定的程序
  - 如：汇编语言的合成
  - Phitchaya Mangpo Phothilimthana, Aditya Thakur, Rastislav Bodík, Dinakar Dhurjati: Scaling up Superoptimization. ASPLOS 2016. 297-310

双向搜索

function BIDIRECTIONALSEARCH(grammar G, spec $\phi \equiv (\phi_{\texttt{pre}}, \phi_{\texttt{post}})$)
  $\widetilde{F} \leftarrow \phi$ // Set of expressions from Forward search
  $\widetilde{B} \leftarrow \phi$ // Set of expressions from Backward search
  progSize $\leftarrow 1$
  while $\neg \exists f \in \widetilde{F}, b \in \widetilde{B}$: MATCHSTATE(f, b) **do**
    $\widetilde{F} \leftarrow$ ENUMFORWARDEXPRS$(G, \widetilde{F}, \phi_{\texttt{pre}}, \texttt{progSize})$
    $\widetilde{B} \leftarrow$ ENUMBACKWARDEXPRS$(G, \widetilde{B}, \phi_{\texttt{post}}, \texttt{progSize})$
    progSize $\leftarrow$ progSize $+ 1$
  $p \leftarrow f \oplus b$, where $\exists f \in \widetilde{F}, b \in \widetilde{B}$: MATCHSTATE(f, b)
  return p

优化

- 等价性削减
  - 如果等价于一个之前的程序，则停止展开。
  - Expr+x, ~~x+Expr~~
- 剪枝
  - 如果所有对应完整程序都不能满足约束，则停止展开
  - ~~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
  - 运行所有测试检测 $f = f'$
  - 并不能保证结果的正确性
  - 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

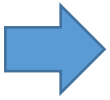
语法上的静态预分析

- 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=0, max2(x,y)=1
- 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

- 复习：SAT
  - 什么是DPLL?
  - 什么是CDCL?

例子：序列操作合成

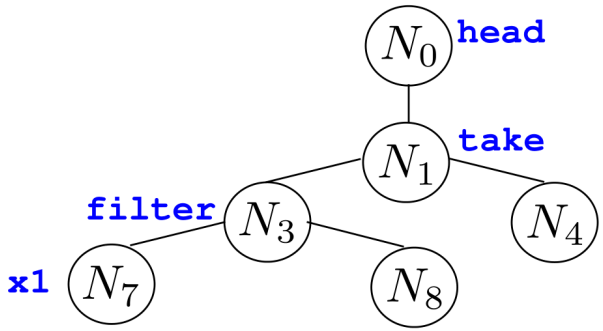
$$N \to \texttt{0} \mid \ldots \mid \texttt{10} \mid x_i \mid \texttt{last}(L) \mid \texttt{head}(L) \mid \texttt{sum}(L)$$
$$\mid \texttt{maximum}(L) \mid \texttt{minimum}(L)$$
$$L \to \texttt{take}(L, N) \mid \texttt{filter}(L, T) \mid \texttt{sort}(L) \mid \texttt{reverse}(L) \mid x_i$$
$$T \to \texttt{geqz} \mid \texttt{leqz} \mid \texttt{eqz}$$

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)



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

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

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

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

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

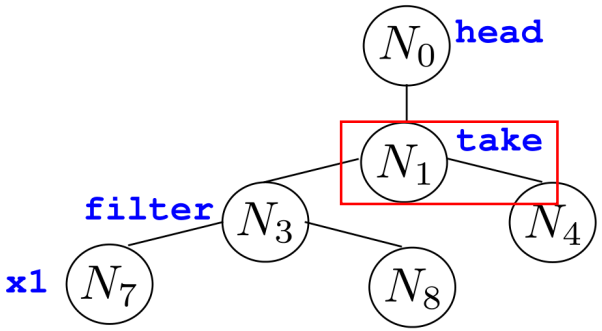
加下划线的为极小矛盾集ψ

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



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

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

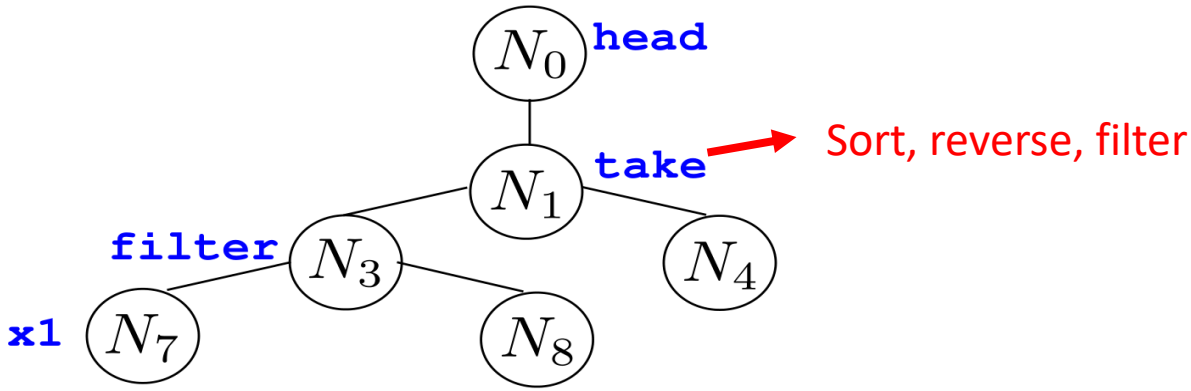
$$\phi_{N_3} = v_3.size < v_7.size \ \wedge \ v_3.max \le v_7.max$$

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

- 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。

课程项目2

感谢曾沐焓刘鑫远同学准备课程项目！

- 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
- 限制：只考虑基础算术和逻辑表达式
- 截止日期：1月3日提交，1月6日报告
- 程序包包括：
  - 完整的测试环境（含所有官方测试用例）
  - 基本的solver
- 评分：根据解出来的样例个数评分（每个时限5分钟）

程序包所带Solver

- 实现了一个比较傻的自顶向下宽度优先搜索
  - 时限内能解出来一个问题：three.sl
- 只需要改动两行就能变成普通的自顶向下搜索
  - 可以在10秒的时限解出三个问题

参考资料

- 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.
- Yingfei Xiong, Bo Wang, Guirong Fu, Linfei Zang. Learning to Synthesize. GI'18: Genetic Improvment Workshop, May 2018.