



软件分析

程序综合：基础

熊英飞
北京大学
2018



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



程序综合

- 根据用户需求（形式化规约、自然语言描述等）自动生成程序
- “程序语言理论中最核心的问题之一”
 - ——Amir Pnueli, 图灵奖获得者



程序综合的常见形式

- 用户需求
 - 基于霍尔逻辑的形式化规约
 - 输入输出样例
 - 自然语言描述
 - 相同功能的程序
 - 程序执行的Trace
 - 部分程序
- 程序
 - 采用语法和语义来定义
 - 生成的程序符合语法的要求
 - 对于形式化规约，生成的程序在语义上保证满足规约



常见应用——Excel

	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



常见应用——程序优化

- 用户撰写程序
 - $a = a * 32$
- 自动优化成
 - $a = a \ll 5$

常见应用——自动编写重复程序



```
[NAME]
Acidic Swamp Ooze
[ATK] 3
[DEF] 2
[COST] 2
[DUR] -1
[TYPE] Minion
[CLASS] Neutral
[RACE] NIL
[RARITY] Common
[DESCRIPTION]
"Battlecry: Destroy Your Opponent's Weapon"
```

```
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)
```



常见应用——代码补全

```
1 import tensorflow as tf
2 import numpy as np
3 def training(loss, learning_rate):
4     tf.summary.scalar('loss', loss)
5     optimizer = tf.train.
6     GradientDescentOptimizer(learning_rate)
7
8
9
```

```
textclassification
with tf.name_scope('hidden1'):
    weights = tf.Variable(
        tf.truncated_normal(
            [IMAGE_PIXELS, hidden1_units],
            stddev=1.0 / math.sqrt(float
                (IMAGE_PIXELS))),
        name='weights')
    biases = tf.Variable(tf.zeros(
        [hidden1_units]),
        name='biases')
    hidden1 = tf.nn.relu(
        tf.matmul(images, weights) + biases)
```

```
textclassification
def estimator_spec_for_softmax_classification(
    logits, labels, mode):
    predicted_classes = tf.argmax(logits, 1)
    if mode == tf.estimator.ModeKeys.PREDICT:
        return tf.estimator.EstimatorSpec(
            mode=mode,
            predictions=[
                'class': predicted_classes,
                'prob': tf.nn.softmax(logits)
            ])
    )
```




程序综合是软件分析问题

- 程序综合问题：编写程序实现函数 f ，满足 $f(x, 1) = x \wedge f(x, y) = f(y, x)$
 - $\text{expr} = \text{var op var}$
 - $\text{var} = x \mid y$
 - $\text{op} = + \mid - \mid * \mid /$
- 给空间里的所有程序编号，然后编写如下程序：

```
Int f(n, x, y) {  
    switch(n) {  
        case 1: return x+y;  
        case 2: return x-y;  
        case 3: return x*y; }  
}
```
- 软件分析问题：是否存在 n ，使得上述规约满足？



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>



例子：max问题

- 语法：

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

- 规约：

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

- 期望答案： $\text{ite } (x \leq y) \ y \ x$



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

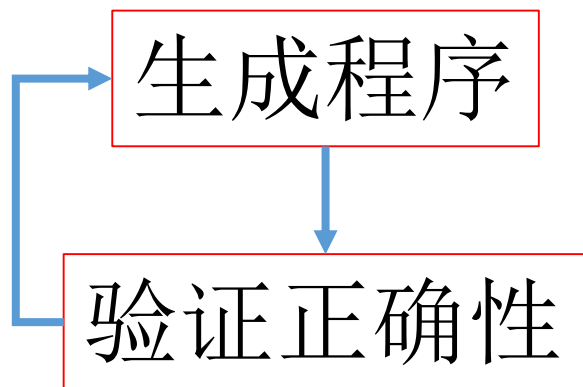


课程项目2

- 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
- 限制：只考虑基础算术和逻辑表达式
- 截止日期：12月31日提交，1月2日报告
- 程序包包括：
 - 完整的测试环境（含所有官方测试用例）
 - 最基本的solver（解不出来几道题）
- 评分：根据解出来的样例个数评分（每个时限5分钟）
- 明天课上曾沐焱同学详细介绍



程序综合作为搜索问题



如何产生下一个被搜索的程序？

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

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 —— 将程序搜索问题整体转成约束求解问题
 - 启发式搜索法 —— 采用启发式搜索
 - 统计法 —— 采用机器学习等方法寻找概率最高的程序



枚举法



自顶向下遍历

- 按语法依次展开
 - S
 - $x, y, S+S, S-S, \text{if}(B, S, S)$
 - $y, S+S, S-S, \text{if}(B, S, S)$
 - $S+S, S-S, \text{if}(B, S, S)$
 - $x+S, y+S, S+S+S, S-S+S, \text{if}(B, S, S)+S, S-S, \text{if}(B, S, S)$
 - ...



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



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**

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

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

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

return c

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

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

progSize $\leftarrow \text{progSize} + 1$

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

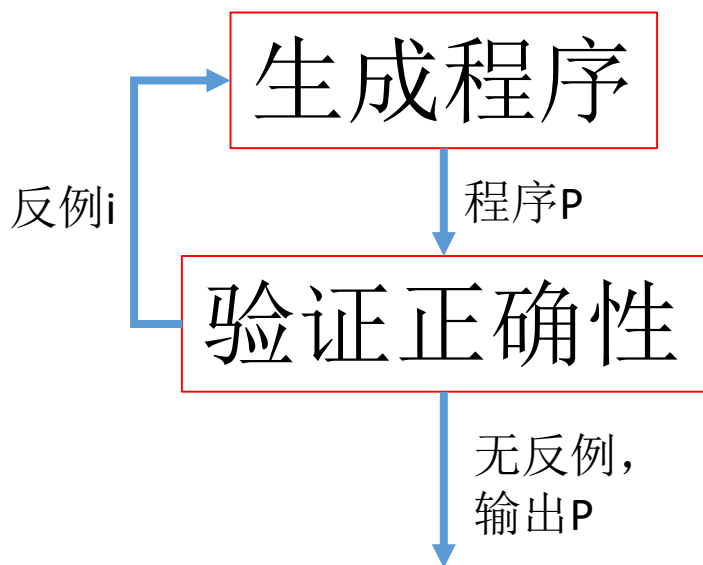


判断程序是否等价

- 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
 - 对于不完整的程序不容易编码
- 通过预定义规则判断
 - 如 $S+x$ 和 $x+S$ 的等价性



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
- 优化一：验证的时候首先采用测试验证
- 优化二：判断等价性的时候首先采用测试的结果判断



约束求解法

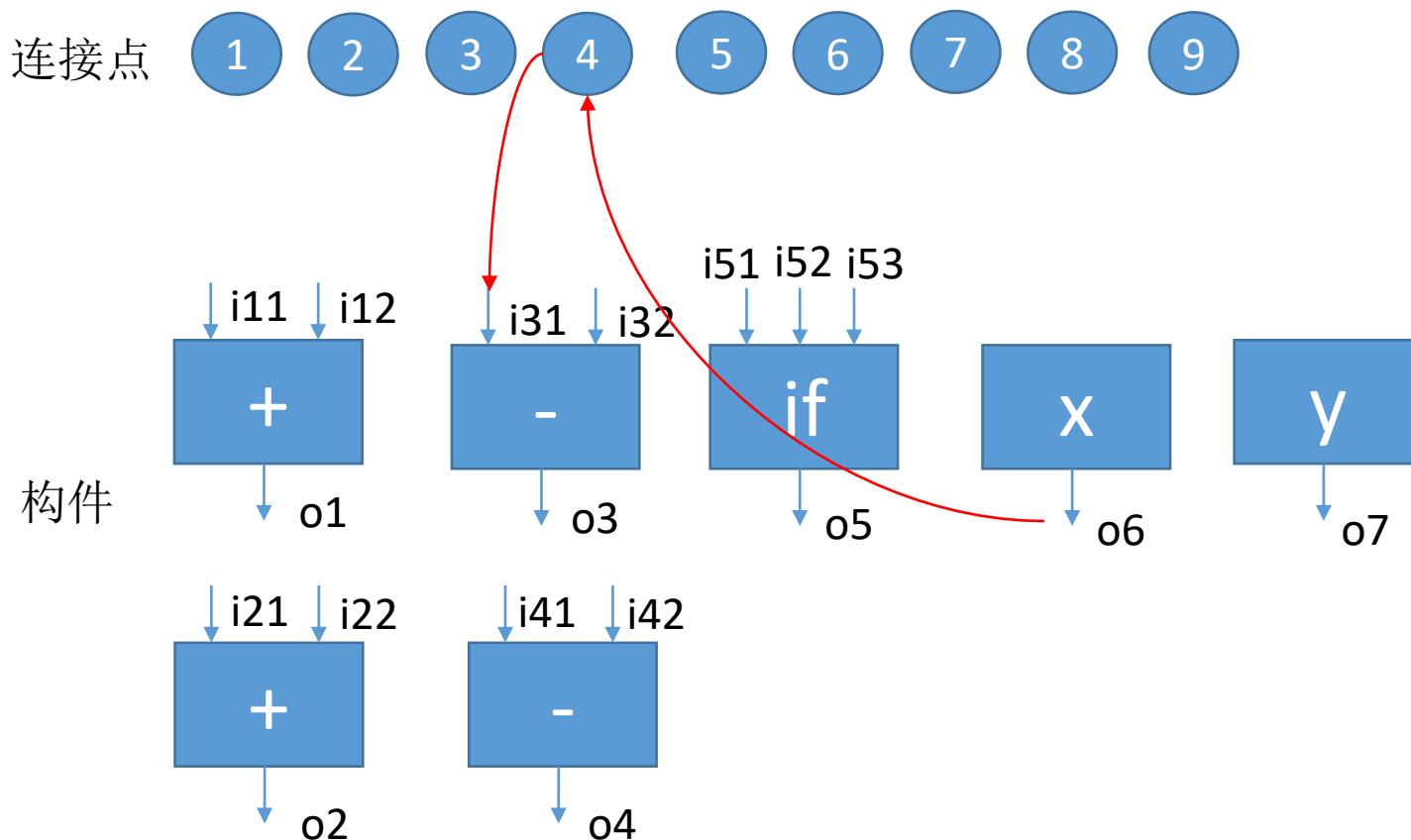


约束求解法

- 将程序综合问题整体转换成约束求解问题，由SMT求解器求解

基于构件的程序综合

Component-Based Program Synthesis



添加标签变量:

- l_{i11}, l_{i22}, \dots
- l_{o1}, l_{o2}, \dots
- l_o : 程序输出

$$l_{o6} = l_{i31} = 4$$



产生约束

- 产生规约约束:
 - $\forall x, y: o \geq x \wedge o \geq y \wedge (o = x \vee o = y)$
- 对所有component产生语义约束:
 - $o_1 = i_{11} + i_{12}$
- 对所有的输入输出标签对产生连接约束:
 - $l_{o_1} = l_{i_{11}} \rightarrow o_1 = i_{11}$
- 对所有的输出标签产生编号范围约束
 - $l_{o_1} \geq 1 \wedge l_{o_1} \leq 9$
- 对所有的 o_i 对产生唯一性约束
 - $l_{o_1} \neq l_{o_2}$
- 对统一构件的输入和输出产生防环约束
 - $l_{i_{11}} < l_{o_1}$

能否去掉连接点和输出标签 $l_{ox} \dots$ ，直接用 l_{ixx} 的值表示应该连接第几号输出？



约束限制

- 之前的约束带有全称量词，不好求解
- 实践中通常只用于规约为输入输出样例的情况
- 假设规约为
 - $f(1,2) = 2$
 - $f(3,2) = 3$
- 则产生的约束为：
 - $x = 1 \wedge y = 2 \rightarrow o = 2$
 - $x = 3 \wedge y = 2 \rightarrow o = 3$
- 通过和CEGIS结合可以求解任意规约



启发式搜索法



启发式搜索法

- 定义fitness函数
 - 通过的测试样例的数量
- 初始程序
 - 通常随机产生
- 定义变异操作（爬山法、模拟退火、遗传算法）
 - 随机将一颗子树替换成另一颗子树
- 定义交叉操作
 - 随机交换两个程序的两颗子树



统计法



统计法

- 狭义程序综合问题：返回满足形式化规约的程序
- 广义程序综合问题，也称程序估计问题：
 - 找到最大可能满足需求的程序，即求解
$$\operatorname{argmax}_{prog} P(prog \mid context)$$
 - **Context**: 程序的需求，如规约，输入输出样例，自然语言需求等
- 对于概率的估计也可以用于加速狭义程序综合问题的求解



统计法

- 自顶向下展开的过程中，通过统计/机器学习求不同产生式的概率
- 如：
 - $x + S$ ，展开 S
 - $S \rightarrow x$ 0.2
 - $S \rightarrow y$ 0.3
 - $S \rightarrow S + S$ 0.1
 - ...
- 即求 $P(\text{rule} \mid \text{context}, \text{prog}, \text{position})$
 - rule: 某个特定的产生式
 - context: 当前的上下文，如：程序的规约
 - prog: 目前已经产生的程序
 - position: 被展开的非终结符的位置



统计法

- 可以证明，对于任意给定的一组展开序列 $(position_i, rule_i)$ ，最后展开得到的程序 $prog$ 的概率为
 - $P(prog | context) = \prod_i P(rule_i | context, prog_i, position_i)$
 - 其中 $prog_i$ 是第 $i-1$ 次规则执行后得到的程序
- 也就是说，程序的概率计算和展开的顺序无关

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



通过机器学习估计概率

- 训练集：
 - 随机生成题目，用传统方法生成程序
 - 或者随机生成程序，用谓词变换计算题目
- 对于每个程序，按固定格式分解出一个产生式序列
- 特征提取：
 - 根据问题设计从 $context, prog, position$ 中提取特征的方法
 - 或者设计将 $context, prog, position$ 编码输入神经网络的方法



寻找概率最大的程序

- 需要求解 $\operatorname{argmax}_{prog} P(prog \mid context)$
- 贪心法：每次选择概率最大的产生式
- **Beam Search**：保留最多k个概率最大的程序，然后每次对这些程序展开一步，然后排序前k个
- 启发式搜索算法：设置fitness函数为 $P(prog \mid context)$



预测顺序问题

- 假设要预测的目标表达式为
 - $a > 0$
- 可能有以下概率：
 - $P(a \mid context) = 0.8$
 - $P(b \mid context) = 0.1$
 - $P(c \mid context) = 0.05$
 - $P(Expr > 0 \mid context) = 0.1$
 - $P(Expr > 1 \mid context) = 0.11$
 - $P(Expr \leq 0 \mid context) = 0.12$
- 如果自顶向下进行Beam搜索，很可能搜索不到



展开顺序定义

- 引入新的向上产生式
 - $\widehat{Expr} \hookrightarrow BoolExpr(\widehat{Expr} > 0)$
 - 允许 a 向上展开成 $a > 0$
- 转换原始语法为一组新产生式
 - $Expr \rightarrow a|b|c$
 - $\widehat{Expr} \hookrightarrow \begin{array}{l} BoolExpr(\widehat{Expr} > 0) \\ | BoolExpr(\widehat{Expr} > 1) \\ | BoolExpr(\widehat{Expr} \leq 0) \\ \dots \end{array}$



扩展产生式

- 无二义性：给定任意程序，如果当前规则集合只能通过同一个集合的规则应用来展开（顺序可变），则称规则无二义性
- 扩展后的产生式如果满足无二义性，则仍然有
 - $P(prog \mid context) = \prod_i P(rule_i \mid context, prog_i, position_i)$
- 可以通过定义无二义性的规则集合来控制展开顺序



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