



软件分析

程序综合：优化

熊英飞
北京大学



基于空间表示的合成



例子：化简的max问题

- 语法：

```
Expr ::= x | y
      | 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： $\text{ite}(x \leq y) y x$



自顶向下遍历

- 按语法依次展开
 - S
 - x, y, S+S, if(B, S, S)
 - y, S+S, if(B, S, S)
 - S+S, if(B, S, S)
 - x+S, y+S, S+S+S, if(B, S, S)+S, if(B, S, S)
 - ...

S+S无法满足原约束
所有展开S+S的探索都是浪费的
如何知道这一点？



基于反向语义（Inverse Semantics）的自顶向下遍历

- 首先对规约求解或者利用CEGIS获得输入输出对
 - 求模型： $ret \geq x \wedge ret \geq y \wedge (ret = x \vee ret = y)$
 - 得到 $x=1, y=2, ret=2$
- 由于只有加号，任何原题目的程序都必然满足：
 - $ret \geq x \vee ret \geq y$
- 以返回值作为约束去展开该程序
 - [2]S
 - [2]y, [1]S+[1]S, if([true]B, [2]S, [*]S), if([false]B, [*]S, [2]S)
 - ...
- 只有可能满足该样例展开方式才被考虑



Witness function

- Witness function 针对反向语义具体展开分析
- 输入：
 - 样例输入，如 $\{x=1, y=2\}$
 - 期望输出上的约束，如 $[2]$ ，表示返回值等于 2
 - 期望非终结符，如 S
- 输出：
 - 一组展开式和非终结符上的约束列表，如
 - $[2]y, [1]S+[1]S, \text{if}([\text{true}]B, [2]S, [*]S), \text{if}([\text{false}]B, [*]S, [2]S)$
- Witness Function 需要由用户提供

注：在原始文献中，witness 函数细分为 witness 和 skolemization 两种函数，这里简单起见不再区分。



Witness Function性质

- Witness Function具有必要性，如果
 - 满足原约束的所有程序都被至少一个展开式覆盖
- Witness Function具有充分性，如果
 - 满足展开式的所有程序都被原约束覆盖
- 必要的witness function保证不排除正确的程序
- 充分的witness function保证产生的程序一定是正确的



问题

- 多个样例怎么办？
 - 在CEGIS求解过程中，样例会逐渐增多，如何采用多个样例剪枝？
- 如何避免重复计算？
 - if([true]B, [2]S, [*]S),
 - if([false]B, [*]S, [2]S)
- 红色和绿色部分的展开完全相同，但却分布在两颗树中



FlashMeta

- 一个基于反向语义的程序综合框架
 - 由微软的Sumit Gulwani设计
- 采用了VSA来表示程序子空间
 - Version Space Algebra(VSA)是上下文无关文法的子集
 - 上下文无关文法求交不一定是上下文无关文法
 - VSA求交一定是VSA
- 微软的PROSE工具实现了FlashMeta框架



Sumit Gulwani
14年获SIGPLAN
Robin Milner青年
研究者奖



VSA

- VSA是只包含如下三种形式的上下文无关文法，且每个非终结符只在左边出现一次
 - $N \rightarrow p_1 \mid p_2 \mid \dots \mid p_n$
 - $N \rightarrow N_1 \mid N_2 \mid \dots \mid N_n$
 - $N \rightarrow f(N_1, N_2, \dots, N_n)$
 - N 是非终结符， p 是终结符列表， f 是终结符
- 无递归时，VSA可表示产生式数量指数级的程序空间。
- 有递归时，VSA可表示无限大的程序空间。

VSA例子

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

- Expr ::= V | Add | If
- Add ::= + (Expr, Expr)
- If ::= ite(BoolExpr, Expr, Expr)
- BoolExpr ::= And | Neg | Less
- And ::= \wedge (BoolExpr, BoolExpr)
- Neg ::= Not(BoolExpr)
- Less ::= \leq (Expr, Expr)



从样例产生语法规则

- 递归调用witness function，将约束和原非终结符同时作为新非终结符
- $[2]S \rightarrow y \mid [1]S + [1]S$
 $\mid if([true]B)[2]S [*]S \mid if([false]B) \dots$
- $[1]S \rightarrow x$
- $[*]S \rightarrow \dots$
- $[true]B \rightarrow true \mid \neg[false]B \mid [2]S \leq [2]S \mid$
 $[1]S \leq [2]S \mid [1]S \leq [1]S \mid \dots$

根据表达式最大大小可算出最大值
在最大值和最小值之间遍历所有可能



VSA求交

- $merge(N, N')$ 输出把非终结符 N 和 N' 求交之后的产生式
- $N \cap N'$ 表示把 N 和 N' 求交之后的终结符
- 如果 $N \rightarrow N_1 \mid N_2 \mid \dots$
 - $merge(N, N') = \{N \cap N' \rightarrow N_1 \cap N'_1 \mid N_2 \cap N'_2 \mid \dots\} \cup merge(N_1, N') \cup merge(N_2, N') \cup \dots$
- 如果 $N \rightarrow f(N_1 \mid \dots \mid N_k)$ 且 $N' \rightarrow f'(N'_1 \mid \dots \mid N'_{k'})$ 且 $f \neq f'$ 或者 $k \neq k'$
 - $merge(N, N') = \{N \cap N' \rightarrow \perp\}$
- 如果 $N \rightarrow f(N_1 \mid \dots \mid N_k)$ 且 $N' \rightarrow f(N'_1 \mid \dots \mid N'_k)$
 - $merge(N, N') = \{N \cap N' \rightarrow f(N_1 \cap N'_1, \dots, N_k \cap N'_k)\} \cup merge(N_1, N'_1) \cup \dots \cup merge(N_k, N'_k)$



VSA求交

- 如果 $N \rightarrow f(N_1 \mid \dots \mid N_k)$ 且 $N' \rightarrow p'_1 \mid p'_2 \mid \dots$ 且存在 $p'_{j_1}, \dots, p'_{j_m}$ 使得 N 可以生成 $p'_{j_1}, \dots, p'_{j_m}$ 中的每一个而不能生成别的
 - $merge(N, N') = \{N \cap N' \rightarrow p'_{j_1}, \dots, N \cap N' \rightarrow p'_{j_m}\}$
 - 否则 $merge(N, N') = \{N \cap N' \rightarrow \perp\}$
- 如果 $N \rightarrow p_1 \mid p_2 \mid \dots$ 且 $N' \rightarrow p'_1 \mid p'_2 \mid \dots$ 且 $p''_{j_1}, \dots, p''_{j_m}$ 是两者的公共部分
 - $merge(N, N') = \{N \cap N' \rightarrow p''_{j_1}, \dots, N \cap N' \rightarrow p''_{j_m}\}$
- 如果不符合以上情况, 则
 - $merge(N, N') = merge(N', N)$
- 最后再删掉所有产生结果都包括 \perp 的非终结符



多个样例的情况

- 每个样例产生VSA，然后求交
- 在一个VSA上用另外一个样例做过滤
 - 第二个样例上witness函数不能展开的选项就去掉
 - 在CEGIS的时候可以加快速度
- 两个样例同时生成
 - 两个样例同时产生的选项才保留
 - 在规约是样例的时候可以加快速度



基于抽象精化的合成



例子

- $n \rightarrow x \mid n + t \mid n \times t$
- $t \rightarrow 2 \mid 3$
- 输入: $x=1$, 输出: $ret=9$
- 目标程序举例: $(x+2)*3$

- 按某通用witness函数分解得到
- $[9]n_1 \rightarrow [1]n + [8]t \mid [2]n + [7]t \mid \dots$
 $\mid [1]n \times [9]t \mid [3]n \times [3]t \mid [9]n \times [1]t$

大量展开式都是无效的
能否一次排除而不是一个一个排除?



基本思想

- 之前见到的VSA按具体执行结果组织程序
- 但对于特定规约，很多具体程序是等价的
- 按抽象域组织程序可以进一步合并同类项

- 即：
- $[[5,12]]n \rightarrow [[0,4]]n + [[5,8]]t$

- 如何知道适合当前规约的抽象域是什么？
 - 从最抽象的抽象域开始，逐步精华



基于区间的元抽象域

- 糶, 即 $x \in [-\infty, +\infty]$
- ... $-7 \leq x \leq 0, 1 \leq x \leq 8, 9 \leq x \leq 18, \dots$
- ... $-3 \leq x \leq 0, 1 \leq x \leq 4, 5 \leq x \leq 8, \dots$
- ... $-1 \leq x \leq 0, 1 \leq x \leq 2, 3 \leq x \leq 4, \dots$
- ... $x = -1, x = 0, x = 1, \dots$

- 元抽象域由以上抽象值构成
- 实际抽象域的抽象值由元抽象域的值构成
- 一开始只包含糶, 在精化过程中逐步增加



1.1 抽象域上的计算

- 抽象域包括 棵
- 构造VSA, 得
 - $[\text{棵}] n \rightarrow x \mid [\text{棵}] n + [\text{棵}] t \mid [\text{棵}] n \times [\text{棵}] t$
 - $[\text{棵}] t \rightarrow 2 \mid 3$
- 输入为 $x = \text{棵}$, 输出为 $\text{ret} = \text{棵}$
- 随机从VSA中采样程序, 得到 $\text{ret} = x$



1.2 抽象域的精化

查找一个最抽象的抽象值，使得期望值和计算值不等
最抽象=偏序上达到极大
添加抽象值[1, 8]

| | | | |
|-----|-------|--------|-----------|
| 期望值 | ✖ | 9 | ✖ |
| 计算值 | x:✖ | x:1 | x:[1,8] |
| | 抽象域计算 | 实际域上反例 | 精华后的抽象域计算 |



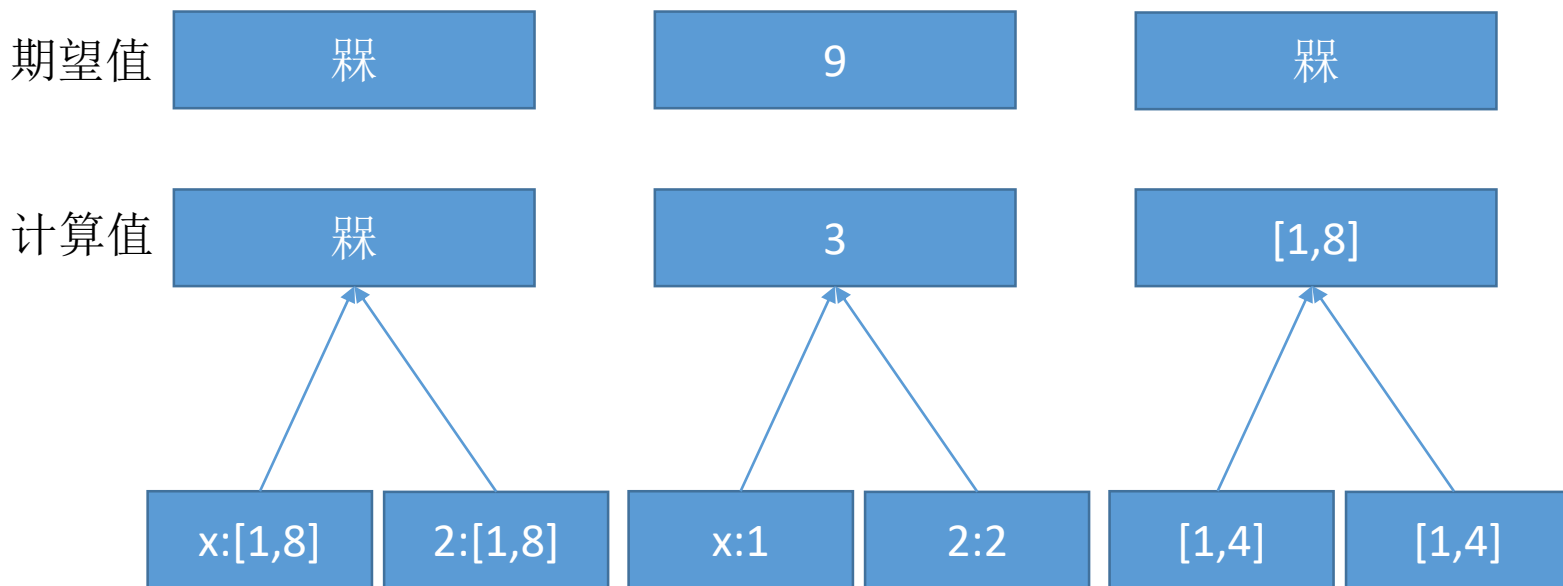
2.1 抽象域上的计算

- 抽象域包括 { 靛 , $[1, 8]$ }
- 构造VSA, 得
 - $[\text{靛}]n \rightarrow [\text{靛}]n + [1,8]t \mid [\text{靛}]n \times [1,8]t \mid [1,8]n + [1,8]t \mid [1,8]n \times [1,8]t$
 - $[1,8]n \rightarrow x$
 - $[1,8]t \rightarrow 2 \mid 3$
- 输入为 $x=[1,8]$, 输出为 $\text{ret}=\text{靛}$
- 随机从VSA中采样程序, 得到 $\text{ret}=x+2$



2.2 抽象域的精化

自顶向下依次精化
添加抽象值[1, 4]



抽象域计算

实际域上反例

精华后的抽象域计算



3.1 抽象域上的计算

- 抽象域包括 $\{\text{靛}, [1, 8], [1, 4]\}$
- 构造VSA, 得
 - $[\text{靛}]n \rightarrow [\text{靛}]n + [1, 4]t \mid [\text{靛}]n \times [1, 4]t \mid [1, 8]n + [1, 4]t \mid [1, 8]n \times [1, 4]t \mid \dots$
 - $[1, 8]n \rightarrow [1, 4]n + [1, 4]t \mid \dots$
 - $[1, 4]n \rightarrow x$
 - $[1, 4]t \rightarrow 2 \mid 3$
- 输入为 $x=[1, 4]$, 输出为 $\text{ret}=\text{靛}$
- 随机从VSA中采样程序, 得到 $\text{ret}=(x+2)*3$



冲突制导的学习



复习： SAT

- 什么是DPLL?
- 什么是CDCL?



例子：序列操作合成

$N \rightarrow \emptyset \mid \dots \mid 1\emptyset \mid x_i \mid \text{last}(L) \mid \text{head}(L) \mid \text{sum}(L)$
 $\mid \text{maximum}(L) \mid \text{minimum}(L)$

$L \rightarrow \text{take}(L, N) \mid \text{filter}(L, T) \mid \text{sort}(L) \mid \text{reverse}(L) \mid x_i$

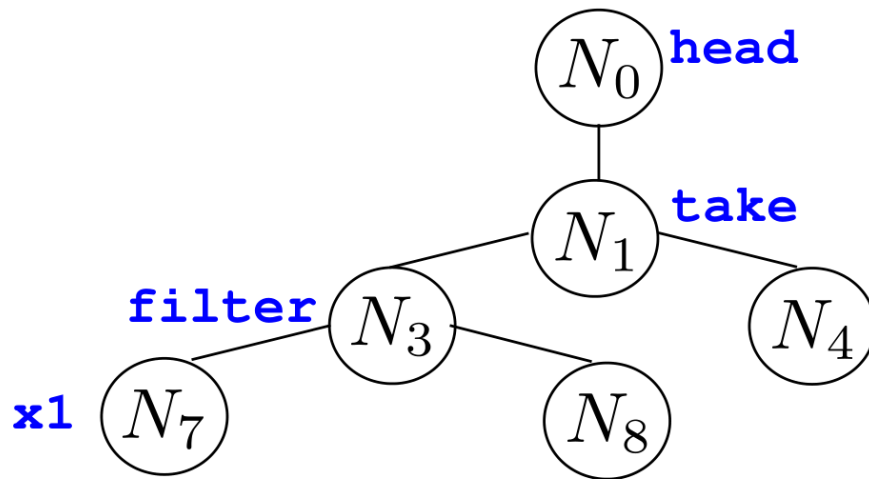
$T \rightarrow \text{geqz} \mid \text{leqz} \mid \text{eqz}$

输入： $x=[49, 62, 82, 54, 76]$

输出： $y=158$



分析（不完整）程序



给定一个可能不完整的程序：

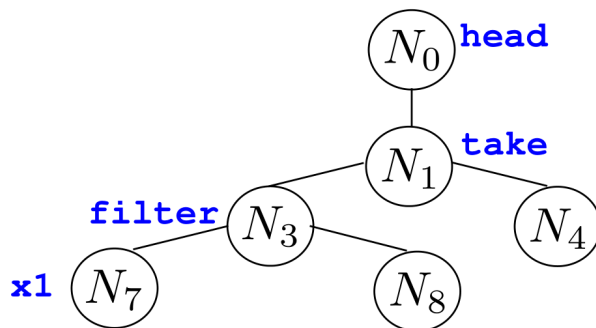
1. 如何知道该程序不满足规约？
2. 如何利用该分析过程排除更多的程序？



如何知道该程序不满足规约?

| 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 | $y.size < x_1.size \wedge y.max \leq x_1.max$ |

该规约可以不充分但必须必要，子句越多越好



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

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

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

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

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

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

加下划线的为极小矛盾集 ψ

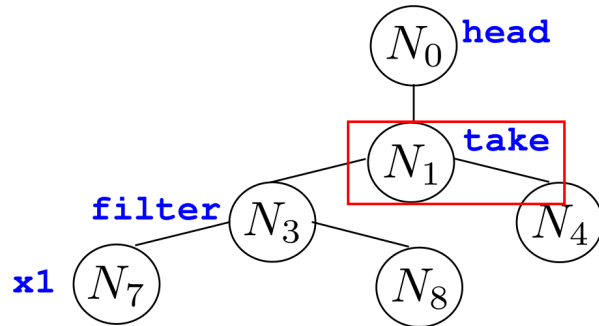
如何利用该分析过程排除更多的程序？



- 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



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

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

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

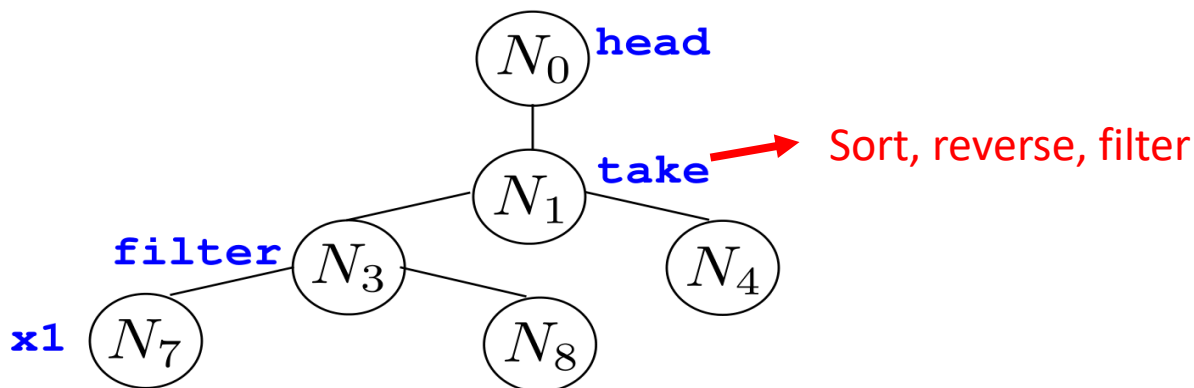
$$\phi_{N_7} = \underline{x_1 = v_7}$$

- 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。



参考文献

- Polozov O , Gulwani S . FlashMeta: a framework for inductive program synthesis[C]// Acm Sigplan International Conference on Object-oriented Programming. ACM, 2015.
- Feng Y , Martins R , Bastani O , et al. Program Synthesis using Conflict-Driven Learning[J]. PLDI 2018.
- Wang X , Dillig I , Singh R . Program Synthesis using Abstraction Refinement[J]. POPL 2018.