软件分析

程序综合：优化

熊英飞
北京大学
2018

基于空间表示的合成

例子：化简的max问题

- 语法：

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

- 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

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, if([true]B, [2]S, [*]S), if([false]B, [*]S, [2]S)

- Witness Function需要由用户提供

注：在原始文献中，witness函数细化为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
- 采用了语法产生式来表示程序子空间
  - 一组语法产生式被机器学习研究人员称为一个Version Space Algebra(VSA)
  - 一个VSA表示满足一个样例的程序空间
  - 两个VSA可以完成求交操作
- 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是对原产生式精化得到的一组新的产生式，如：

- $S_1 \rightarrow y \mid S_2 + S_2 \mid if(B_1)\ S_3\ S_4 \mid if(B_2)\ S_4\ S_3$
- $S_2 \rightarrow x$
- $S_3 \rightarrow y$
- $S_4 \rightarrow x \mid y$
- $B_1 \rightarrow true \mid x \leq y$
- $B_2 \rightarrow false \mid y \leq x$

- 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。

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
- $[2]S \to y \mid [1]S + [1]S$
  $\mid if([true]B)[2]S\ [*]S \mid if([\text{false}]B)$ …
- $[1]S \to x$
- $[*]S \to \cdots$
- $[true]B \to true \mid \neg[false]B \mid [2]S \leq [2]S \mid$
  $[1]S \leq [2]S \mid [1]S \leq [1]S \mid \cdots$

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

语法规则求交

- 假设
  - $N \to P_1 \mid \cdots \mid P_k$
  - $N' \to P'_1 \mid \cdots \mid P'_{k'}$

- $N \cap N' = N''$
  - 且$N'' = P_1 \cap P'_1 \mid P_1 \cap P'_2 \mid \cdots \mid P_1 \cap P'_{k'}$
    $\mid P_2 \cap P'_1 \mid P_2 \cap P'_2 \mid \cdots \mid P_2 \cap P'_{k'}$
    $\mid \cdots$
    $\mid P_k \cap P'_1 \mid P_k \cap P'_2 \mid \cdots \mid P_k \cap P'_{k'}$
- $P_1 \cap P_2 = \emptyset$　如果P_1和P_2不是同一类型
- $f(N_1, \ldots, N_k) \cap f(N'_1, \ldots, N'_k) = f(N_1 \cap N'_1, \ldots, N_k \cap N'_k)$　如果f是k元操作符

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

- $n \to x \mid n + t \mid n \times t$
- $t \to 2 \mid 3$
- 输入：x=1，输出：ret=9
- 期望：(x+2)*3

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

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

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

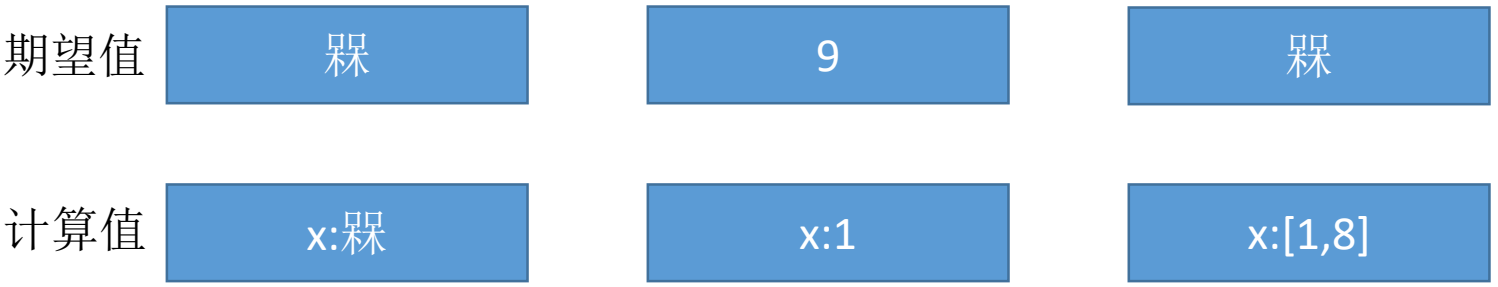
- 元抽象域由以上抽象值构成
- 实际抽象域的抽象值由元抽象域的值构成
- 一开始只包含True，在精化过程中逐步增加

1.1 抽象域上的计算

- 抽象域包括 踝
- 构造VSA，得
  - [踝]$n \to x$ | [踝]$n +$ [踝]t | [踝]$n \times$ [踝]t
  - [踝]$t \to 2$ | 3
- 输入为x=踝，输出为ret=踝
- 随机从VSA中采样程序，得到ret=x

1.2 抽象域的精化

寻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]



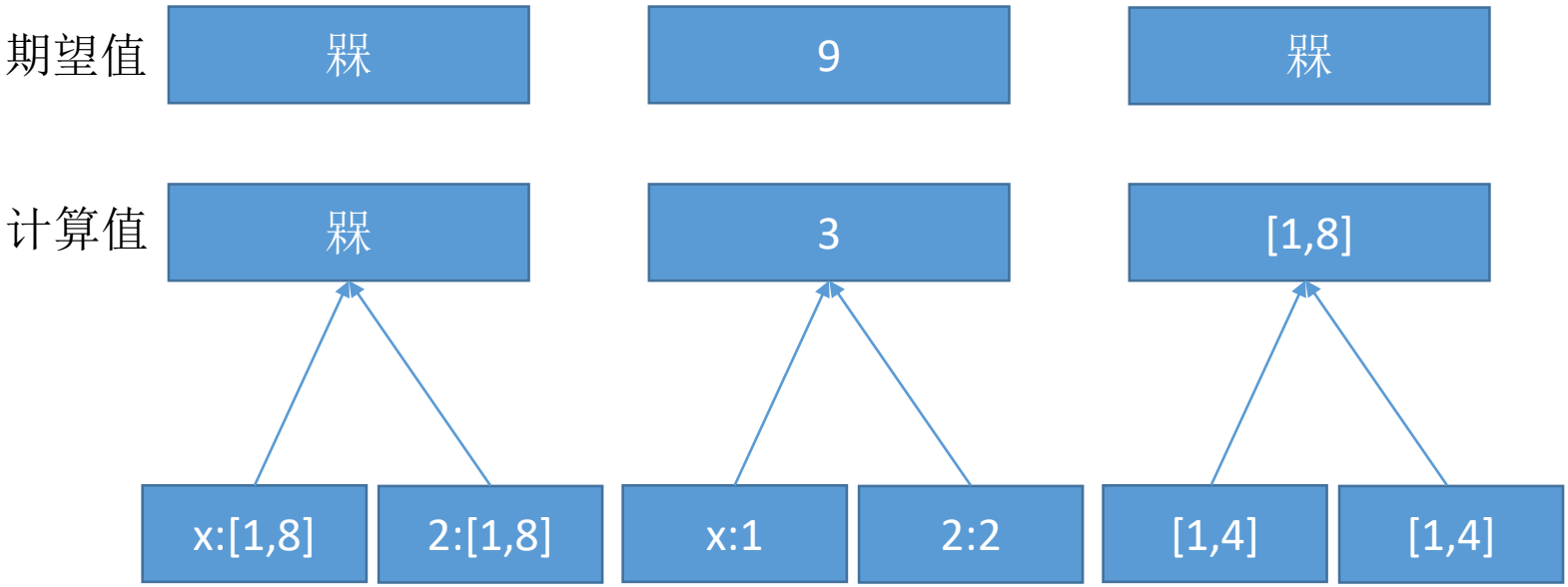
抽象域计算　　实际域计算　　精华后的抽象域计算

2.1 抽象域上的计算

- 抽象域包括 {粿, [1, 8]}
- 构造VSA，得
  - [粿]$n \rightarrow$ [粿]n + [1,8]t | [粿]$n \times$ [1,8]t | [1,8]n + [1,8]t | [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ret=粿
- 随机从VSA中采样程序，得到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　　实际域计算　　精华后的抽象域计算

3.1 抽象域上的计算

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

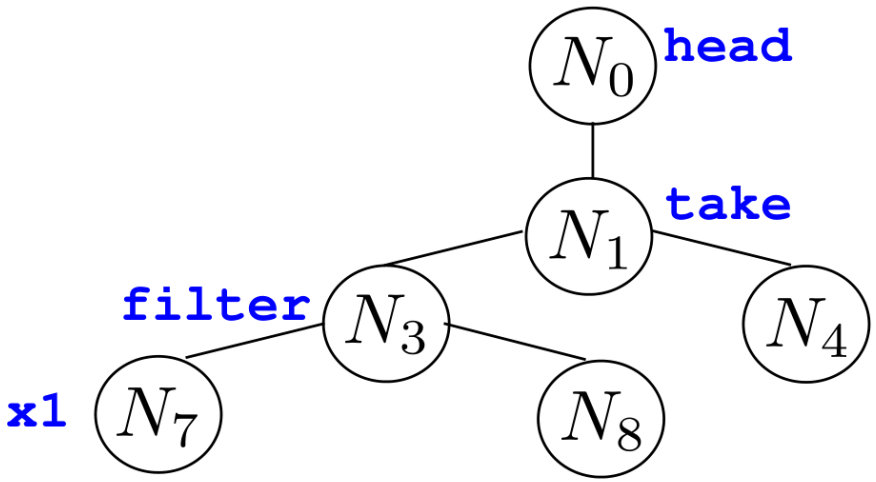
$N \rightarrow$ 0 | ... | 10 | x_i | last(L) | head(L) |sum(L)
| maximum(L) | minimum(L)

$L \rightarrow$ 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

分析（不完整）程序



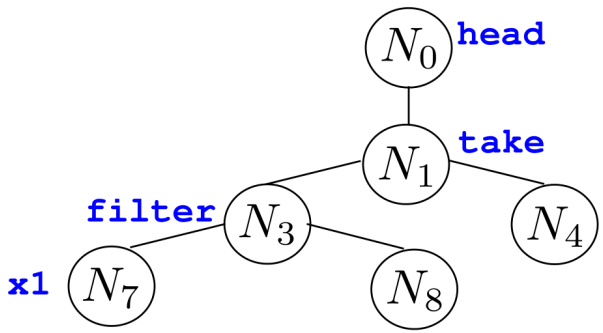
给定一个可能不完整的程序：

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



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

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

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

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

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

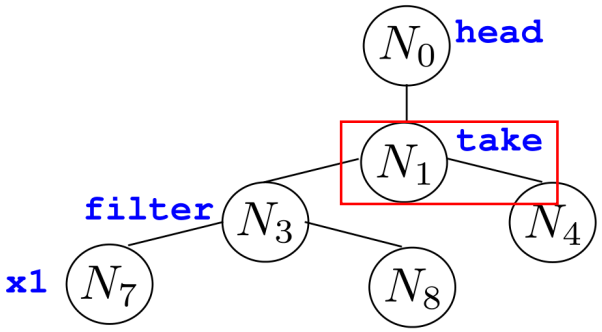
加下划线的为极小矛盾集ψ

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 ⇒ 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



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

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

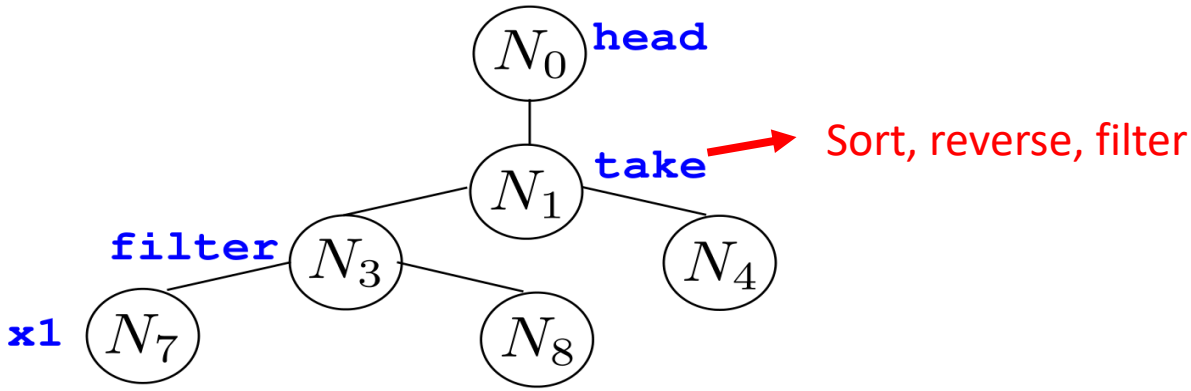
$$\phi_{N_3} = v_3.size < v_7.size \ \wedge \ v_3.max \le v_7.max$$

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

- 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$\Rightarrow y.max \le x_1.max$
- 则
  - f在N_1位置和当前冲突等价
- 因为只涉及到一个组件的规约，可以用SMT solver快速验证

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。

现状和总结

- 存在多种不同已有的方法来加速程序综合
- 但仍有很多问题未被解答
  - 已有的方法需要大量领域特定的定制，如何定制？
  - 多种不同的加速方法如何合并起来工作？
  - 是否还存在更好的加速方法？
- 未来还需要大量投入

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]. 2017.
- Wang X , Dillig I , Singh R . Program Synthesis using Abstraction Refinement[J]. 2017.