软件全方位检测技术论坛

从参数化到概率化

北京大学在程序合成和调试上的近期部分工作介绍

汇报人：熊英飞

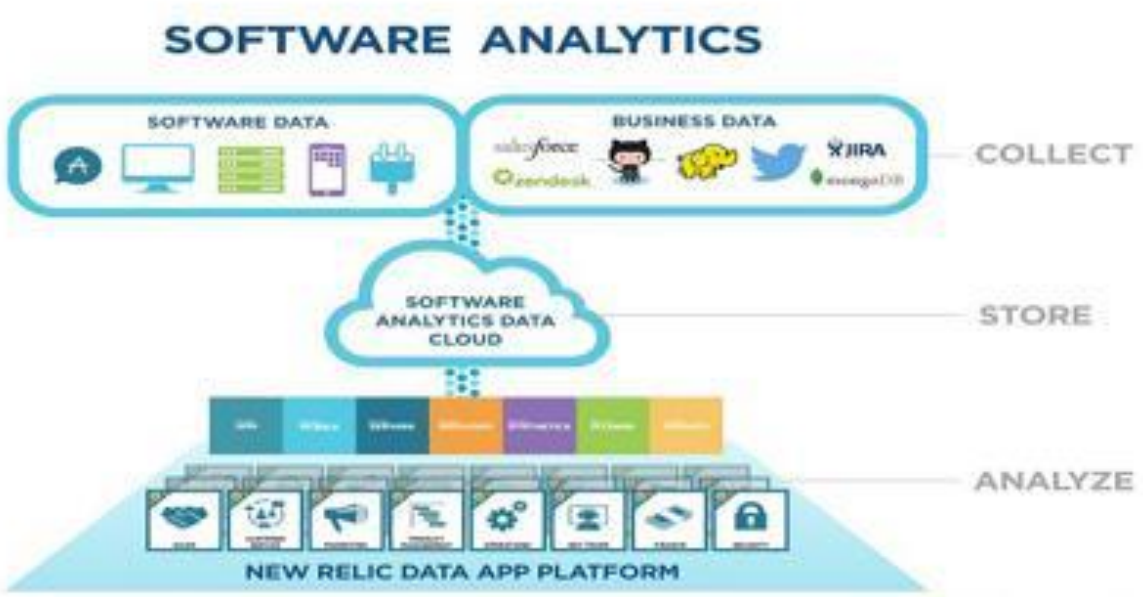
北京大学信息科学技术学院计算机科学技术系

数据驱动的软件工程

过去二十年中，软件工程发展的一大趋势是利用软件大数据支撑软件工程各项任务。

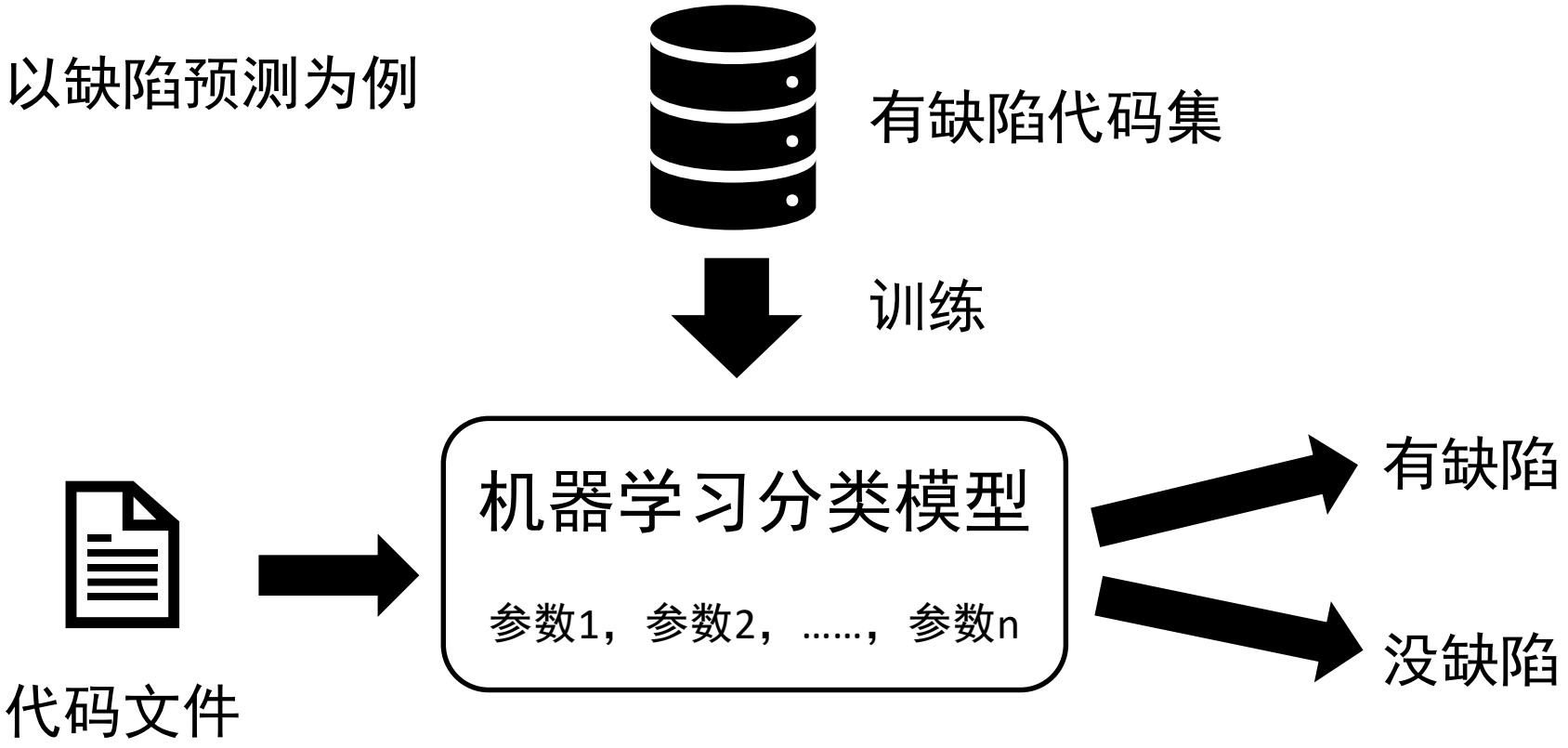
MSR 2020 Mon 29 - Tue 30 June 2020

Attending Travel Support Program Tracks Organization Search Series



已有工作的主要特征是参数化

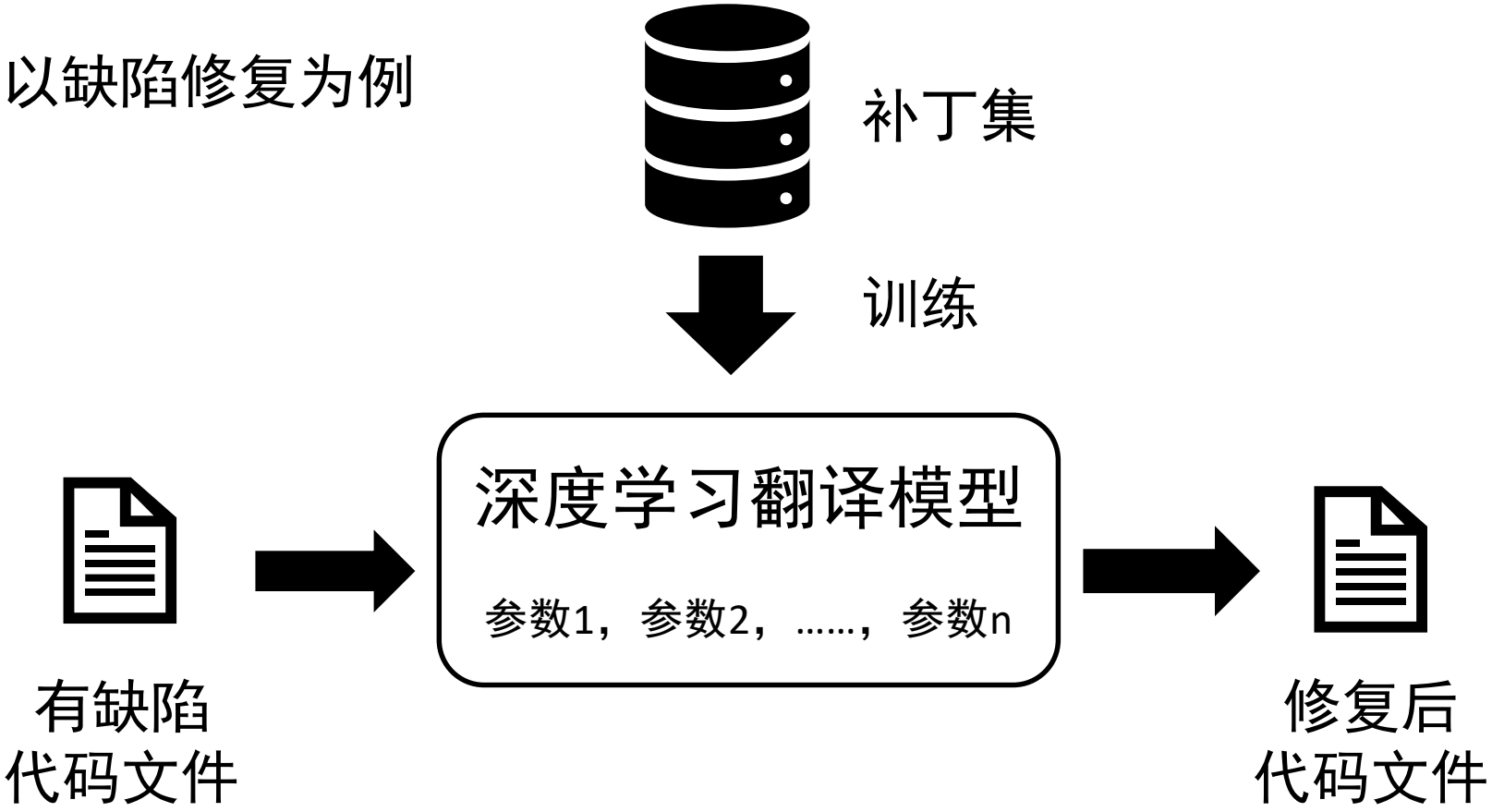
参数化：给定训练集，利用已有机器学习或数据挖掘模型和算法，找到在训练集上表现最好的一组模型参数。



已有工作的主要特征是参数化

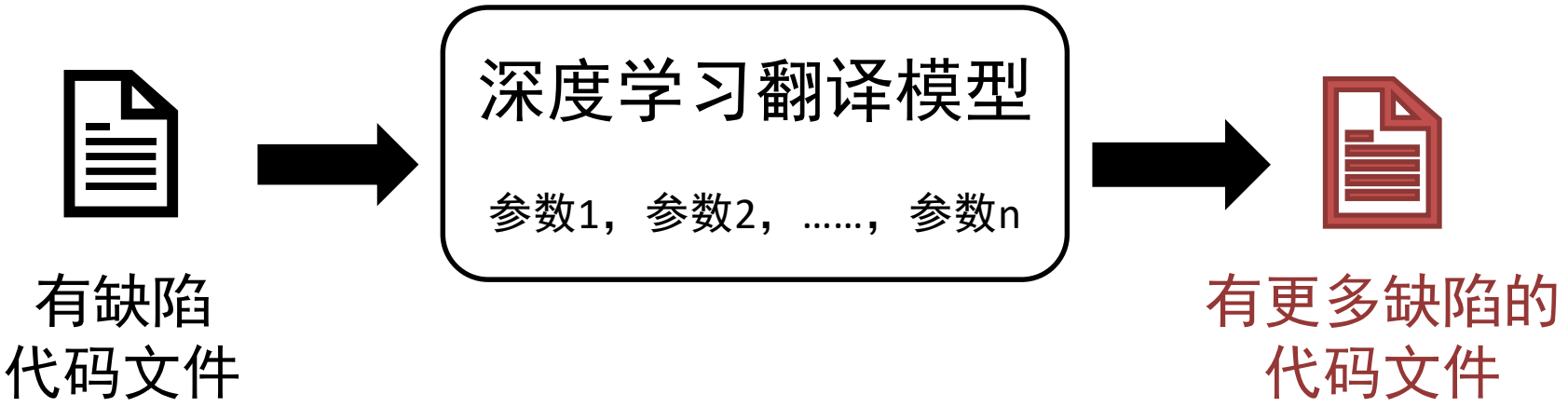
参数化：给定训练集，利用已有机器学习或数据挖掘模型和算法，找到在训练集上表现最好的一组模型参数。

以缺陷修复为例



参数化的问题–难解释

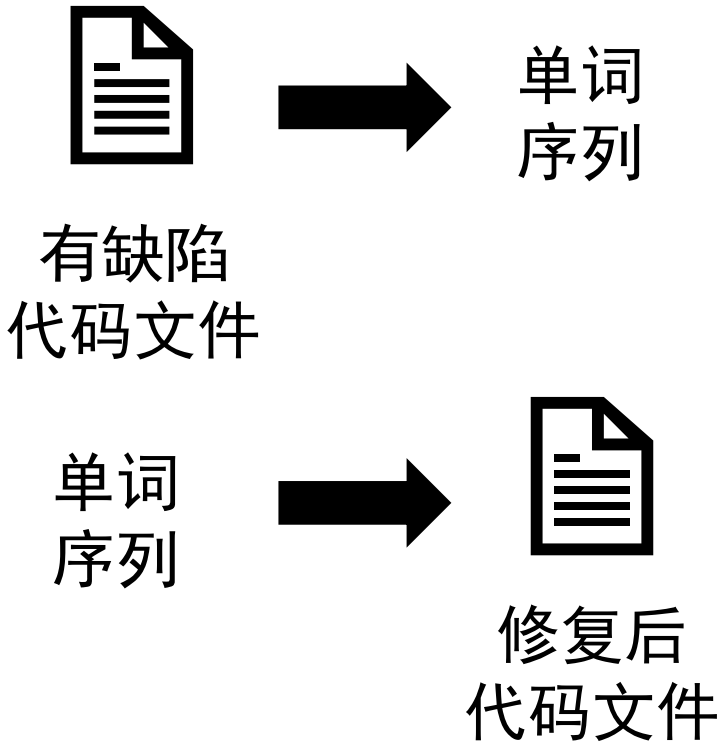
参数并没有概率上或者逻辑上的意义，难以理解学出的模型是如何工作的，在效果不好的时候也不知道如何改进。

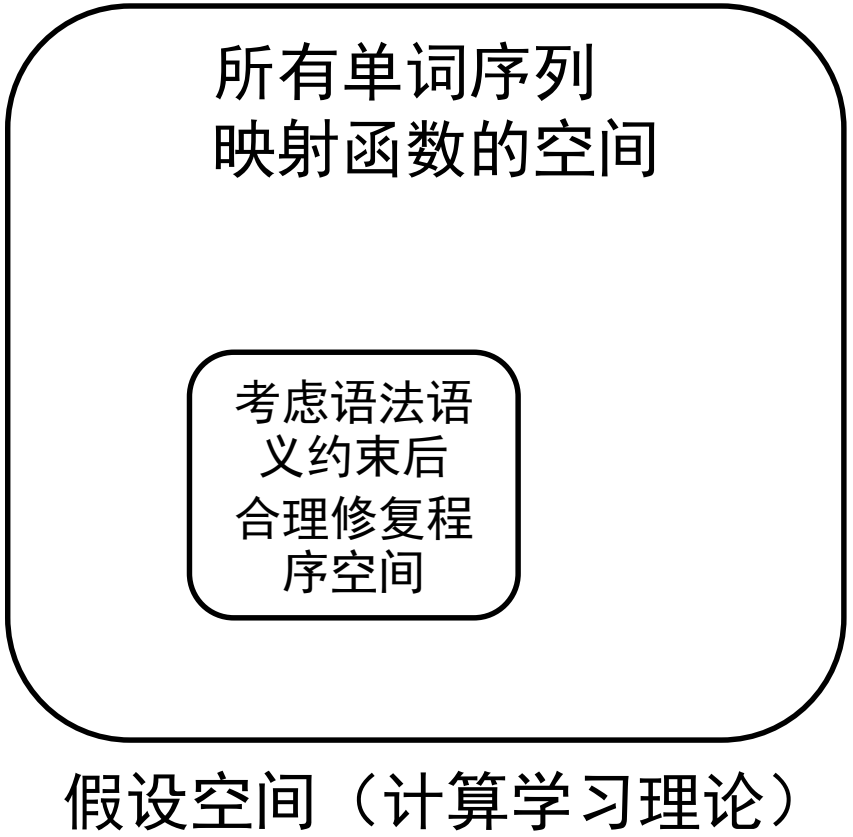


基于深度学习的缺陷修复提出4年多，效果一直没能超过传统基于启发式规则的缺陷修复技术。

参数化的问题-难定制

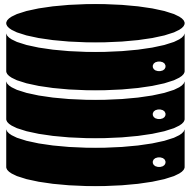
软件系统有大量语法、语义等领域知识，构建软件工程工具往往需要用这些知识做推理，但学习模型无法利用这些知识，也很难从数据中学会这些知识。





参数化的问题–难组合

现有学习模型采用大量端到端的数据训练，难以用于其他任务，也难以利用少量的或非端到端的数据。



补丁集

训练

深度学习翻译模型

参数1，参数2，……，参数n

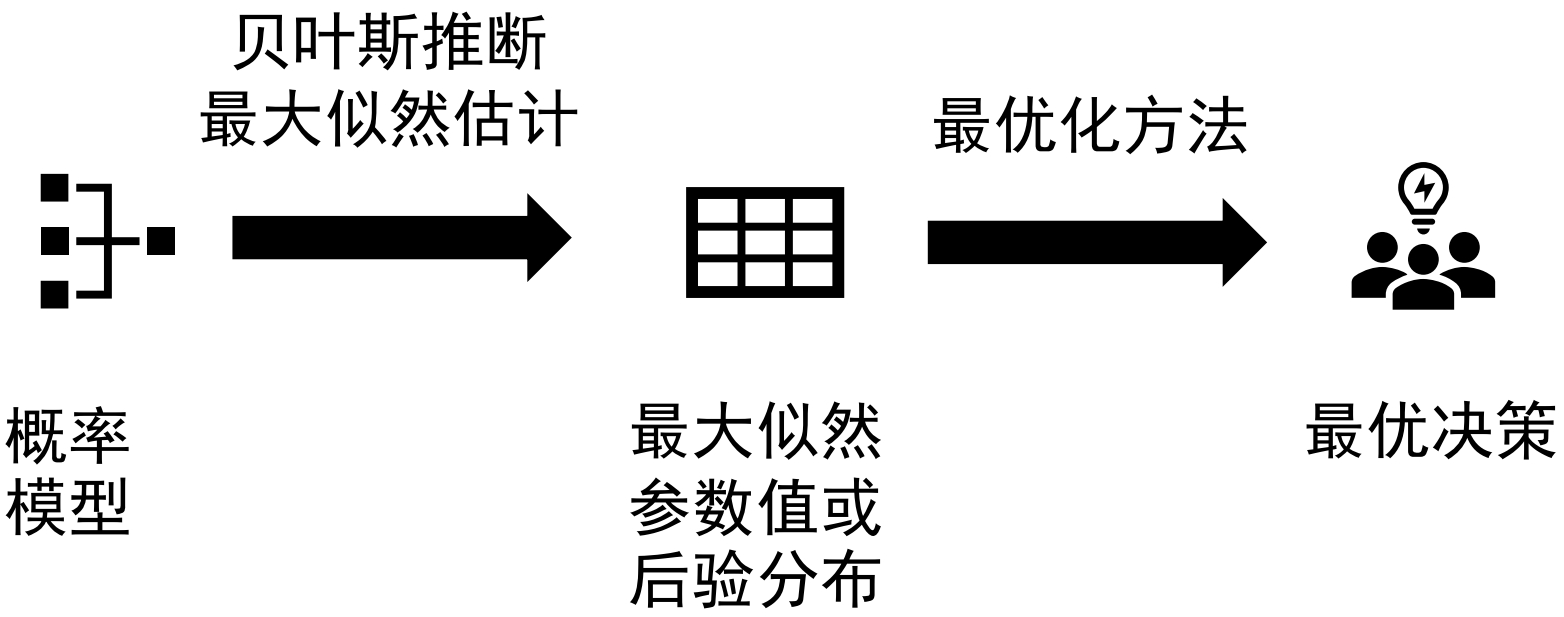
正确代码数据

少量缺陷补丁

- 很多缺陷类型无法收集到足够多的补丁，难以训练模型，也无法利用其他数据。
- 训练好的模型也难以用于其他任务，如刻画补丁的分布

从参数化到概率化

概率化：构造概率模型，通过贝叶斯推断、最大似然估计的方式学习，通过最优化方法做出决策。



概率化的好处

可解释

- 概率模型的参数都有明确的概率解释
- 根据参数分布可知优化方向
  - 如：置信区间过大，则需要增加训练数据

可定制

- 软件领域知识可建模在模型中
- 如：A=10 → B = 10
  - 则P(B=10 | A=10)=100%

可组合

- 可针对不同的优化目标寻找最优策略
- 可根据不同的观察值计算后验概率/最大似然参数值，且一个数据也能计算

本报告剩余内容

概率差异化调试

代码生成框架——玲珑框架及应用

差异化调试

差异化调试：在保留特定性质的前提下约减一个元素集合。

应用领域：编译器调试、回归调试、缺陷理解、软件精简

```
1 import tensorflow as tf
2 x = tf.constant(3.0)
3 b = 1.0
4 with tf.GradientTape() as tape:
5     tape.watch(x)
6     y = x**2
7 b = tape.gradient(y,x)
8 print(type(b))
```

现有工作

主流差异化调试算法围绕ddmin算法构建

- 2002年提出，近20年没有本质变化

采用固定的尝试顺序

- 无法充分利用测试结果包含的信息

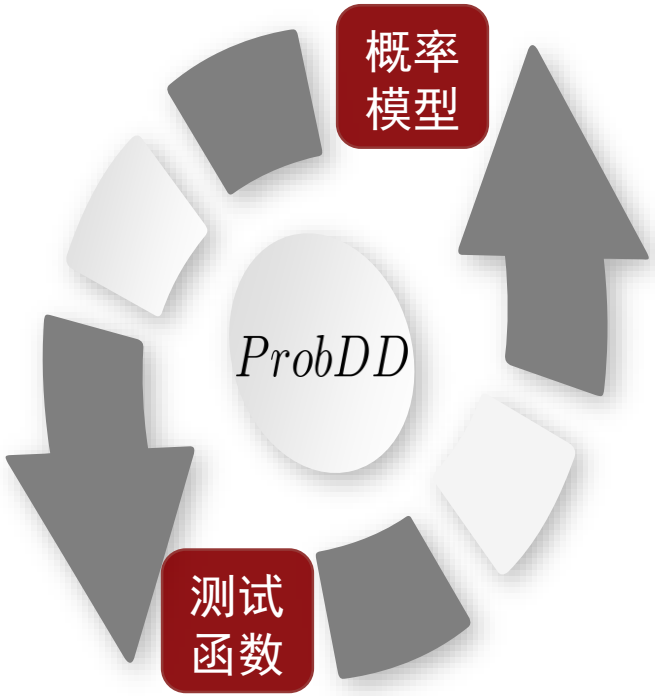
实际应用中效果不尽人意

- 一次差异化调试常常需要数小时或数十小时
- 结果可能数倍于最优结果

	s1	s2	s3	s4	s5	s6	s7	s8	
1	s1	s2	s3	s4	s5	s6	s7	s8	F
2	s1	s2	s3	s4	s5	s6	s7	s8	F
3	s1	s2	s3	s4	s5	s6	s7	s8	F
4	s1	s2	s3	s4	s5	s6	s7	s8	F
5	s1	s2	s3	s4	s5	s6	s7	s8	F
6	s1	s2	s3	s4	s5	s6	s7	s8	F
7	s1	s2	s3	s4	s5	s6	s7	s8	F
8	s1	s2	s3	s4	s5	s6	s7	s8	F
9	s1	s2	s3	s4	s5	s6	s7	s8	F
10	s1	s2	s3	s4	s5	s6	s7	s8	F
11	s1	s2	s3	s4	s5	s6	s7	s8	F
12	s1	s2	s3	s4	s5	s6	s7	s8	F
13	s1	s2	s3	s4	s5	s6	s7	s8	F
14	s1	s2	s3	s4	s5	s6	s7	s8	F
15	s1	s2	s3	s4	s5	s6	s7	s8	F
16	s1	s2	s3	s4	s5	s6	s7	s8	F
17	s1	s2	s3	s4	s5	s6	s7	s8	F
18	s1	s2	s3	s4	s5	s6	s7	s8	F
19	s1	s2	s3	s4	s5	s6	s7	s8	F
20	s1	s2	s3	s4	s5	s6	s7	s8	F
21	s1	s2	s3	s4	s5	s6	s7	s8	T
22	s1	s2	s3	s4	s5	s6	s7	s8	F
23	s1	s2	s3	s4	s5	s6	s7	s8	F
24	s1	s2	s3	s4	s5	s6	s7	s8	F
25	s1	s2	s3	s4	s5	s6	s7	s8	F
26	s1	s2	s3	s4	s5	s6	s7	s8	F
27	s1	s2	s3	s4	s5	s6	s7	s8	F
28	s1	s2	s3	s4	s5	s6	s7	s8	F

我们的方法：概率差异化调试ProbDD

收益为期望删掉的元素数量
计算最大化收益的测试方案



根据测试结果，计算后验概率，
更新模型

	s1	s2	s3	s4	s5	s6	s7	s8	
	0.2500	0.2500	0.2500	0.2500	0.2500	0.2500	0.2500	0.2500	
1	s1	s2	s3	s4	s5	s6	s7	s8	F
	0.3657	0.3657	0.3657	0.2500	0.2500	0.2500	0.2500	0.3657	
2	s1	s2	s3	s4	s5	s6	s7	s8	T
	0.3657	0.3657	0.3657	0	0	0	0	0.3657	
3	s1	s2	s3	s4	s5	s6	s7	s8	F
	0.6119	0.3657	0.3657	0	0	0	0	0.6119	
4	s1	s2	s3	s4	s5	s6	s7	s8	F
	0.6119	0.6119	0.6119	0	0	0	0	0.6119	
5	s1	s2	s3	s4	s5	s6	s7	s8	T
	0.6119	0	0.6119	0	0	0	0	0.6119	
6	s1	s2	s3	s4	s5	s6	s7	s8	F
	0.6119	0	1	0	0	0	0	0.6119	
7	s1	s2	s3	s4	s5	s6	s7	s8	T
	0	0	1	0	0	0	0	0.6119	
8	s1	s2	s3	s4	s5	s6	s7	s8	F
	0	0	1	0	0	0	0	1	

每个元素有一个概率值决定其是否必要，不同元素必要的概率彼此独立，当且仅当所有必要元素都在的时候测试通过

ProbDD的理论性质

高效性

- 时间复杂度O(n)
- ddmin的复杂度O(n^2)

正确性

- 返回结果保留所需性质

最优性

- 如原问题单调，则ProbDD的结果是极小的
- 如原问题还具有无二义性，则ProbDD的结果是最小的

ProbDD验证结果

数据集

类别	实验案例	比较方法
树结构	Perses 数据集（20 个） XML 案例（10 个）	HDD
C 程序	Perses 数据集（20 个） Chisel 数据集（10 个）	Chisel

对比ddmin的结果

类别	输出结果大小（减少比例）	平均每秒移除 token 数（加速比）	运行时间（减少比例）
树结构	59.48%	2.25	63.22%
C 程序	11.51%	1.82	45.27%

ProbDD小结

- ProbDD有望全面提升现代差异化调试的效率和效果
- 概率化方法的优点使得ProbDD可行
  - 利用了领域知识估计测试通过的概率
  - 单次测试结果即可以计算后验概率
  - 可解释的概率使得可定制结束条件
- 未来还可以通过更精细的建模提高表现
- 论文被ESEC/FSE21接收，三名审稿人均给出Strong Accept的分数

软件自动化

南京大学徐家福先生定义

狭义的软件自动化指让计算机自动编写程序

广泛应用于数据处理、编译优化、测试修复、辅助开发等领域

“软件自动化是提升软件生产率的根本途径”

----徐家福先生

中国软件先驱

“One of the most central problems in the theory of programming.”

----Amir Pneuli

图灵奖获得者

两种软件自动化的方法

程序合成

程序生成

很多时候需要两者的结合

程序缺陷修复

```
/** 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;
}
```

合成新的表达式来替换掉旧的

- 既需要通过测试
- 又需要在当前上下文中概率最大

程序估计Program Estimation

输入：

- 一个程序空间*Prog*
- 一条规约*Spec*
- 一个上下文*context*
- 一个训练集*Data*，包含上下文和程序的对

输出：

- 一个程序*prog*，满足
  - $prog = \text{argmax}_{prog \in Prog \wedge prog \vdash spec} P(prog \mid context; Data)$

示例：条件补全

输入：缺一个if条件的程序和若干测试

输出：通过测试，概率最大的条件

```
public static long fibonacci(int n) {
    if ( ?? ) return n;
    else return fibonacci(n-1) + fibonacci(n-2);
}
```

E → E “>12”
  | E “>0”
  | E “+” E
  | “hours”
  | “value”
  | …

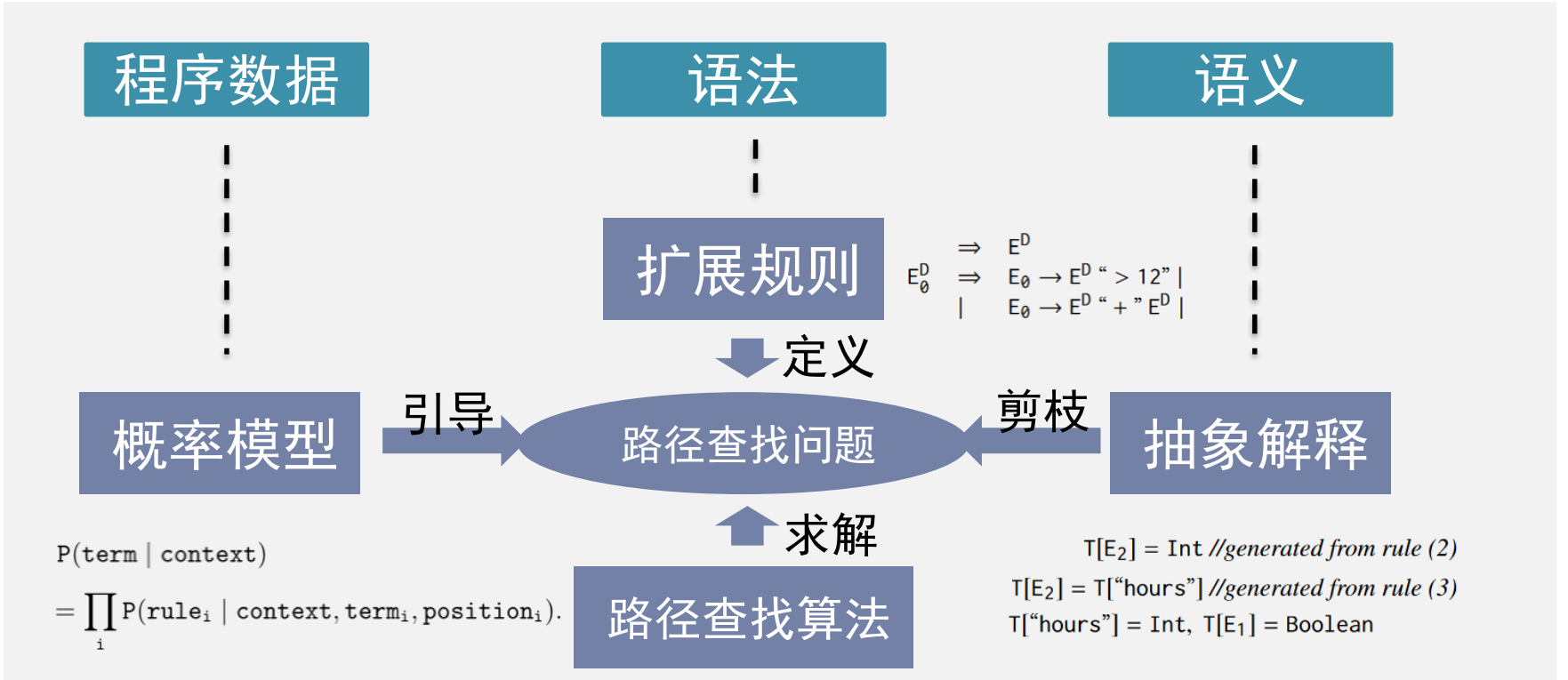
语法定义的条件空间

在程序缺陷修复中有重要应用

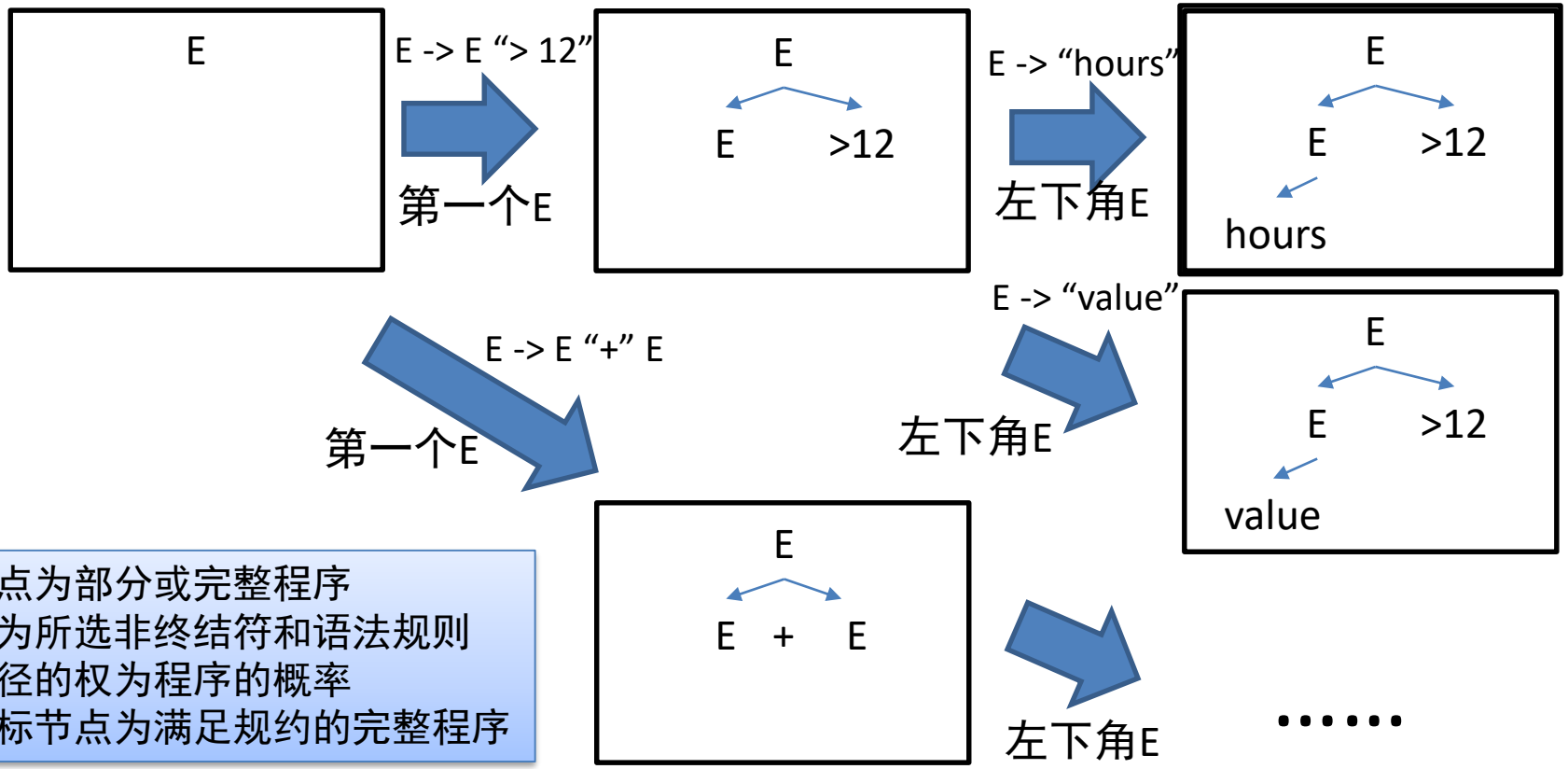
- **约一半的缺陷都和if条件有关**
- **正确合成条件有望自动修复大量缺陷**

玲珑框架

玲珑框架L2S：以路径查找算法为核心，集成概率模型和基于语义约束的剪枝



程序估计问题作为路径查找问题



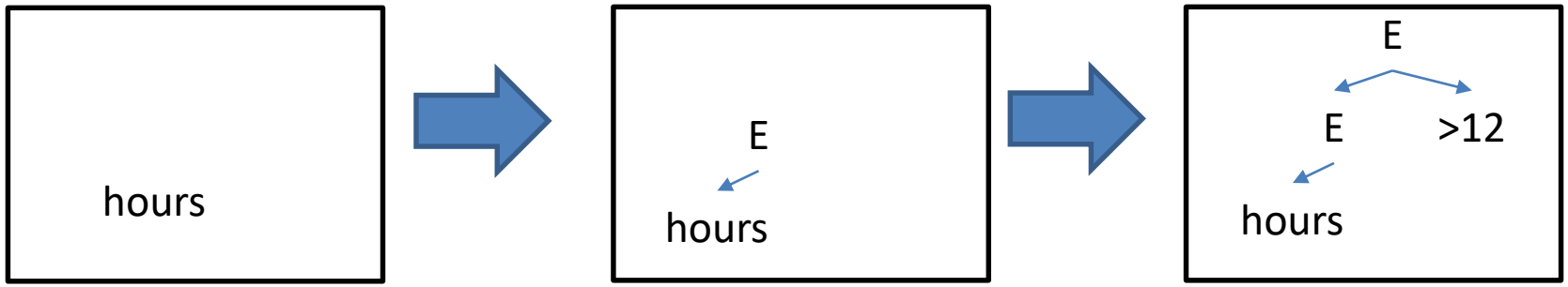
节点为部分或完整程序
边为所选非终结符和语法规则
路径的权为程序的概率
目标节点为满足规约的完整程序

玲珑框架——概率定理

$$P(prog \mid context)$$
$$= \prod_i P(\ rule_i \mid context, prog_i, position_i\ )$$

- 程序的概率只和规则选择概率有关，和AST结点展开顺序无关
- 使得计算规则概率的分类模型可以用于程序估计
- 使得算法不用遍历到达同一程序的不同路径

玲珑框架——扩展规则



- 扩展规则：上下文无关文法的扩展，用于支撑任意方向的生成
- 提出了扩展规则树，对应AST
- 提出了扩展规则树和AST的双向转换条件和算法

$$
\begin{aligned}
&\langle E \rightarrow \text{“}hours\text{”}, && \bot\rangle\\
&\langle E \rightarrow \text{“}value\text{”}, && \bot\rangle\\
&\langle E \rightarrow E\ \text{“}>12\text{”}, && 1\rangle\\
&\langle E \rightarrow E\ \text{“}+\text{”}\ E, && 1\rangle\\
&\langle T \rightarrow E, && 1\rangle\\
&\langle E \rightarrow E\ \text{“}>12\text{”}, && 0\rangle\\
&\langle E \rightarrow E\ \text{“}+\text{”}\ E, && 0\rangle\\
&\langle E \rightarrow \text{“}hours\text{”}, && 0\rangle\\
&\langle E \rightarrow \text{“}value\text{”}, && 0\rangle
\end{aligned}
$$

玲珑框架——剪枝方法

基于抽象解释设计语法上的静态预分析算法，可以计算出动态剪枝条件，快速剪枝不能满足规约的部分程序

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  $V[E] = V[x] \cup V[x]$

玲珑框架应用——从自然语言生成代码

已有方法主要采用end-to-end的神经网络结构，如RNN, LSTM

RNN/LSTM有长依赖问题

- 长依赖问题Long dependency problem: 无法处理距离较远的依赖关系
- 程序中长依赖很多，如当前使用的变量可能很早之前声明


```
[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)
```

结果1[AAAI19]

玲珑框架允许我们采用任意分类模型
采用长依赖问题较小的CNN

Benchmark: HearthStone

Model	StrAcc	Acc+	BLEU
LPN (Ling et al. 2016)	6.1	–	67.1
SEQ2TREE (Dong and Lapata 2016)	1.5	–	53.4
SNM (Yin and Neubig 2017)	16.2	~18.2	75.8
ASN (Rabinovich, Stern, and Klein 2017)	18.2	–	77.6
ASN+SUPATT (Rabinovich, Stern, and Klein 2017)	22.7	–	79.2
Our system	**27.3**	**30.3**	**79.6**

结果2[AAAI20]

将CNN换成Transformer

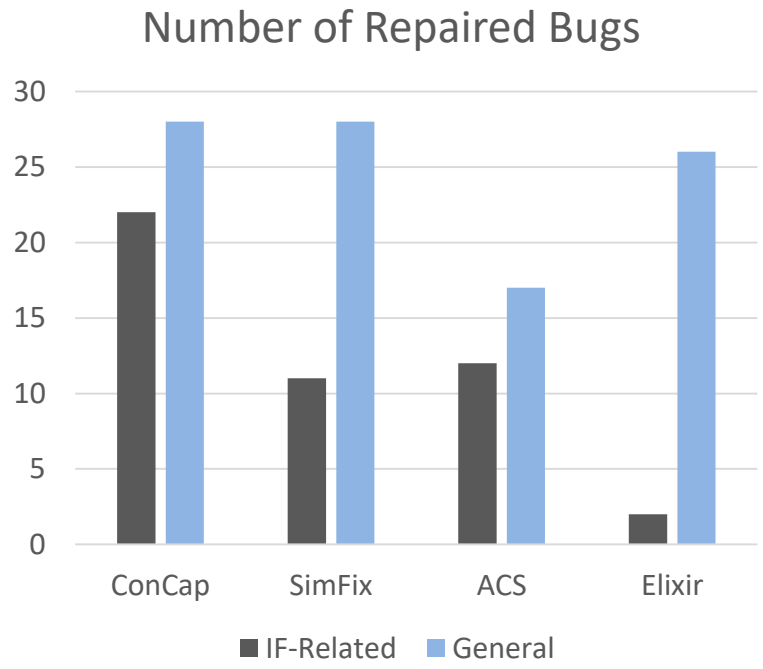
- Transformer：2017年新提出来的网络体系结构
- L2S允许灵活替换不同的统计模型

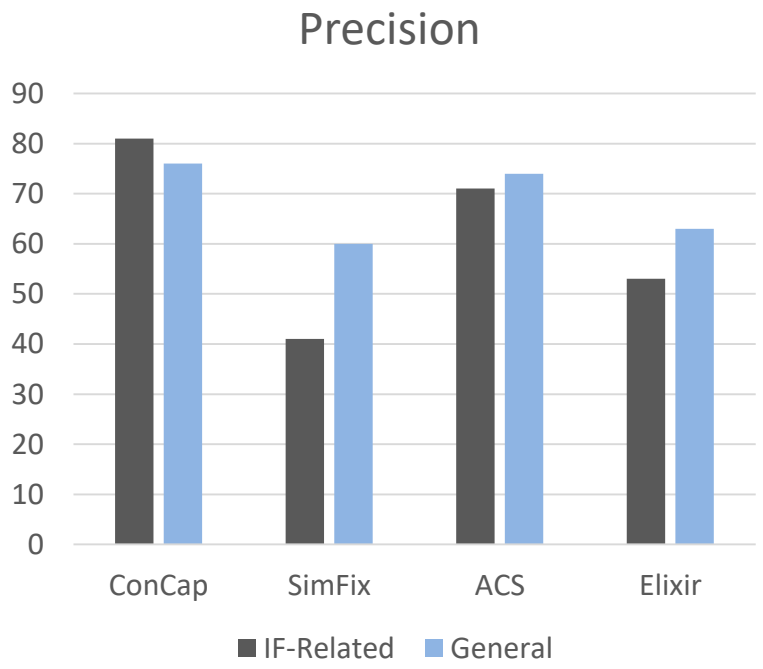
	Model	StrAcc	Acc+	BLEU
Plain	LPN (Ling et al., 2016)	6.1	–	67.1
	SEQ2TREE (Dong and Lapata, 2016)	1.5	–	53.4
	YN17 (Yin and Neubig, 2017)	16.2	~18.2	75.8
	ASN (Rabinovich et al., 2017)	18.2	–	77.6
	ReCode (Hayati et al., 2018)	19.6	–	78.4
	CodeTrans-A	**25.8**	**25.8**	**79.3**
Structured	ASN+SUPATT (Rabinovich et al., 2017)	22.7	–	79.2
	SZM19 (Sun et al., 2019)	27.3	30.3	79.6
	CodeTrans-B	**31.8**	**33.3**	80.8

采用玲珑框架修复条件缺陷［TOSEM投稿］

采用xgboost和类型语义约束来解决条件合成问题

Benchmark: Defects4J





条件缺陷修复数量和准确率达到最高
8个没有被任何别的工作修复过的全新缺陷

基于玲珑框架的程序缺陷修复-最新结果

采用神经网络来构造概率模型，并用修改操作定义了程序空间

Table 2: Comparison without Perfect Fault Localization

Project	jGenProg	HDRepair	Nopol	CapGen	SketchFix	FixMiner	SimFix	TBar	DLFix	PraPR	AVATAR	Recoder
Chart	0/7	0/2	1/6	4/4	6/8	5/8	4/8	**9/14**	5/12	4/14	5/12	8/14
Closure	0/0	0/7	0/0	0/0	3/5	5/5	6/8	8/12	6/10	12/62	8/12	**17/31**
Lang	0/0	2/6	3/7	5/5	3/4	2/3	**9/13**	5/14	5/12	3/19	5/11	**9/15**
Math	5/18	4/7	1/21	12/16	7/8	12/14	14/26	**18/36**	12/28	6/40	6/13	15/30
Time	0/2	0/1	0/1	0/0	0/1	1/1	1/1	1/3	1/2	0/7	1/3	**2/2**
Mockito	0/0	0/0	0/0	0/0	0/0	0/0	0/0	1/2	1/1	1/6	**2/2**	**2/2**
Total	5/27	6/23	5/35	21/25	19/26	25/31	34/56	42/81	30/65	26/148	27/53	**53/94**
P(%)	18.5	26.1	14.3	**84.0**	73.1	80.6	60.7	51.9	46.2	17.6	50.9	56.4

In the cells, x/y:x denotes the number of correct patches, and y denotes the number of patches that can pass all the test cases.

神经网络修复首次超过传统修复的效果

录用于ESEC/FSE21，被审稿人提名最佳论文候选

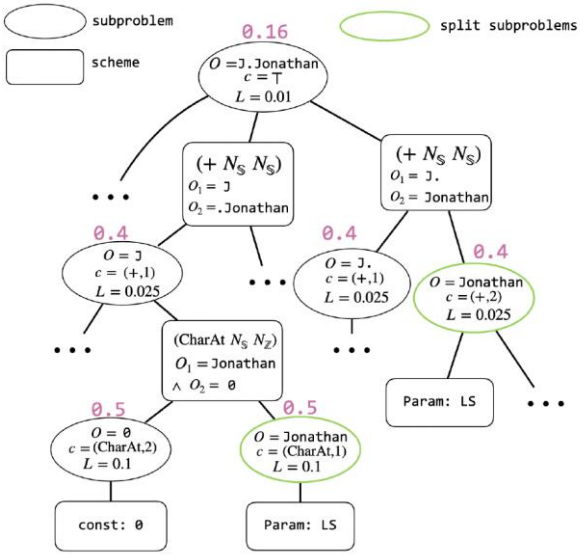
后续发展——结合动态规划

玲珑框架可以看做是将概率加入枚举的程序合成算法

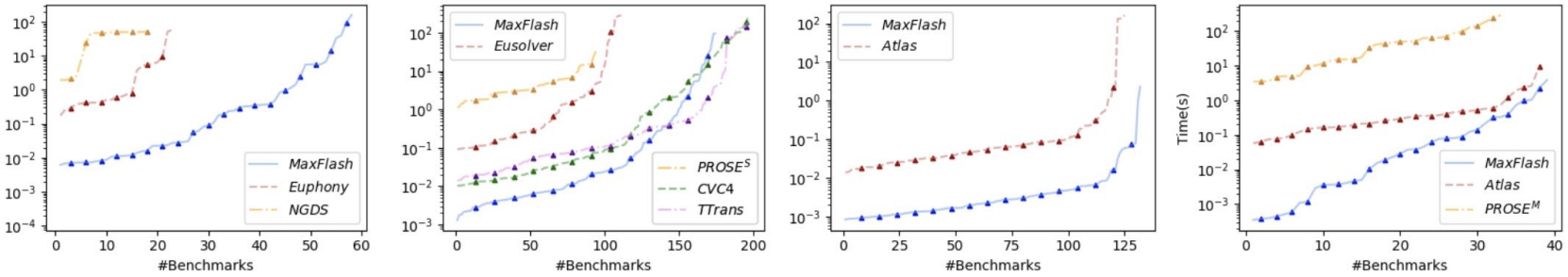
现代程序合成算法采用动态规划来达到较高效率

MaxFlash：
将结构概率与动态规划结合。

动态规划基于局部子问题，结构概率基于全局的结构。



相比于已有的程序综合技术，**MaxFlash**（1）取得了×4–×2080倍的平均加速比；（2）在500ms的响应时间内解决了更多的综合任务；（3）更加节约空间。



发表于OOPSLA2020。

玲珑框架小结

- 概率化方法给玲珑框架带来一系列优点
  - 同时考虑训练数据的统计信息和语法语义的逻辑约束信息
  - 各部分可以灵活变换
    - 可以采用任意带概率计算的分类模型
    - 程序空间可以按需定义
- 程序估计问题在很多领域都有潜在应用

总结

参数化 → 概率化

概率化是未来趋势

- 概率化方法相比参数化方法有多种优势
- 概率化方法可解释性特点使得方法可持续改进
- 蓬勃发展的概率编程语言也方便了方法实现

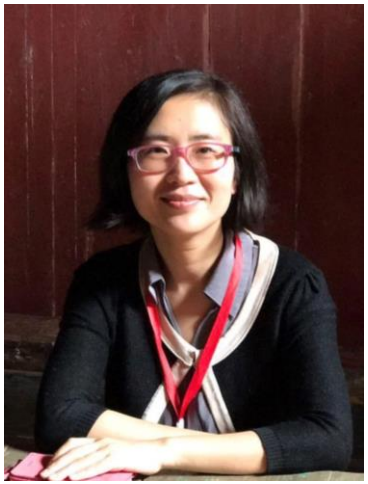
“我猜想概率化、基于采样或优化的统计推断是软件分析和测试调试的发展方向。”

——南京大学马晓星教授

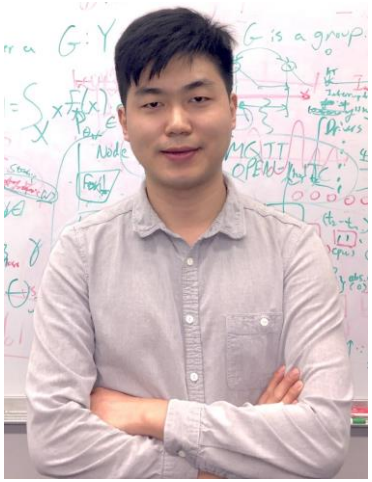
感谢北京大学的合作者们！

胡振江教授

张路教授



郝丹副教授



张昕助理教授

个人简介：张昕

教育工作经历

2020-现在	北京大学 计算机科学与技术系	助理教授兼特聘研究员 国家级青年人才计划入选者
2017-2020	美国麻省理工学院 计算机与人工智能实验室	博士后 合作导师：Armando Solar-Lezama
2011-2017	美国佐治亚理工学院 计算机学院	博士生 导师：Mayur Naik
2007-2011	上海交通大学软件学院	本科生

研究方向

机器学习 ➡ 程序分析

首次提出逻辑和概率结合的程序分析，赋予分析学习和适应的能力

PLDI'14杰出论文, FSE'15杰出论文, OOPSLA'16, OOPSLA'17,

开发了针对程序分析的概率推理算法，效率远超现有算法，能解百万级变量问题

POPL'16 , CP'16, AAAI'16, SAT'16

程序分析、语言 ➡ 机器学习

高可信人工智能分析

可解释性 NeurIPS'18

公平性 OOPSLA'19

概率编程

因果推理 ICML'21

基于分布性质的推理 In Submission