设计一个
缺陷自动修复系统

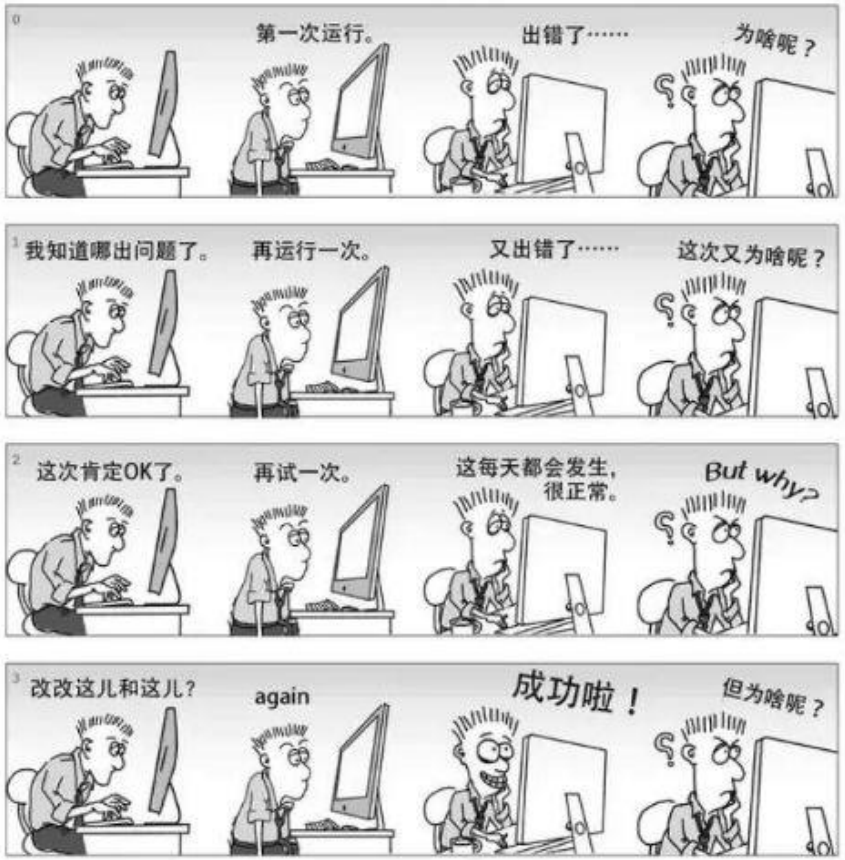
熊英飞

北京大学

2019

程序员的人生

- 就是写Bug和修Bug交织在一起的悲歌



到底花了多少时间修Bug?

- 软件维护35.6%的时间是在修Bug[1]
  - 软件维护成本通常认为占软件成本的90%
- 开发人员花在修复上的时间占全部开发时间一半左右[2]
- 开发团队可能没有足够资源修复所有缺陷[3]
- 软件在包含已知缺陷的情况下发布[4]

[1] B. P. Lientz, E. B. Swanson, and G. E. Tompkins, “Characteristics of application software maintenance,” Commun. ACM, vol. 21, no. 6, pp. 466–471, 1978
[2] Britton et al. Quantify the time and cost saved using reversible debuggers. Cambridge report, 2013
[3] J. Anvik, L. Hiew, and G. C. Murphy, “Coping with an open bug repository,” eXchange, 2005, pp. 35–39
[4] B. Liblit, A. Aiken, A. X. Zheng, and M. I. Jordan, “Bug isolation via remote program sampling,” in PLDI, 2003, pp. 141–154

缺陷自动修复的形式

- 输入：一个程序和其正确性约束，并且程序不满足正确性约束
- 输出：一个补丁，可以使程序满足约束

研究和实践中考虑最广泛的正确性约束——
软件项目中的测试

缺陷自动修复的作用

- Yida Tao, Jindae Kim, Sunghun Kim, Chang Xu: Automatically generated patches as debugging aids: a human study. SIGSOFT FSE 2014: 64-74
  - 当程序员有高质量的补丁做辅助的时候，修复正确率大幅提高，修复时间小幅减少
  - 修复正确率大幅提高=>未来修复时间大幅减少

缺陷自动修复的学术价值

- “自动程序生成是程序设计理论最核心的问题。”
  - Amir Pnueli, 图灵奖获得者
  - On the synthesis of a reactive module, POPL 1989
- “程序缺陷修复和自动程序生成是等价问题。”
  - 林惠民院士，雁栖湖会议2018
- 给定规约，给定空白程序，如果能修复空白程序相对规约的缺陷，我们就针对规约自动生成了程序

缺陷修复重要质量指标

- 正确率：产生的补丁中有多少是正确的
  - 决定技术是否可用
- 召回率：在所有的缺陷中有多少是能够修复的
  - 决定技术的应用效果
- 修复效率：每个缺陷要花多少时间修复
  - 决定技术的应用场景

发展历史-史前阶段

- 时间：-2009
- 修复一些特定类型的缺陷
  - 演化缺陷
- 修复一些特定类型的程序或软件制品
  - 布尔程序
  - 软件模型

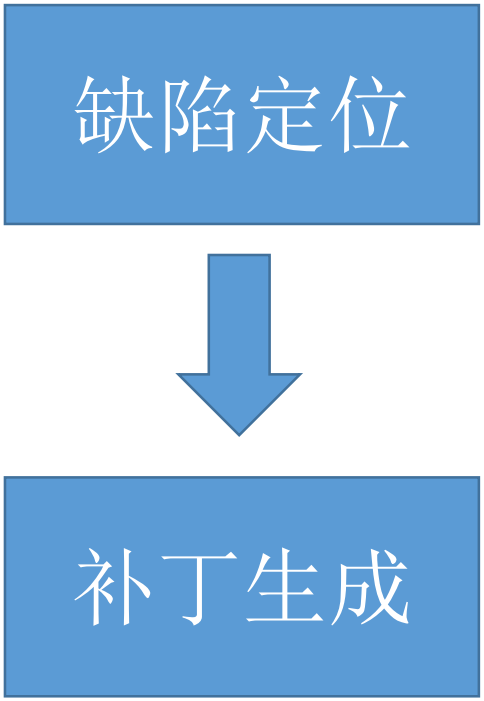
发展历史-GenProg时代

- Automatically finding patches using genetic programming.
  - Westley Weimer, ThanhVu Nguyen, Claire Le Goues, Stephanie Forrest. ICSE 2009: 364-374
  - 基本思路：天下程序一大抄
  - 随机从别的地方复制语句替换/插入到当前位置，或删除当前语句
  - 用遗传算法组合这些基本操作
- A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each.
  - Claire Le Goues, Michael Dewey-Vogt, Stephanie Forrest, Westley Weimer. ICSE 2012: 3-13
  - 105个大型程序上的缺陷
  - 修复了一半
- 大量后续工作： AutoFix, RSRepair, NOPOL, relifix, SemFix, DirectFix, PAR…

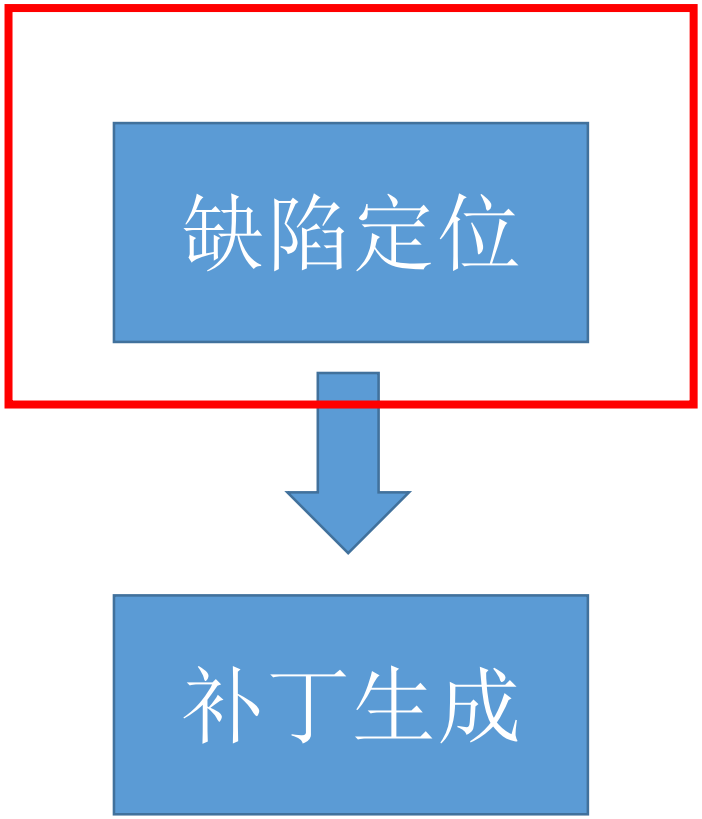
发展历史-后GenProg时代

- An analysis of patch plausibility and correctness for generate-and-validate patch generation systems.
  - Zichao Qi, Fan Long, Sara Achour, Martin C. Rinard. ISSTA 2015
  - GenProg修复的55个缺陷中只有2个是正确的
  - 通过测试≠完整修复
- 后期修复技术将正确率作为重要指标
  - 通过和程序员的修复对比，等价的算作正确的
  - 大量技术致力于提供高正确率的修复：Prophet, Angelix, HDRepair, ACS, Anti-Pattern, Elixir, JAID, CapGen, Genesis…
- Precise condition synthesis for program repair
  - Yingfei Xiong, Jie Wang, Runfa Yan, Jiachen Zhang, Shi Han, Gang Huang, Lu Zhang. ICSE 2017: 416-426
  - 采用数据驱动的方式修复缺陷
  - 正确率提高到70%以上

现代缺陷修复系统工作流程



现代缺陷修复系统工作流程



基于测试的错误定位

- 输入：
  - 软件系统的源码
  - 一组测试，至少有一个没有通过
- 输出：
  - 一个可能有错误的程序元素列表，根据出错概率排序
- 程序元素可以定义在不同级别上
  - 表达式
  - 语句
  - 方法
  - 类
  - 文件
  - ……

基于标准测试覆盖的错误定位

基于频谱的错误定位

- 使用最广泛的自动化错误定位方法
  - 形式简单，效果较好
- 程序频谱(Program Spectrum)
  - 最早由威斯康星大学Tom Reps于1997年在处理千年虫问题时发明
  - 指程序执行过程中的统计量
- 基于频谱的错误定位
  - 佐治亚理工James Jone, Mary Jean Harrold等人2002把Tom Reps的方法通用化成通用调试方法
  - 主要用到的频谱信息为测试覆盖信息

基于频谱的错误定位

- 基本思想
  - 被失败的测试用例执行的程序元素，更有可能有错误
  - 被成功的测试用例执行的程序元素，更有可能没有错误
- 程序元素可以定义在不同的粒度上
  - 基本块
  - 方法
  - 类
  - 文件
  - 可以是语句、表示式吗？

例子

		T15	T16	T17	T18
1	int count; int n; Ele *proc; List *src_queue, *dest_queue; if (prio >= MAXPRIO) /*maxprio=3*/	●	●	●	●
2	{return;}	●			

3	src_queue = prio_queue[prio];
	dest_queue = prio_queue[prio+1];
	count = src_queue->mem_count;
	if (count > 1) /* Bug*//* supposed : count>0*/ {
			●	●	●

4	n = (int) (count*ratio + 1);
	proc = find_nth(src_queue, n);
	if (proc) {
			●	●	

5	src_queue = del_ele(src_queue, proc);
	proc->priority = prio;
	dest_queue = append_ele(dest_queue, proc); }
			●	●	

	Pass/Fail of Test Case Execution :	**Pass**	**Pass**	**Pass**	**Fail**

计算程序元素的怀疑度

- a_{ef}：执行语句a的失败测试的数量，a_{nf}：未执行语句a的失败测试的数量
- a_{ep}：执行语句a的通过测试的数量，a_{np}：未执行语句a的通过测试的数量

- Tarantula: $\frac{a_{ef}}{a_{ef}+a_{nf}} / \left(\frac{a_{ef}}{a_{ef}+a_{nf}} + \frac{a_{ep}}{a_{ep}+a_{np}} \right)$
- Jaccard: $\frac{a_{ef}}{a_{ef}+a_{nf}+a_{ep}}$
- Ochiai: $\frac{a_{ef}}{\sqrt{(a_{ef}+a_{nf})(a_{ef}+a_{ep})}}$
- D*: $\frac{a_{ef}^{*}}{a_{nf}+a_{ep}}$，*通常设置为2或者3
- Naish1: $\begin{cases} -1 & a_{nf} > 0 \\ a_{np} & a_{nf} = 0 \end{cases}$

哪个公式是最好的公式？

- 实验验证
  - 在不同对象上的实验结果并不一致
  - 早期实验认为Ochiai最好，D*论文认为D*最好
  - 最新在Java的真实缺陷上的研究认为不同公式之前并无统计性显著差异
    - 语句级别Top-5能平均能定位准18%，Top10为27%
- 理论研究
  - 武汉大学谢晓园等人理论上证明了Naish1优于Ochiai, Ochiai优于Jaccard, Jaccard优于Tarantula，但不存在单一最佳公式
  - 新加坡管理大学David Lo等人做实验验证出和谢晓园不一致的结论

基于测试的错误定位

基于状态覆盖的错误定位

程序元素的粒度如何选择？

- 粒度越细
  - 缺陷定位的结果越精细，对测试信息的利用越精确
  - 单个元素上覆盖的测试数量越少，统计显著性越低

- 常见情况举例
  - 方法级别
  - 基本块级别

能否比语句更精细？

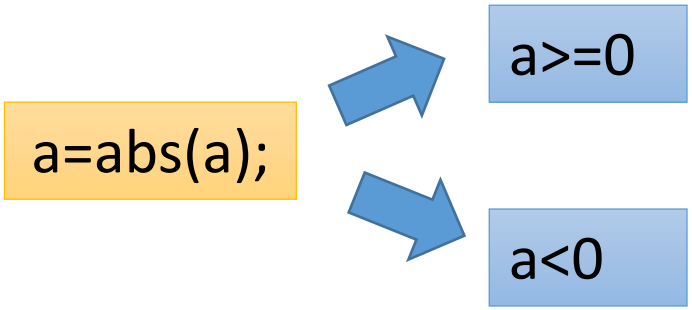
- 状态级别：程序的每个执行状态作为一个元素
  - 定位结果最精细，对测试的利用最充分
  - 几乎不会有两个测试覆盖同样的状态
- 能否找到一个折中方案？

基于状态覆盖的错误定位代表方法

- 统计性调试
  - 斯坦福Ben Liblit 和Alex Aiken等人于2003年提出
- 概率依赖图
  - 佐治亚理工Mary Jean Harrold等人于2010年（Mary去世前三年）提出
- Savant
  - 新加坡管理大学David Lo等人于2016年提出

基于状态覆盖的错误定位

- a=abs(a);
- ....
- If (...) {
- b=sqrt(a);
- }



- 该语句执行完系统的状态可以分成两组抽象状态
  - 通过的测试只有a>=0的状态。
  - 只有失败的测试有a<0的状态。
- 可以判断出a<0是缺陷状态，引入该状态的语句为缺陷语句。

预定义谓词作为抽象状态

- 定义一些常见谓词(predicate)，每个谓词的不同状态把具体状态划分成抽象状态
  - 不同谓词形成的划分可以重叠
- 常见谓词
  - 对整形变量a
    - a>0
    - a<0
    - a==0
  - 对布尔变量b
    - b==true
    - b==false
  - 对对象o
    - o==null
    - o!=null

如何给抽象状态的出错可能性打分

- 基于频谱的公式理论上可以继续使用
  - 但现在居然还没人试过，诡异……
- 打分方法：统计性调试公式
  - 假设令predicate为真的状态为a，为假的状态为b
  - $$\frac{2}{\frac{1}{\frac{a_{ef}}{a_{ep}+a_{ef}}-\frac{a_{ef}+b_{ef}}{a_{ep}+a_{ef}+b_{ep}+b_{ef}}}+\frac{\log F}{\log a_{ef}}}$$

基于变异的错误定位

变异分析

- 变异：对程序的任意随机修改
- 变异分析：收集变异后程序上原测试用过与否的信息的分析
- 变异分析被广泛应用于测试领域来衡量一个测试集的好坏
  - 如果一个测试集中任意测试在一个变异后的程序上执行失败，称为该变异体被这个测试杀死
  - 能杀死越多变异体的测试集越好

常见变异测试工具

- C
  - Milu
- Java
  - MuJava：基础变异测试工具，支持变异算子较完善
  - Javalanche：支持Mutation Schemata的加速
  - Major：支持预先过滤测试执行的加速，支持变异算子较少
  - PIT：商业工具，功能最完善速度最快

关于变异的假设

- 假设1：变异错误语句时
  - 失败测试用例输出发生变化的概率>通过测试用例输出发生变化的概率
- 假设2：变异正确语句时
  - 失败测试用例输出发生变化的概率<通过测试用例输出发生变化的概率
- 假设3：导致失败测试变成通过的概率
  - 变异错误语句时>变异正确语句时
- 假设4：导致通过测试变成失败的概率
  - 变异正确语句时>变异错误语句时

Metallaxis

MUSE

Metallaxis

- 卢森堡大学的Yves Le Traon教授等人2012年提出
- m:变异体，m_f：输出发生变化的失败测试数，m_p：输出发生变化的通过测试数，F：原失败测试总数
- 变异体可疑度公式
  - $\frac{m_f}{\sqrt{F*(m_f+m_p)}}$
  - 与Ochiai类似，但主要考虑输出的变化
- 程序元素可疑度为该元素上的最高变异体的可疑度

MUSE

- 韩国科学技术院Moonzoo Kim和英国UCL大学Shin Yoo于2014年提出
- m:变异体，m_{f2p}：m上从失败变成通过的测试数，m_{p2f}：m上从通过变成失败的测试数
- 变异体可疑度公式
  - $m_{f2p} - m_{p2f} \frac{\sum_m m_{f2p}}{\sum_m m_{p2f}}$
- 程序元素可疑度为该元素上的变异体的可疑度平均

基于变异vs基于频谱

- 在基于频谱的错误定位中，不同测试只要覆盖了语句，对结果的效果就是相同
- 但不同测试受同一个语句的影响是不同的
  - 不同测试触发错误的概率不同
  - 不同测试传播错误的概率不同
  - 不同测试捕获错误的概率不同
- 基于变异的错误定位实际依靠变异捕获了测试和语句之间的关系

基于测试的错误定位

构造正确执行状态

动机

- 在MUSE中，如果有一个变异让失败的测试通过，同时通过的测试仍然通过，那么该变异有最高的怀疑度
- 换句话说，该变异很可能是正确的补丁
- 动机：直接分析出这样的变异，然后将能产生出的语句当作怀疑度最高的语句
- 困难：直接分析出比较困难
- 解决方案：不分析出变异本身，只分析出该变异对系统状态的影响

谓词翻转Predicate Switching

- 2006年由普度的张翔宇教授提出
- 假设出错的是一个布尔表达式
  - 不考虑表达式的副作用
- 该表达式修改后，必然在原失败测试中至少一次求值返回翻转的结果
  - true -> false
  - false -> true
- 依次翻转失败测试中表达式求值结果，如果测试通过，则说明对应表达式可能有错误

天使调试Angelic Debugging

- 2013年由华盛顿大学的Emina Torlak提出
- 如何把谓词翻转从布尔表达式扩展到任意表达式上？如int, float, double等
- 天使性条件：存在常量c（天使值）把表达式的求职结果替换成c，失败的测试变得通过
- 是否满足天使性条件就代表表达式很可能有缺陷呢？

天使性条件

f(a):
b = a+1;
c = b+1;
d = c++;

失败测试：
f(1);
assert(d=4);

以上每个表达式都满足条件

完整天使调试

- 基础天使调试条件对应原来目标的前一半：失败的测试变得通过
- 利用后一半：通过的测试仍然通过
- 假设：对表达式进行修改后，表达式在所有测试中都会得到不同的结果
  - 比较强的假设，但对数值型表达式有较大概率成立
- 灵活性条件：对于所有通过的测试中的每一次表达式求值，都可以把求值结果换成一个不同的值，并且测试仍然通过。
- 可疑语句需要同时具有天使性和灵活性

完整天使调试

```
f(a):
b = a+1;
c = b+1;
d = c++;
```

失败的测试：

```
f(1);
assert(d=4);
```

通过的测试：

```
f(2);
assert(c=4);
```

只有c++是可疑的表达式

完整天使调试

```
f(a):
b = a+1;
c = b+1;
d = c++;
```

为什么谓词翻转不需要灵活性条件？

失败的测试：

```
f(1);
assert(d=4);
```

通过的测试：

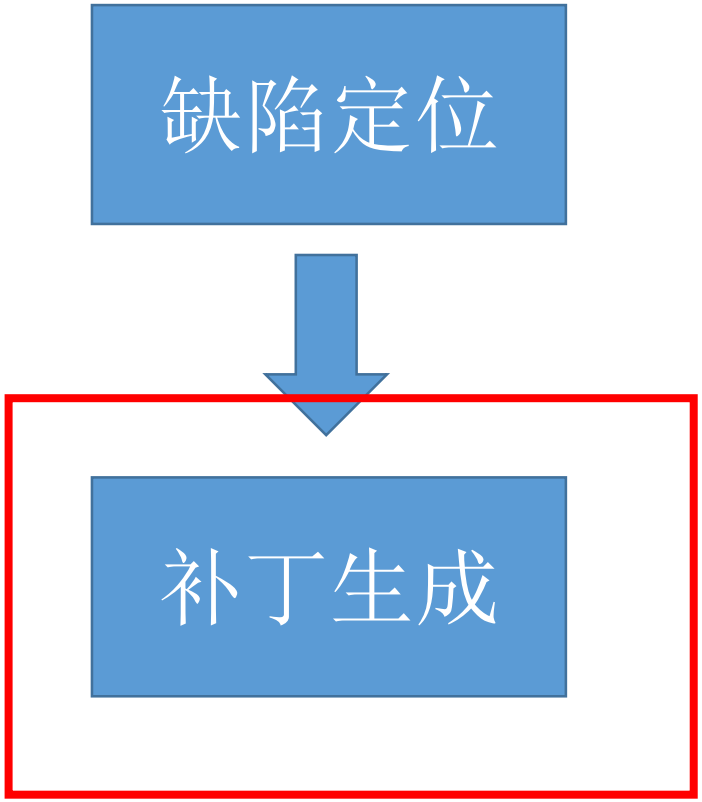
```
f(2);
assert(c=4);
```

只有c++是可疑的表达式

如何判断天使性和灵活性？

- 采用符号执行
- 首先选定表达式
- 将表达式的返回值用符号v替换
- 从该表达式所在语句开始符号执行，考虑所有路径（循环最多执行n次），并收集路径约束和最后test oracle形成的约束求解
- 对于通过的测试，还要添加约束$v \neq c$，其中c是原来运行的结果
- 由于符号执行的开销，天使调试无法应用到大型程序上

现代缺陷修复系统工作流程



Program Estimation

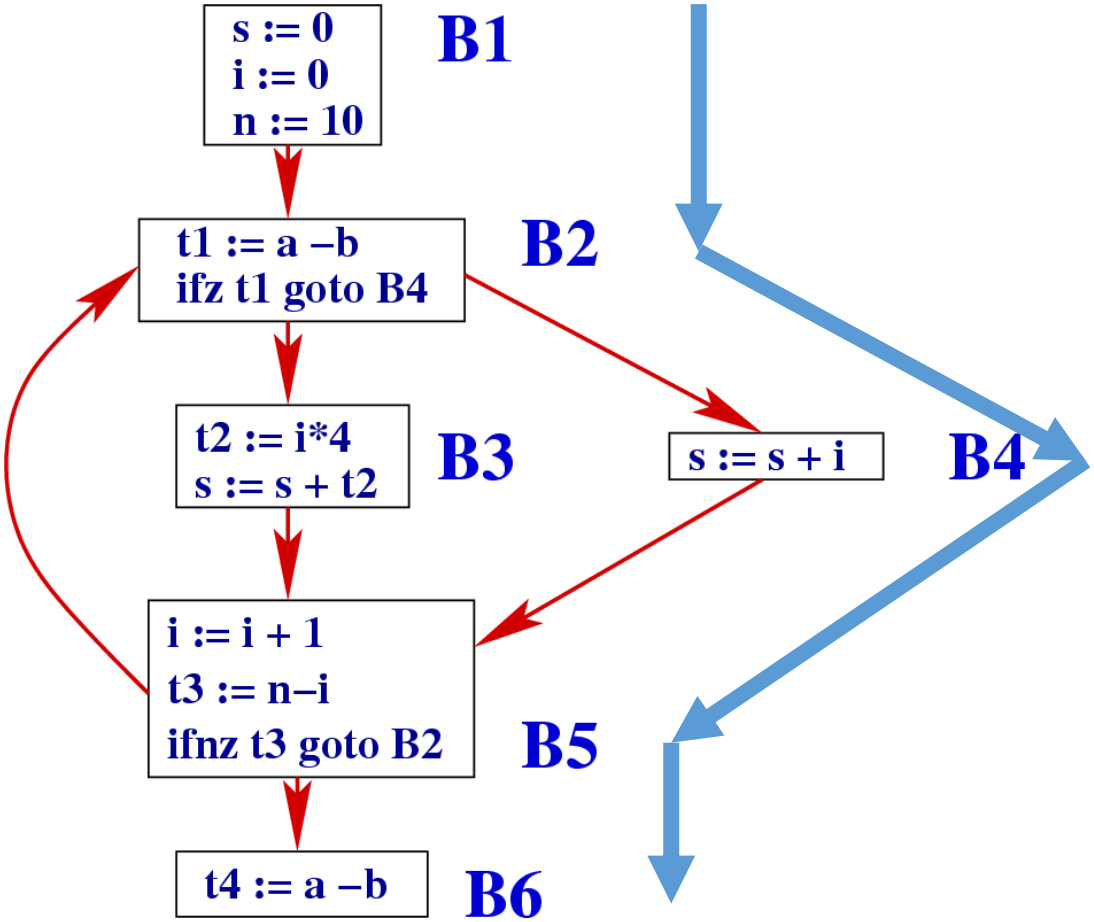
- Input:
  - program space G
  - specification S
  - context C
  - a training set T of context-program pairs
- Output:
  - program P
  - such that $P \in G \wedge P \mapsto S \wedge \Pr(P \mid C)$
  - where $\Pr$ represents the probability learned from T

Application – 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

Application – Testing

Synthesize a unit test to cover a path



Application – Reducing Duplicated Programming


```
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)
```

Application – Program Repair

```
/** 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;
}
```

Synthesize an expression to replace the buggy one

Challenges

- How to estimate the probability $P(prog \mid context)$?

- How to find program s such that $prog \in Prog$ and $P(prog \mid context)$ is the largest?

Learning to synthesis (L2S)

- A general framework to address program estimation

- Combining four tools
  - **Rewriting rules**: defining a search problem
  - **Constraint solving**: pruning off invalid choices in each step
  - **Machine-learned models**: estimating the probabilities of choices in each step
  - **Search algorithms**: solving the search problem

Example: Condition Completion

- Given a program without a conditional expression, completing the condition

```
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”
| …

Space of Conditions

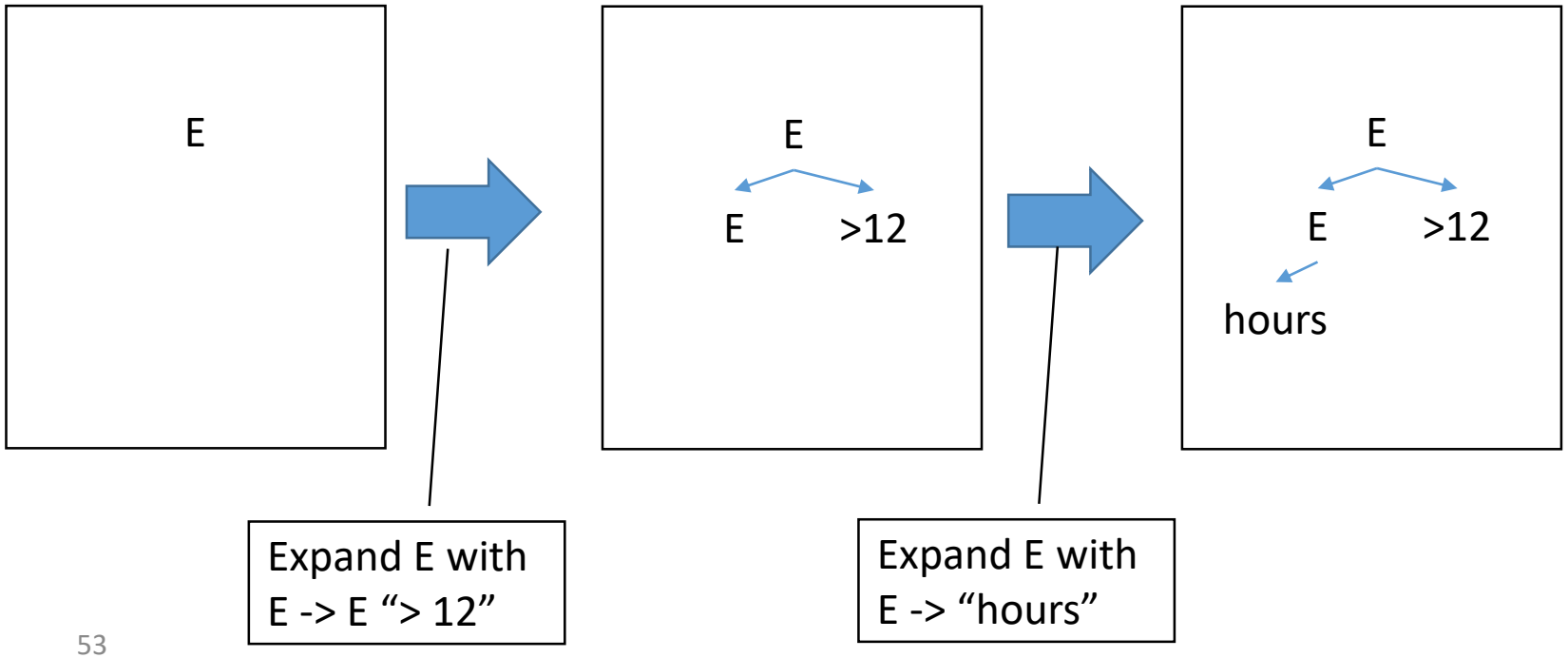
- Useful in program repair
  - Many bugs are caused by incorrect conditions
  - Existing work could localize the faulty condition
  - Can we generate a correct condition to replace the incorrect one?

Challenge 1: Estimating the Probability

- Idea: Using machine learning
  - To train over a set of programs and their contexts
- Problem: machine learning usually works for classification problems
  - where the number of classes are usually small
- Idea: turn the generation problem into a set of classification problem along the grammar

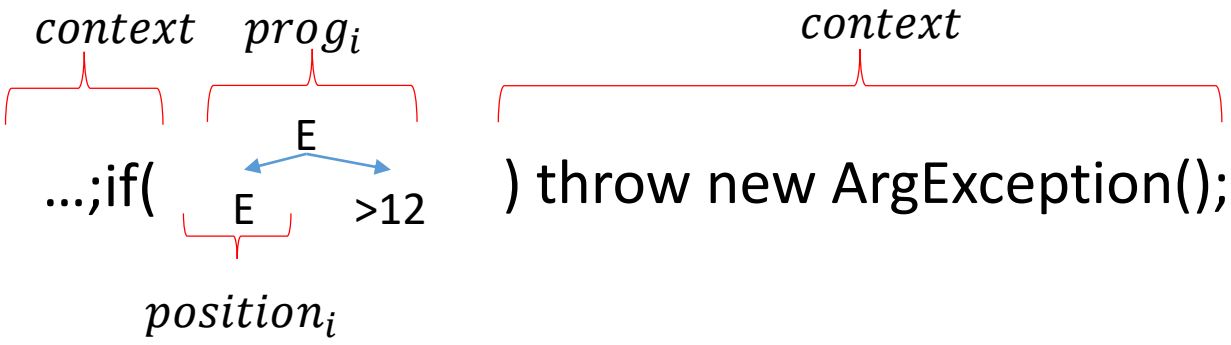
Decomposing Generation

- In each step, we estimate the probabilities of the rules to expand the left-most non-terminal
  - A classification problem



Probability of the program

- $P(prog \mid context) = \prod_i P(rule_i \mid context, prog_i, position_i)$
  - $context$: The context of the program
  - $prog_i$: The AST generated at the ith step
  - $position_i$: The non-terminal to be expanded at the ith step
  - rule: the chosen rule at the ith step
  - $prog$: the complete program

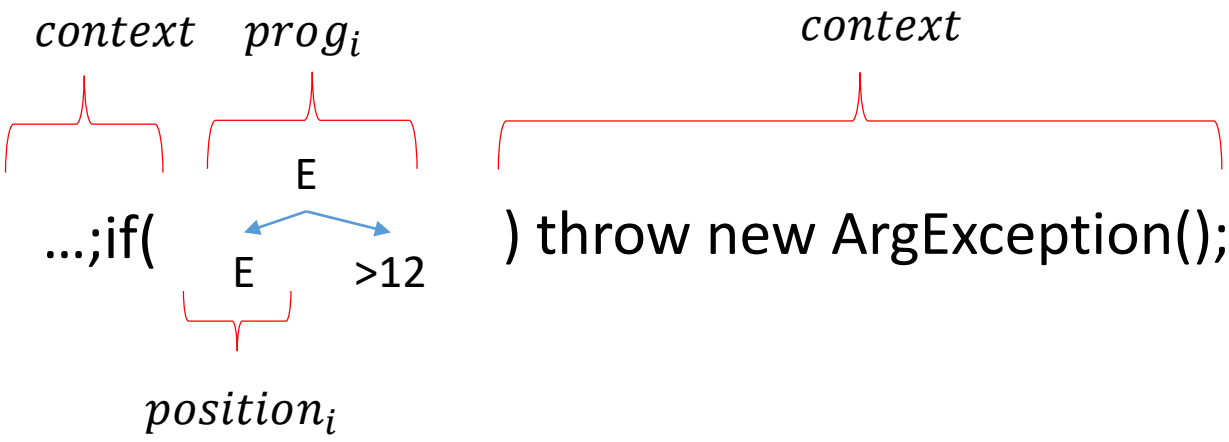


Training models

- Train a model for each non-terminal
  - to classify rules expanding this non-terminal
- Training set preparation
  - The original training set:
    - A set of programs
    - Their contexts
  - Decomposing the training set:
    - Parse the programs
    - Extract the rules chosen for each non-terminal

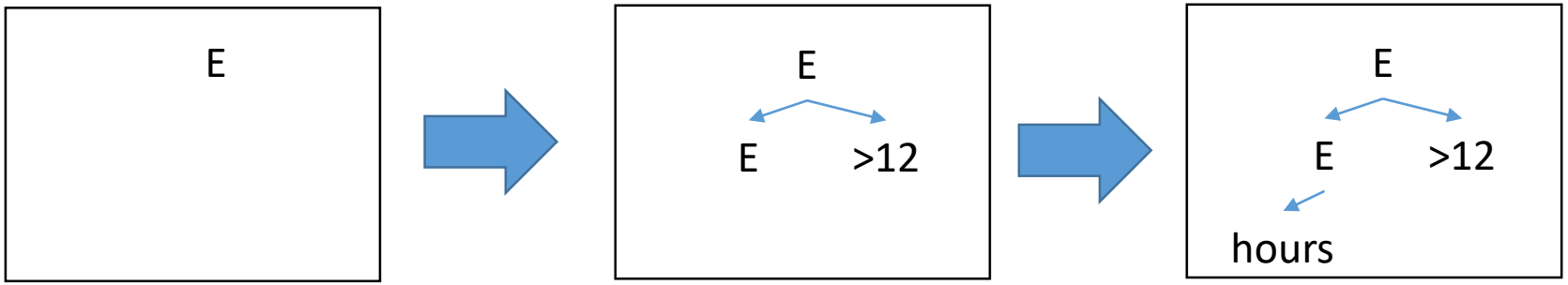
Feature Engineering

- Extract features from
  - $context$: The context
  - $prog_i$: The generated partial AST
  - $position_i$: The position of the node to be expanded

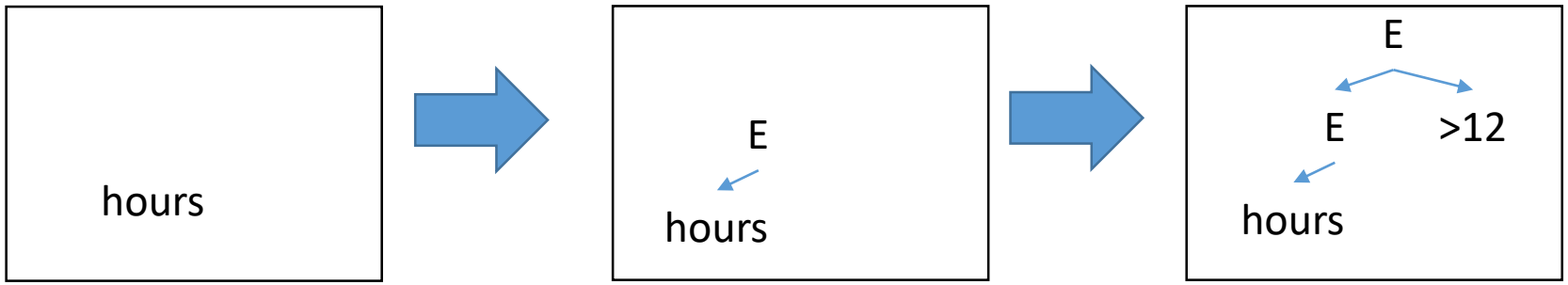


Can we use a different expansion order?

- Top-down



- Bottom-up



The order may greatly affect the performance of L2S.

Annotations

- Introduce annotations to symbols
  - E^D indicates E can be expanded downward
  - E^U indicates E can be expanded upward
  - E^{UD} indicates E can be expanded in both directions

From Grammar to Rewriting Rules

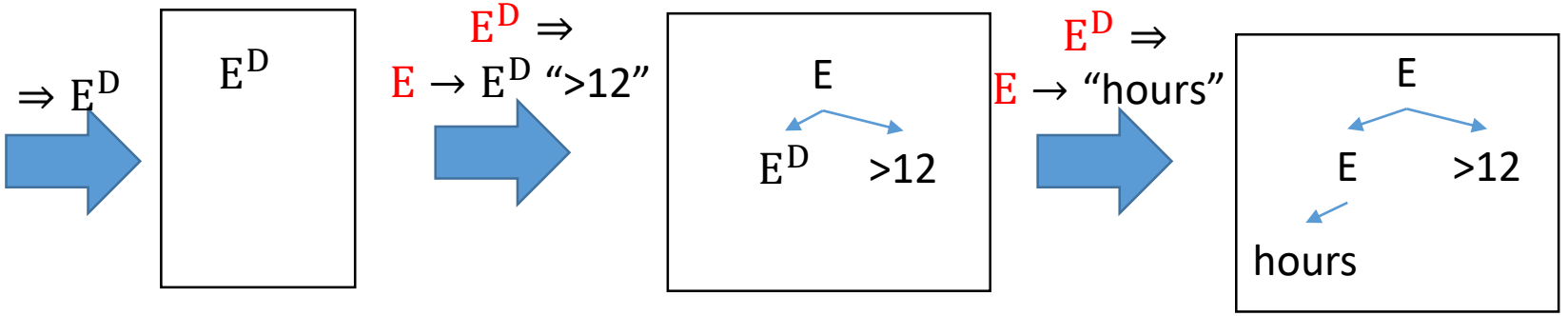
Grammar	Top-down Rules	Bottom-up Rules
$\mathrm{E} \rightarrow \mathrm{E}$ “+” E	$\mathrm{E}^{\mathrm{D}} \Rightarrow \mathrm{E} \rightarrow \mathrm{E}^{\mathrm{D}}$ “+” E^{D}	$\mathrm{E}^{\mathrm{U}} \Rightarrow \mathrm{E}^{\mathrm{U}} \rightarrow \mathrm{E}$ “+” E^{D} $\mathrm{E}^{\mathrm{U}} \Rightarrow \mathrm{E}^{\mathrm{U}} \rightarrow \mathrm{E}^{\mathrm{D}}$ “+” E
$\mathrm{E} \rightarrow \mathrm{E}$ “>12”	$\mathrm{E}^{\mathrm{D}} \Rightarrow \mathrm{E} \rightarrow \mathrm{E}^{\mathrm{D}}$ “>12”	$\mathrm{E}^{\mathrm{U}} \Rightarrow \mathrm{E}^{\mathrm{U}} \rightarrow \mathrm{E}$ “>12”
$\mathrm{E} \rightarrow$ “hours”	$\mathrm{E}^{\mathrm{D}} \Rightarrow \mathrm{E} \rightarrow$ “hours”	“hours”$^{\mathrm{U}} \Rightarrow \mathrm{E}^{\mathrm{U}} \rightarrow$ “hours”

Creation Rules	
$\Rightarrow \mathrm{E}^{\mathrm{D}}$	// starting from the root
$\Rightarrow \mathrm{E}^{\mathrm{DU}}$	// starting from a middle node
$\Rightarrow$ “hours”$^{\mathrm{U}}$	// starting from a leaf

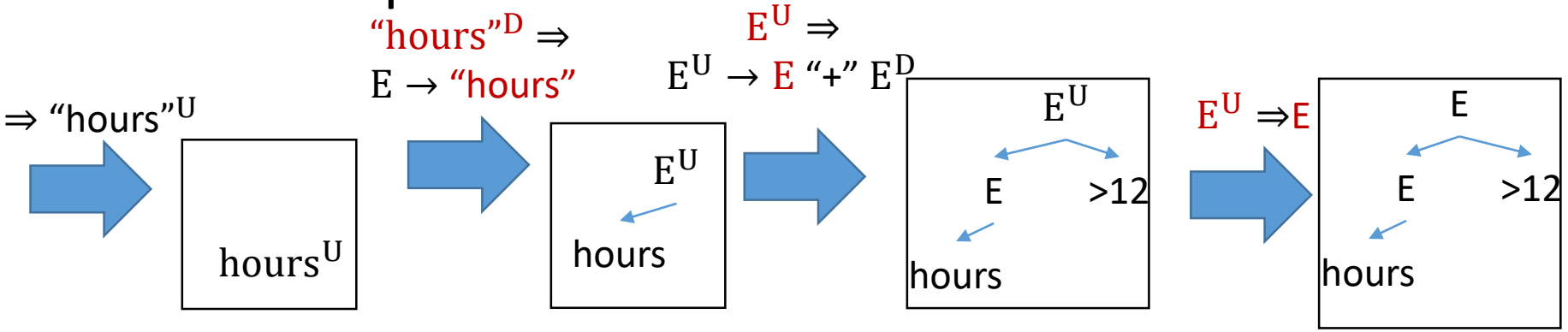
Ending Rule	$\mathrm{E}^{\mathrm{U}} \Rightarrow \mathrm{E}$

Example

- Top-down



- Bottom-up



Unambiguity

- A set of rewriting rules are unambiguous if
  - there is at most one unique set of rule applications to construct any program.

- When the rule set is unambiguous, we still have
  - $P(prog \mid context) = \prod_i P(rule_i \mid context, prog_i, position_i)$

Challenge 2: How to find the most probable program?

- Local Optimal ≠ Global Optimal

E_0

E → E " > 12"	0.3
E → E " > 0"	0.6

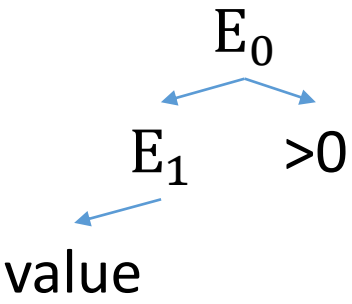
E_1

E → "hours"	0.1
E → "value"	0.2
E → E " + " E	0.05

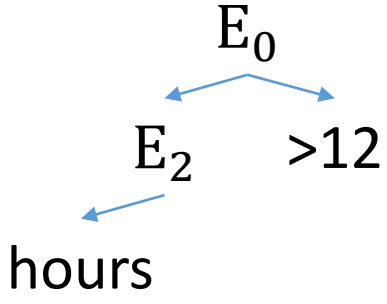
E_2

E → "hours"	0.8
E → "value"	0.1
E → E " + " E	0.05

0.6 * 0.2
= 0.12



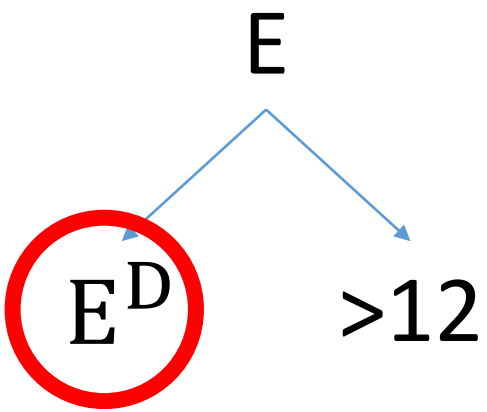
0.3 * 0.8
= 0.24



Idea 1: Use Metaheuristic Search

- Beam Search:
  - Keep n most probable partial programs
  - Expand the programs to get new programs
- Genetic Search:
  - Keep n most probably complete programs
  - Mutate the programs to get new programs

Idea 2: Pruning off Invalid Choices



$$E^D \Rightarrow E \rightarrow E^D \text{ “+” } E^D$$
$$\mid \quad \sout{E \rightarrow E^D \text{ “>12”}}$$
$$\mid \quad E \rightarrow \text{“hours”}$$

- Generating constraints from the partial AST
  - Type constraints
  - Size constraints
  - Semantic constraints from E
- Use a solver to determine invalid choices

Structural Constraint Generation

- Each AST node defines a set of variables
  - For type constraints, each node n defines a type variable type[n]
- Each grammar rule defines a set of constraints
  - $E_1 \rightarrow E_2$ “>12”
    - $T[E_1]$ = Boolean
    - $T[E_2]$ = Int
  - $E_2 \rightarrow E_3$ “>12”
    - $T[E_2]$ = Boolean
    - $T[E_3]$ = Int
- The context gives a set of constraints
  - $T[\text{hours}] = \text{Int}$
  - $T[E_{\text{root}}] = \text{Boolean}$
- If a solver returns unsat, drop the current choice

Applications

- Application 1:
  - Repairing Conditional Expressions
- Application 2:
  - Generating Code from Natural Language Expression

Repairing Conditional Expressions

- Condition bugs are common

```
  hours = convert(value);
+ if (hours > 12)
+   throw new ArithmeticException();
```

Missing boundary checks

```
-  if (hours >= 24)
+ if (hours > 24)
      withinOneDay=true;
```

Conditions too weak or too strong

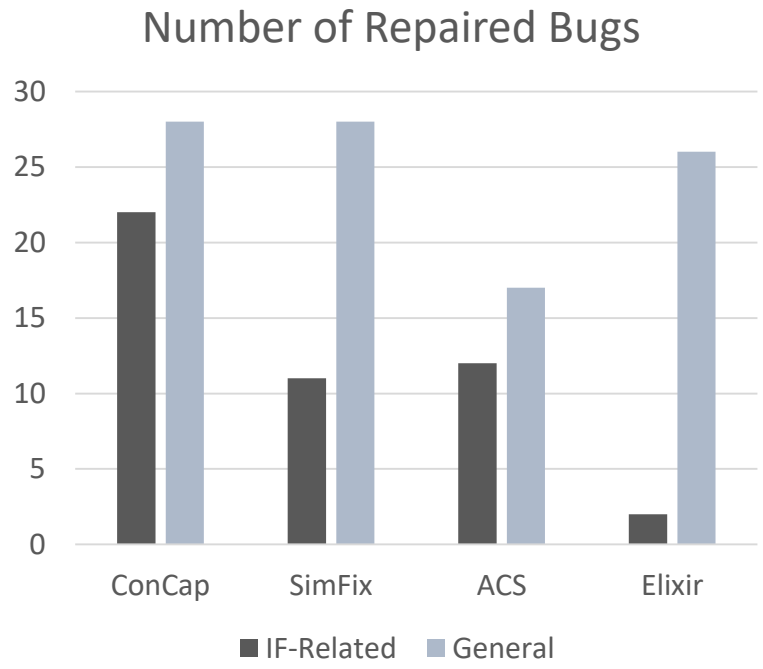
- Steps:
  1. Localize a buggy if condition with SBFL and predicate switching
  2. Synthesize an if condition to replace the buggy one
  3. Validate the new program with tests

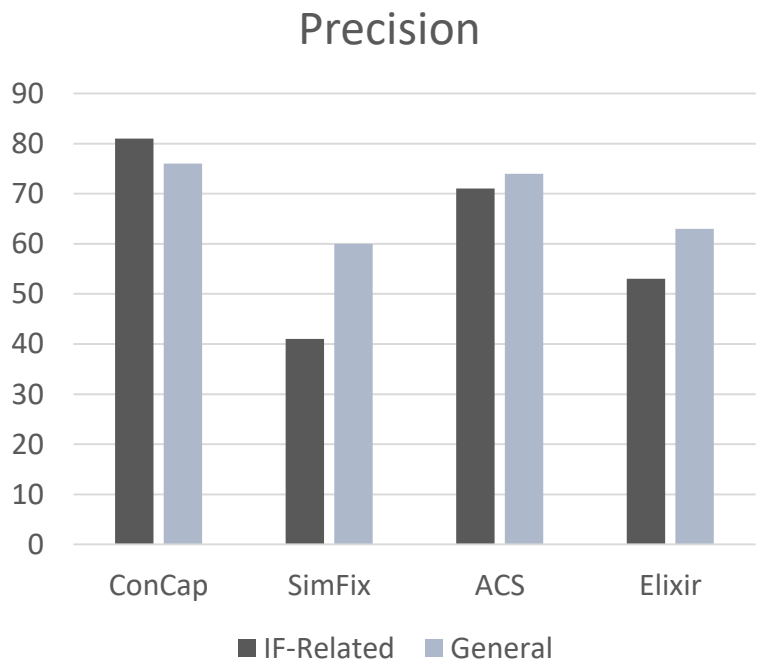
L2S Configuration

- Rewriting rules
  - Bottom-up
  - Estimate the leftmost variable first
- Machine learning
  - Xgboost
  - Manually designed features
- Constraints
  - Type constraints & size constraints
- Search algorithm
  - Beam search

Results

Benchmark: Defects4J





Also repaired 8 unique bugs that have never been repaired by any approach.

Generating Code from Natural Language Expression

- Can we generate code automatically to avoid repetitive coding?
- Existing approaches use RNN to translate natural language descriptions to programs
  - **Long dependency problem**: work poorly on long programs


```
[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"
```

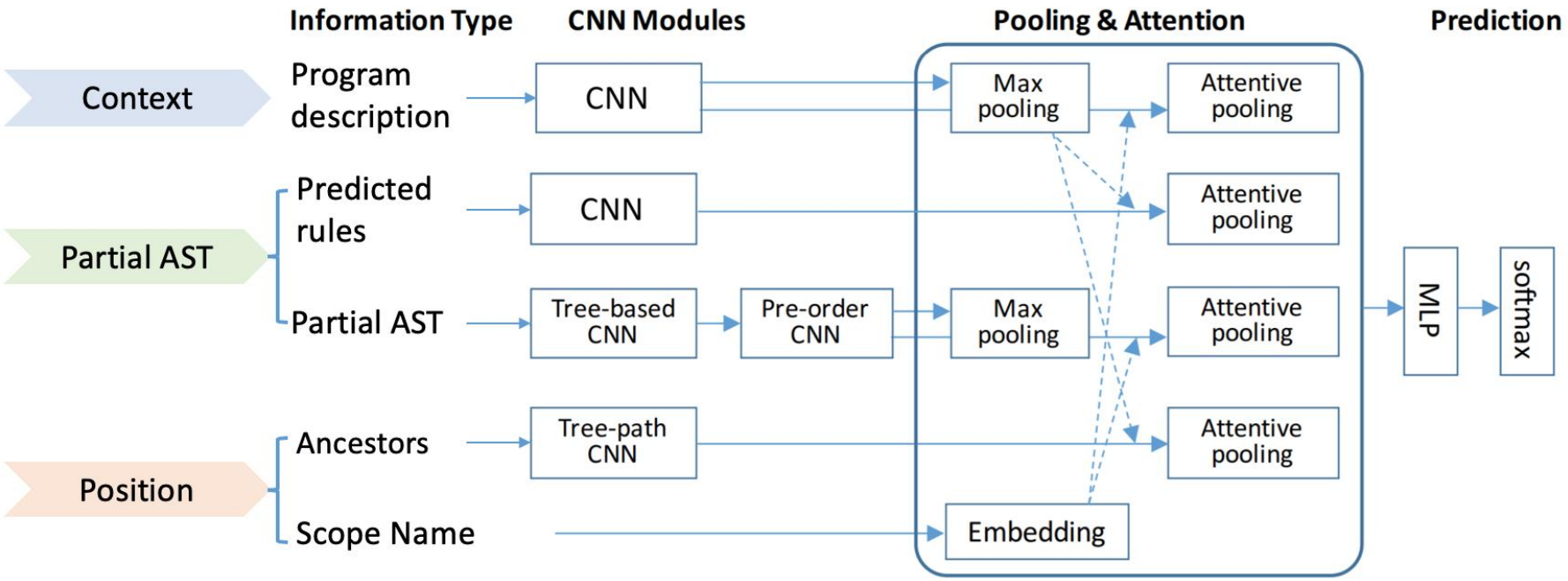
```
class AcidicSwampOoze(MinionCard):
    def __init__(self):
        super().__init__("Acidic Swamp Ooze", 2,
            CHARACTER_CLASS.ALL, CARD_RARITY.COMMON,
            battlecry=Battlecry(Destroy(), WeaponSelector(EnemyPlayer())))

    def create_minion(self, player):
        return Minion(3, 2)
```

L2S Configuration

- Rewriting rules
  - Top-down
- Machine learning
  - A CNN-based network
- Constraints
  - Size constraints
- Search algorithm
  - Beam search

A CNN-based Network Architecture



Results

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**

Newest Results

- Replacing CNN with Transformer
  - Transformer: a new neural architecture at 2017
  - The flexibility of L2S allows to easily utilize new models

	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

Aladdin FL 软件介绍

Aladdin FL 目的

目前的科研人员已经可以在特定的数据集上，将错误定位的准确率提高到58%[1]。

为了将这项研究成果产品化，我们和北大软件所的科研人员们合作，将错误定位的前沿技术与AI结合，***帮助广大的程序员更快的找到代码中的问题，提高开发效率***。

[1] DeepFL: Integrating Multiple Fault Diagnosis Dimensions for Deep Fault Localization

遇到的问题

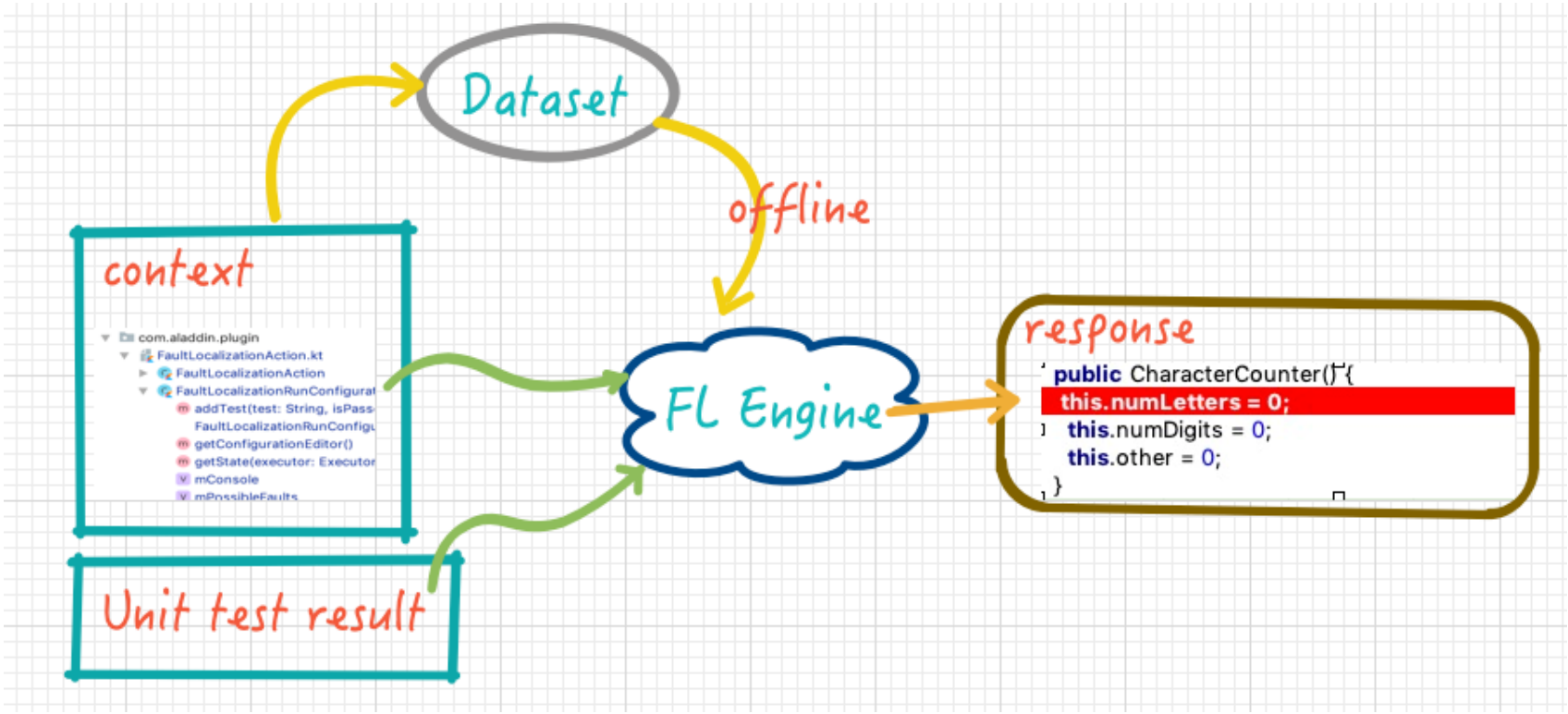
如果要把这个产品做好，一定要解决的几个问题：

- 友好的软件UI和环境，我们是为用户解决问题，而不是增加用户的学习成本
- 大量的数据集，帮助科学家们找到新的方法提高错误定位的效率。
- 稳定且易扩展架构，为用户提供优秀的服务
- 强大计算能力，运行更复杂的算法模型

解决方案

问题	解决方案
界面	我们提供主流IDE 的plugin，使用像执行单元测试一样简单。
数据集	和科学家们共同制定data protocol，将隐私数据在客户端处理后作为特征采集到数据仓库，兼顾用户隐私和训练模型所需的数据。
架构	客户端、服务端、算法端和数据闭环清晰的划分，保证了系统的多种需求。
计算能力	将算法的计算从用户端放到了服务端，突破单机计算的局限性，提高错误定位的效率。

架构设计

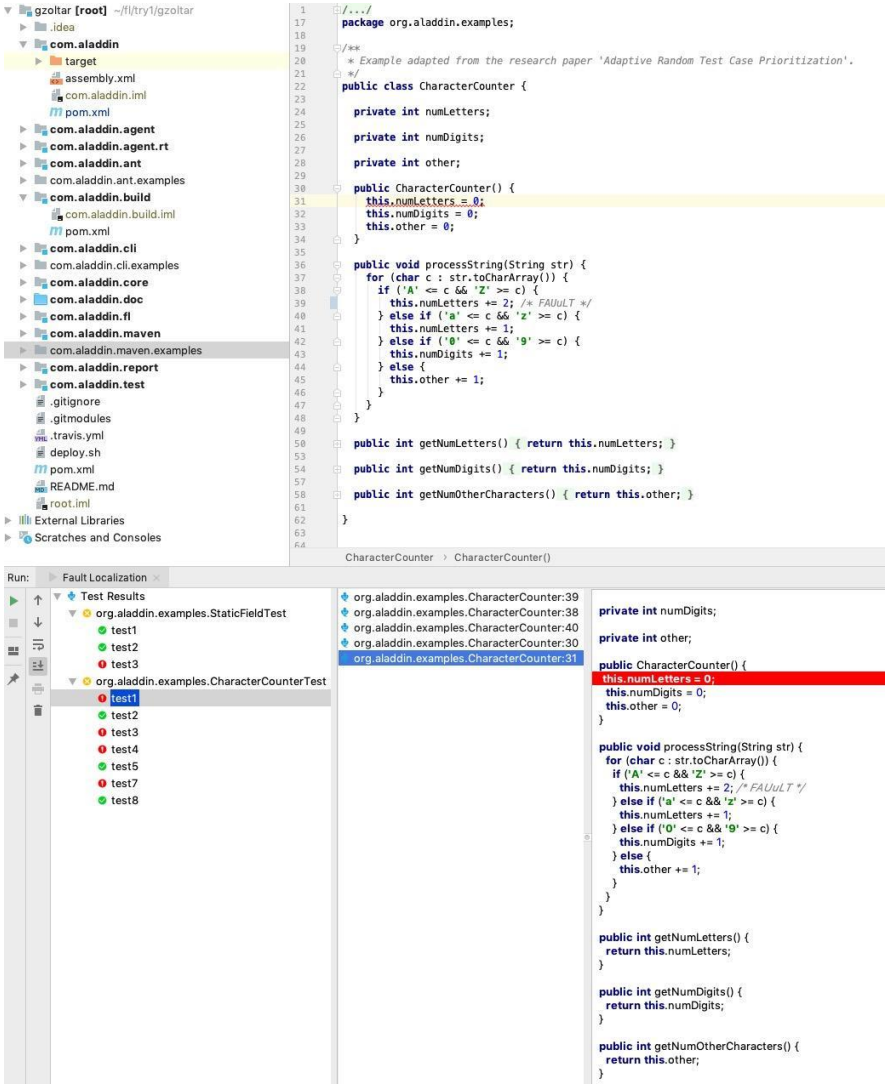

Dataset
offline
context
com.aladdin.plugin
FaultLocalizationAction.kt
FaultLocalizationAction
FaultLocalizationRunConfigurat
addTest(test: String, isPass
FaultLocalizationRunConfigu
getConfigurationEditor()
getState(executor: Executor
mConsole
mPossibleFaults
FL Engine
response
public CharacterCounter(){
this.numLetters = 0;
this.numDigits = 0;
this.other = 0;
}
Unit test result

现阶段成果

已经在实际的项目中做到工程师只需在本地执行一次单元测试，就可以迅速定位到导致失败的代码块。

目前已经支持IntelliJ IDEA / Maven / Gradle 插件。支持语言 Java。

右图所示的效果就是Aladdin在IntelliJ IDEA 使用效果。



欢迎广大同学加入我们团队，用Coding 革命 Coding

在这里你可以：

- 和一流的科研团队一起工作
- 学习到软工前沿的科学技术
- 有一个AI 落地的实践机会
- 锻炼你的工程能力

另外，我们在软工领域还有其它的产品规划：

- 代码自动修复
- 代码AI提示

联系我们

Email:

tai.wang@wuren.com

用Coding 革命Coding

让工程师的生活更美好！