软件分析

程序生成学习

熊英飞
北京大学

This Lecture

Classic Synthesis

- Problem Definition
- Enumerative
- Constraint-based
- Presentation-based

Program Estimation

- Problem Definition
- Estimating Probabilities
- Locating the most-likely one

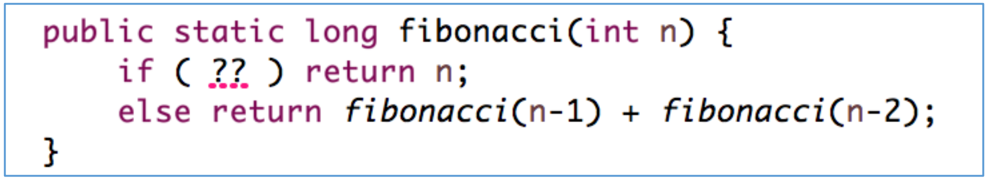
程序估计Program Estimation

- 输入:
  - 程序空间（用语法表示） G
  - 规约 S
  - 上下文 C
  - 包含上下文-程序对的训练集T
- 输出:
  - 程序P满足
    - $P \in G \wedge P \mapsto S \wedge \Pr(P \mid C)$
  - Pr 表示从T学习到的概率

- 如果Pr是满足规约的概率，那么可以用来加速传统程序分析

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 defined by a grammar

- Specification is a set of tests
- 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?

分解为三个问题

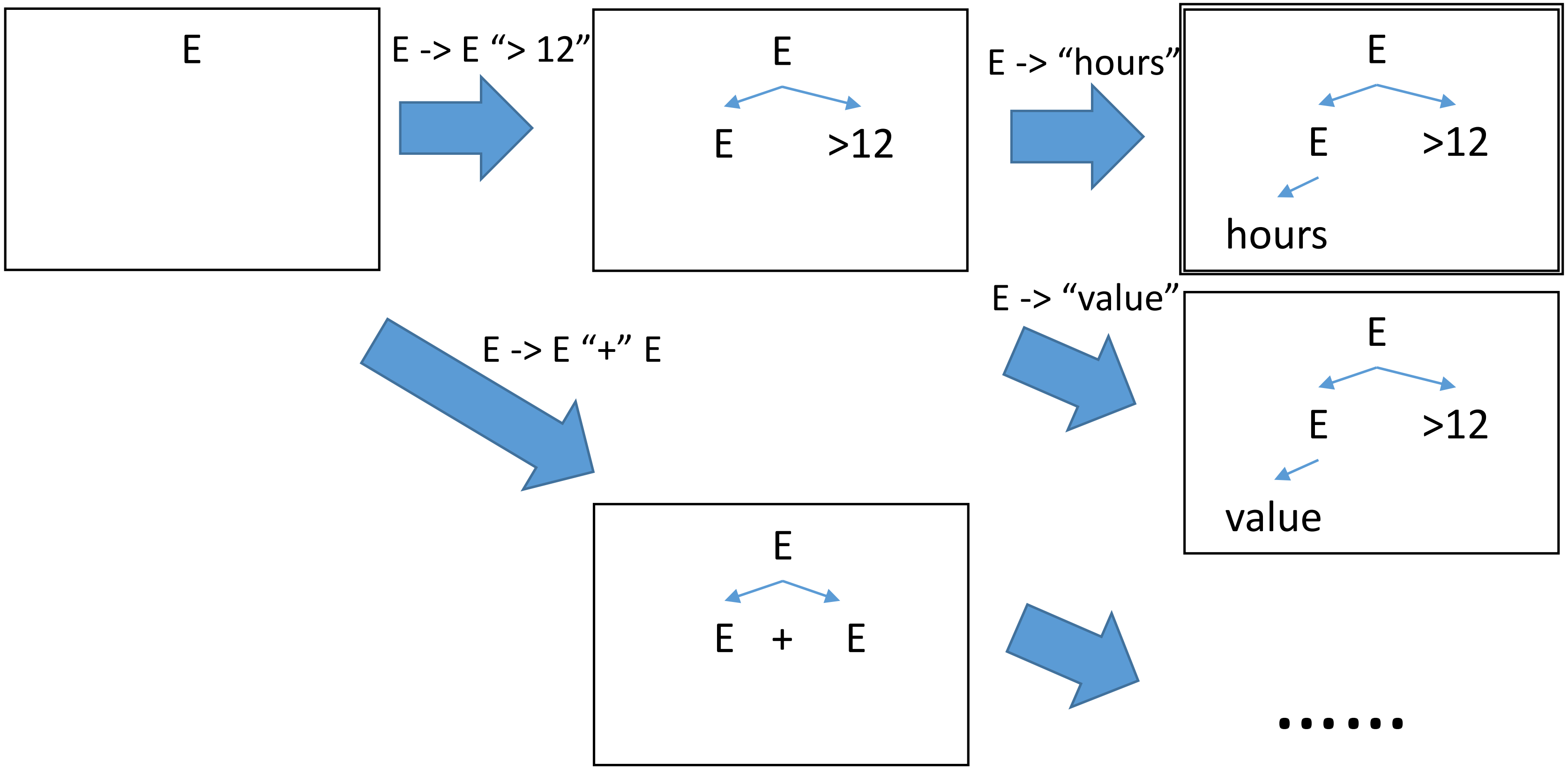
- 如何根据训练集计算一个程序的概率？
- 如何找到概率最大的程序？
- 如何保证找到的程序满足规约的要求？

程序生成学习框架
Learning to Synthesize (L2S)

- 本课题组提出的框架
- 泛化了之前的多种具体方法

- 将程序估计问题看做一个图的路径查找问题
  - 起始节点是空程序
  - 目标节点是满足规约的程序
  - 路径的权是终点程序的概率
  - 目标是找到一条从起始节点到任意目标节点的最优路径

程序估计问题作为路径查找问题


E

E -> E “> 12”

E

E >12

E -> “hours”

E

E >12

hours

E -> “value”

E

E >12

value

E -> E “+” E

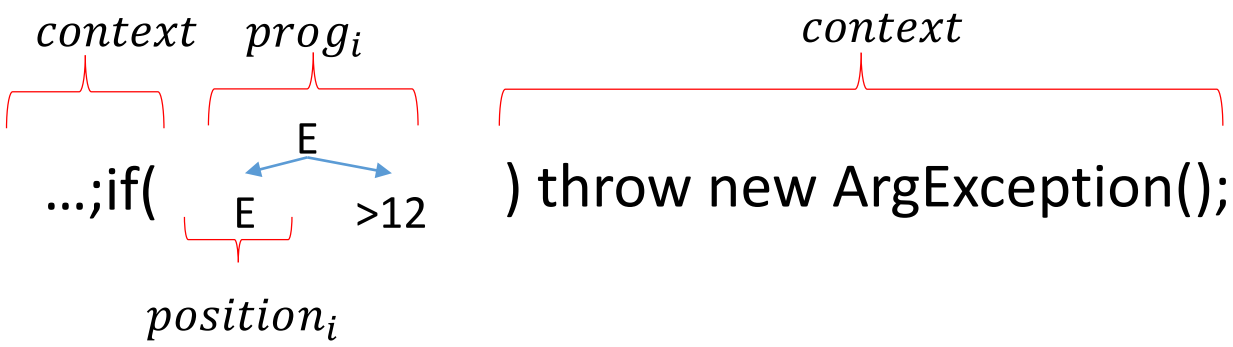
E

E + E

……

如何计算一个程序的概率？

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



$context$ $prog_i$ $context$

E

…;if(E >12) throw new ArgException();

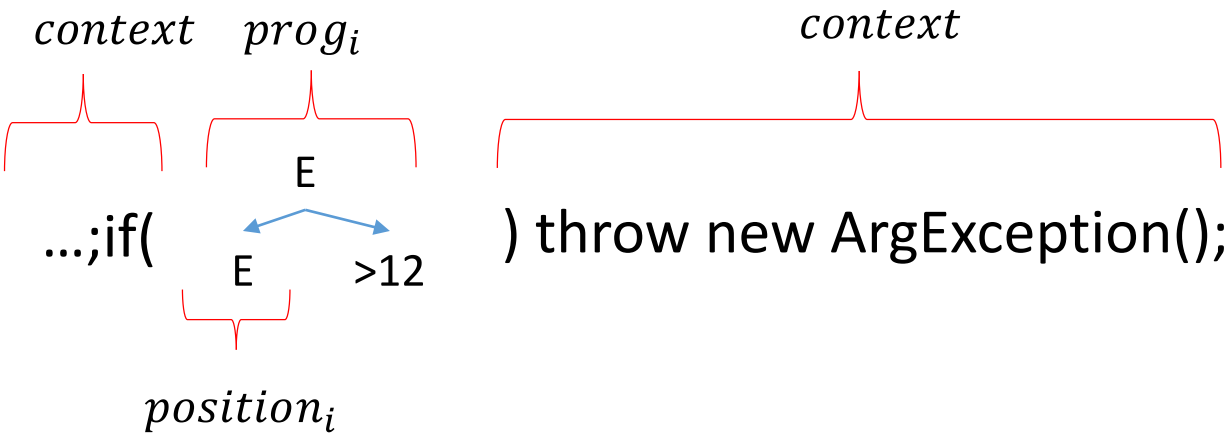
$position_i$

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

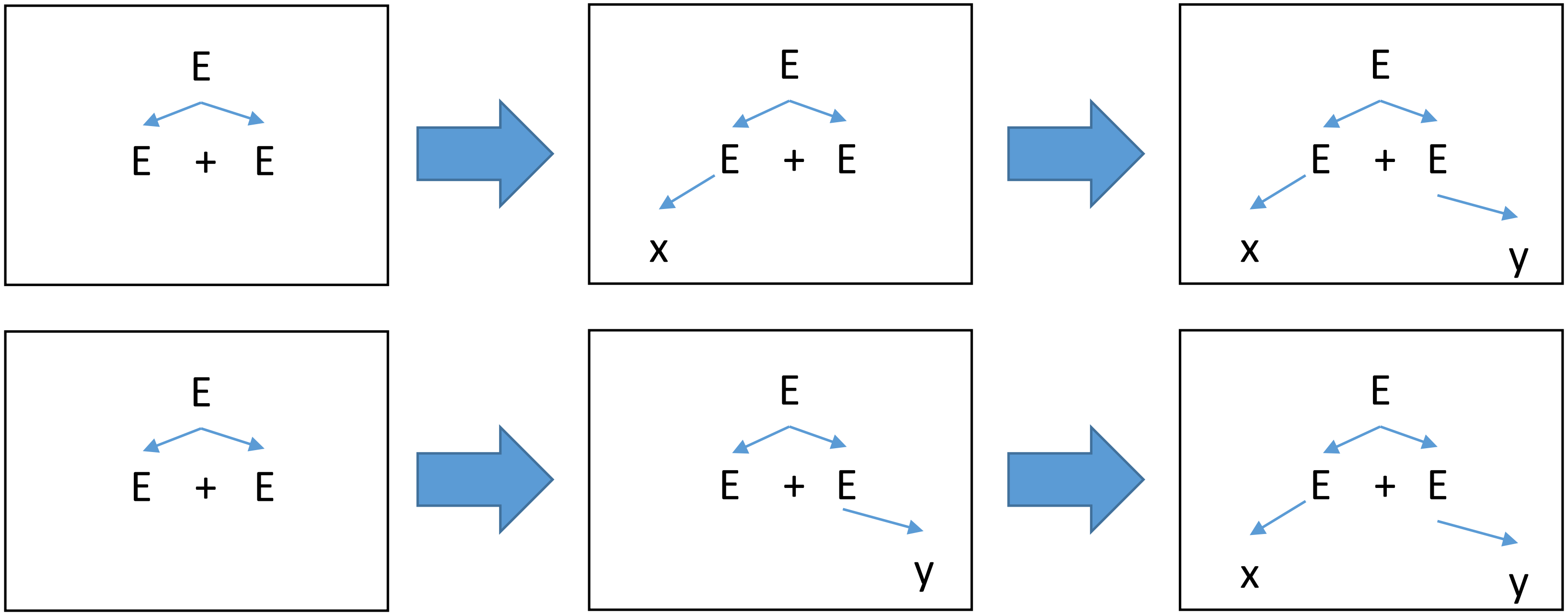


$context$ $prog_i$ $context$

…;if(E E >12) throw new ArgException();

$position_i$

The order of expansion



- Does different order make a difference?

The order of expansion

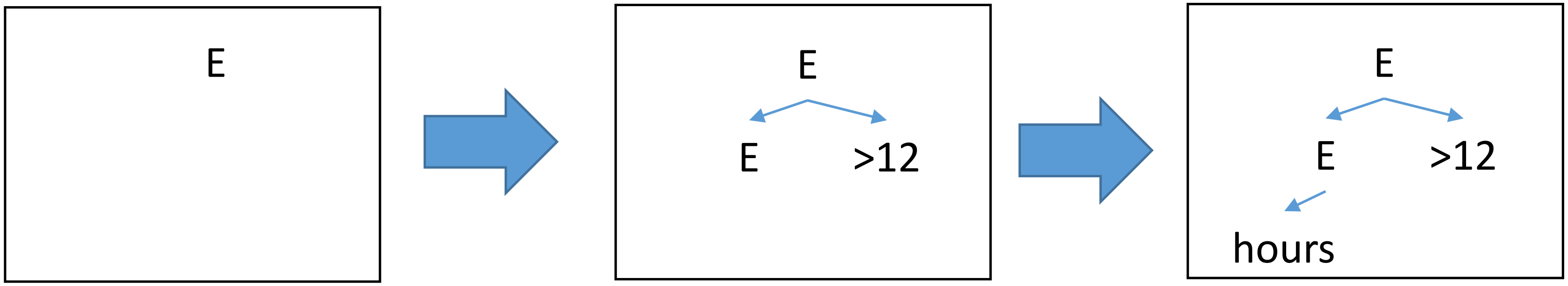
- 如果所有概率都是精确的，两者的结果没有差别
  - $P(prog \mid context) = \prod_i P(\ rule_i \mid context, prog_i, position_i\ )$
- 证明：假设存在一个policy，决定一个不完整程序中哪个节点先被展开，那么policy的选择和prog的概率是独立的
  - $Pr(prog)$
  - $= \ \mathrm{Pr}(prog \mid policy)$ //独立性
  - $= Pr((\langle\ prog_i, pos_i, rule_i\rangle)\ _{i=1}^{n} \mid policy)$
  - $= Pr(\ prog_1 \mid \ policy\ )\ Pr(\ pos_1 \mid \ policy, prog_1\ )$ $Pr(\ rule_1 \mid \ policy, prog_1, pos_1\ )$ $Pr(\ eprog_2 \mid \ policy, prog_1, pos_1, rule_1\ ) \ldots$ $Pr(\ eprog_{n+1} \mid \ policy, (eprog_i)_{i=1}^{n}, (pos_i)_{i=1}^{n}, (rule_i)_{i=1}^{n}\ )$
  - $= \prod_i Pr\left(rule_i \mid policy, \left(rule_j\right)_{j=1}^{i-1}, pos_i\right)$ //删除概率为1的项
  - $= \prod_i \mathrm{Pr}(rule_i \mid \ policy, prog_i, pos_i)$
  - $= \prod_i \mathrm{Pr}(rule_i \mid \ prog_i, pos_i)$ //独立性

The order of expansion

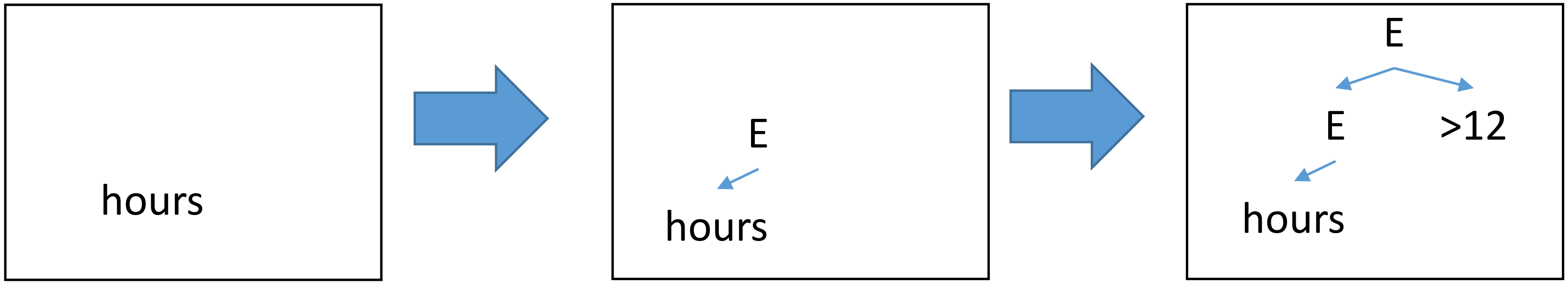
- 虽然精确的概率乘积是相同的，但
  - 统计模型/机器学习的预测精度可能不同
  - 对路径查找问题求解算法的影响不同
- 根据经验，采用不同的顺序可能对结果产生很大影响

Order beyond CFG?

- Top-down



- Bottom-up



CFG to Expansion Rules

$$\mathrm{T} \longrightarrow E$$

$$E \longrightarrow E \text{ “} > 12\text{”} \mid E \text{ “} > 0\text{”} \mid E \text{ “} + \text{” } E \mid \text{“}hours\text{”} \mid \text{“}value\text{”} \mid \ldots$$

$\langle E \longrightarrow \text{“}hours\text{”}, \quad \perp\rangle$

$\langle E \longrightarrow \text{“}value\text{”}, \quad \perp\rangle$

$\langle E \longrightarrow E \text{ “} > 12\text{”}, \quad 1\rangle$

$\langle E \longrightarrow E \text{ “} + \text{” } E, \quad 1\rangle$

$\langle T \longrightarrow E, \quad 1\rangle$

$\langle E \longrightarrow E \text{ “} > 12\text{”}, \quad 0\rangle$

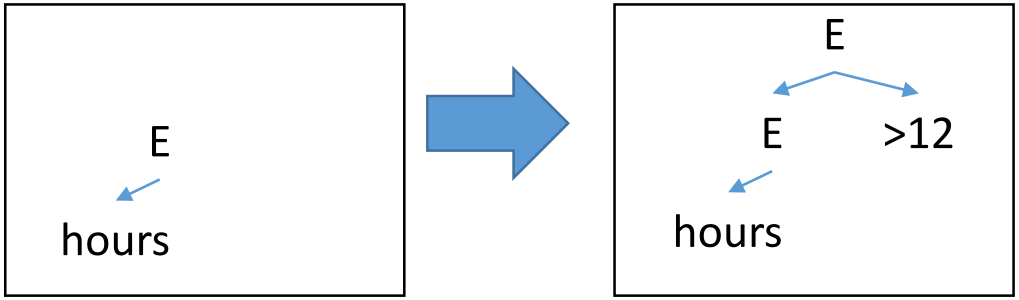
$\langle E \longrightarrow E \text{ “} + \text{” } E, \quad 0\rangle$

$\langle E \longrightarrow \text{“}hours\text{”}, \quad 0\rangle$

$\langle E \longrightarrow \text{“}value\text{”}, \quad 0\rangle$

自底向上规则：$\langle E \rightarrow E\text{“} > 12\text{”}, 1\rangle$

Generate the tree if the ith child is ready



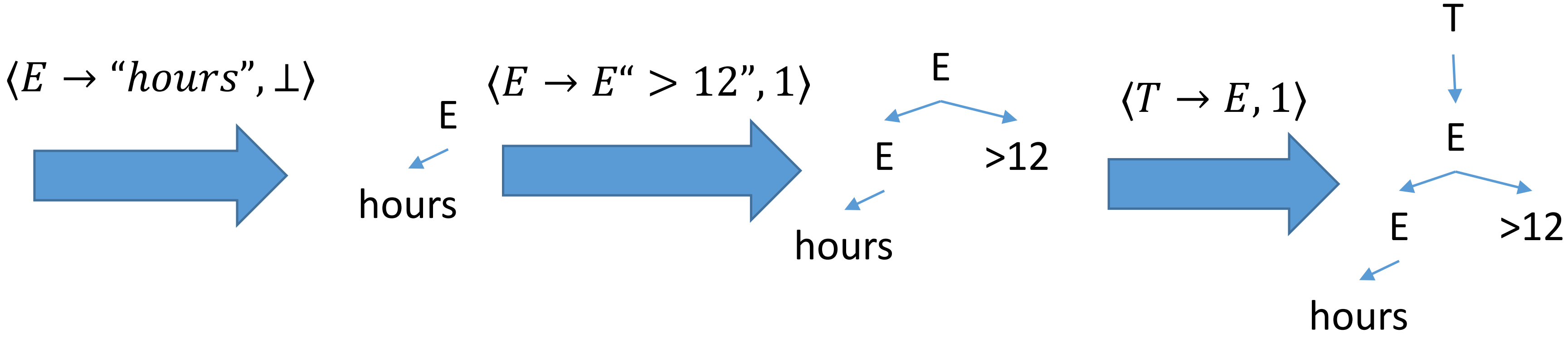
自顶向下规则：$\langle E \rightarrow E\text{“} > 12\text{”}, 0\rangle$

Generate the tree is the root is generated

创建规则： $\langle E \rightarrow \text{“}hours\text{”}, \perp\rangle$

Generate the tree at the beginning

Generation by Expansion Rules



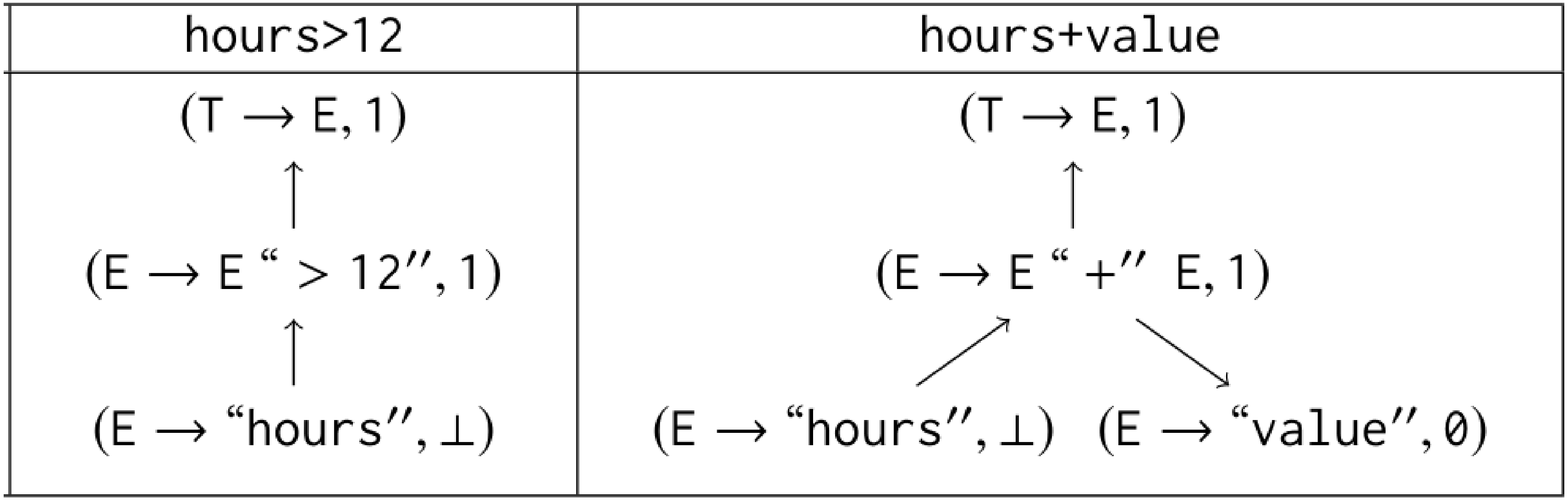
$\langle E \rightarrow \text{“}hours\text{”}, \perp \rangle$

$\langle E \rightarrow E\text{“} > 12\text{”}, 1 \rangle$

$\langle T \rightarrow E, 1 \rangle$

Expansion Trees

- Expansion trees capture how expansion rules are applied



hours>12	hours+value
$(T \longrightarrow E, 1)$ ↑ $(E \longrightarrow E \text{ “} > 12\text{”}, 1)$ ↑ $(E \longrightarrow \text{“hours”}, \bot)$	$(T \longrightarrow E, 1)$ ↑ $(E \longrightarrow E \text{ “}+\text{” } E, 1)$ ↗ ↘ $(E \longrightarrow \text{“hours”}, \bot)$ $(E \longrightarrow \text{“value”}, 0)$

AST -> Expansion Tree

- **完整性Completeness**: for each AST, there is at least one expansion tree
- **唯一性Uniqueness**: for each AST, there is at most one expansion tree

- 是否总是存在完整和唯一的Expansion Rule集合?

唯一和完整集合的充分条件

$\mathrm{T} \rightarrow E$

$E \rightarrow E \text{ “} > 12\text{”} \mid E \text{ “} > 0\text{”} \mid E \text{ “} + \text{” } E \mid \text{“}hours\text{”} \mid \text{“}value\text{”} \mid \ldots$

$\langle E \rightarrow \text{“}hours\text{”}, \quad \perp\rangle$
$\langle E \rightarrow \text{“}value\text{”}, \quad \perp\rangle$
$\langle E \rightarrow E \text{ “} > 12\text{”}, \quad 1\rangle$
$\langle E \rightarrow E \text{ “} + \text{” } E, \quad 1\rangle$
$\langle T \rightarrow E, \quad 1\rangle$
$\langle E \rightarrow E \text{ “} > 12\text{”}, \quad 0\rangle$
$\langle E \rightarrow E \text{ “} + \text{” } E, \quad 0\rangle$
$\langle E \rightarrow \text{“}hours\text{”}, \quad 0\rangle$
$\langle E \rightarrow \text{“}value\text{”}, \quad 0\rangle$

1. 除了初始符号开头的规则，所有语法规则都有对应的自顶向下展开规则
2. 所有语法规则最多只有一条自底向上的展开规则
3. 对于所有从初始符号（延自底向上展开规则）反向可达的非终结符，其所有语法规则都有一条自底向上展开规则或创建规则

从初始符号开始选择创建/自底向上规则即可

AST -> Expansion Tree

- 利用一个动态规划算法，AST可以在O(n)时间内转成Expansion Tree
  - 后根次序依次判断每个AST结点是否可以被自底向上和自顶向下的方式生成，如果可以，记录下采用的规则
  - 先根次序恢复出Expansion Tree

如何找到概率最大的程序？

- 采用求解路径查找问题的标准算法
- 精确算法
  - 迪杰斯特拉算法
  - A*算法
- 近似算法
  - Beam Search

如何保证找到的程序满足规约的要求？

- 基本方法：
  - 寻找概率最大的程序
  - 判断是否满足规约
  - 如不满足，回到第一步

- 能否在搜索过程中就进行剪枝？

剪枝

- 搜索过程中剪枝
  - 语义：假设输入变量的取值仅为2，要求输出为3，且语法中只有加号，那么E+E肯定无法满足
  - 类型：E+E && E肯定无法满足
  - 大小：假设AST树的大小（节点数）限定为4，那么E+E肯定无法满足

- 剪枝的条件
  - 所有可展开的程序都无法满足约束

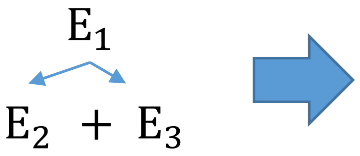
语法上的静态分析

- 假设所有约束都是Pred(Prop(N))的形式
  - N：非终结符
  - Prop：以N为根节点的子树所具有的属性值
  - Pred：该属性值所应该满足的谓词
- 如：
  - 语义约束：Prop为表达式取值
  - 类型约束：Prop为表达式的可能类型
  - 大小约束：Prop为表达式的大小
- 通过静态分析获得Prop的所有可能取值
  - 要求上近似
- 如果所有可能取值都不能满足Pred，则该部分程序可以减掉

语法上静态分析示例：语义

- 抽象域：由1, 2, 3, 4, 5, >5, <1, true, false构成的集合
- 容易定义出抽象域上的计算

- 从语法规则产生方程
- E->E+E | “x” | E “>5” |…
  - V[E]=(V[E]+V[E]) ∪ {2} ∪(V[E]>5) ∪ …
- 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



$V[E_1] = V[E] + V[E]$

语法上静态分析示例：类型和大小

- 抽象域：由Int, Float, Boolean构成的集合

- 从语法规则产生方程
- E->E+E | “hours” | E “>5” |…
  - T[E]=(T[E]+T[E]) ∪ {Int} ∪(V[E]>5) ∪ …
- 其中
  - $t_1 + t_2 = \begin{cases} \{Int, Float\}, \text{同时满足下面两个条件} \\ \{Int\},\ \ Int \in t_1 \wedge Int \in t_2 \\ \{Float\},\ \ Float \in t_1 \wedge Float \in t_2 \\ \emptyset, \text{否则} \end{cases}$
  - $t > 5 = \begin{cases} \{Boolean\}, Int \in t \vee Float \in t \\ \emptyset, \text{否则} \end{cases}$
- 用类似方法可以计算非终结符展开的最小大小

从AST到Expansion Tree

- 类似的规则可以对不完整Expansion Tree定义
- 需要计算出每个非终结符向上/向下展开的所有可能取值

Summary

- L2S Combines four tools
  - **Expansion rules**: a novel extension to CFG for defining a path finding problem
  - **Static Analysis on Rules**: pruning off invalid choices in each step
  - **Statistical models**: estimating the probabilities of choices in each step
  - **Search algorithms**: solving the path finding problem

Evaluation

- Evaluation 1:
  - Repairing Conditional Expressions
- Evaluation 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

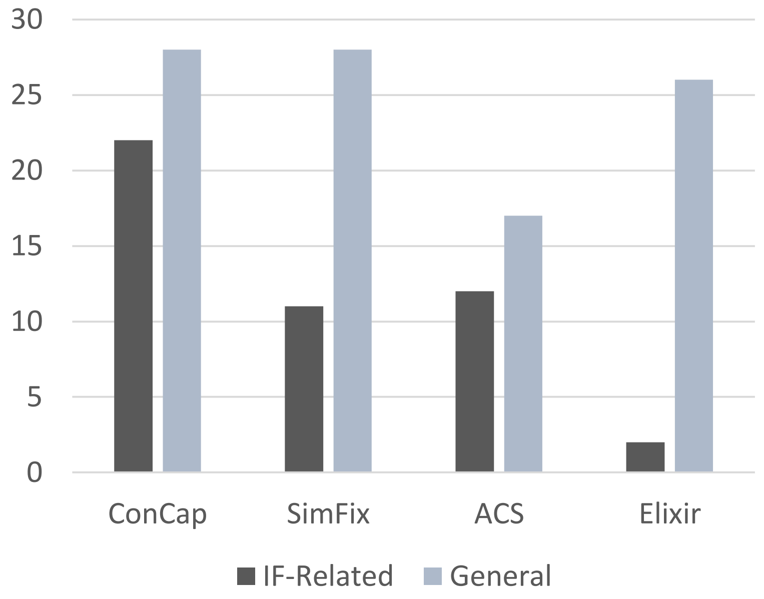
- Steps:
  1. Localize a buggy if condition with SBFL and predicate switching
  2. Synthesize an if condition to replace the buggy one
  3. Validate the new program with tests

L2S Configuration

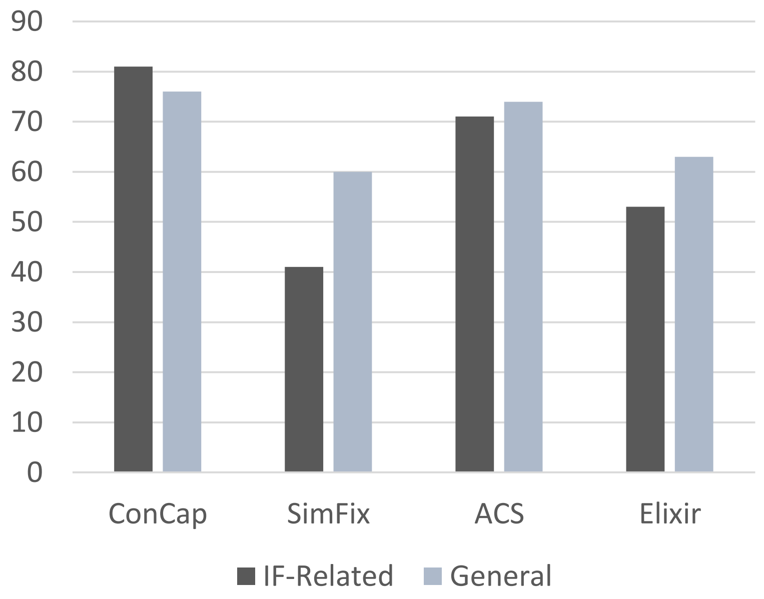
- Expansion rules
  - Bottom-up
  - Estimate the leftmost variable first
- Machine learning
  - Xgboost
  - Manually designed features
- Constraints
  - Type constraints & size constraints
- Search algorithm
  - Beam search

Results

Benchmark: Defects4J



Number of Repaired Bugs

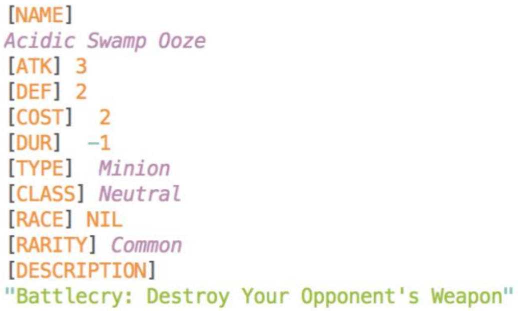


Precision

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"
```

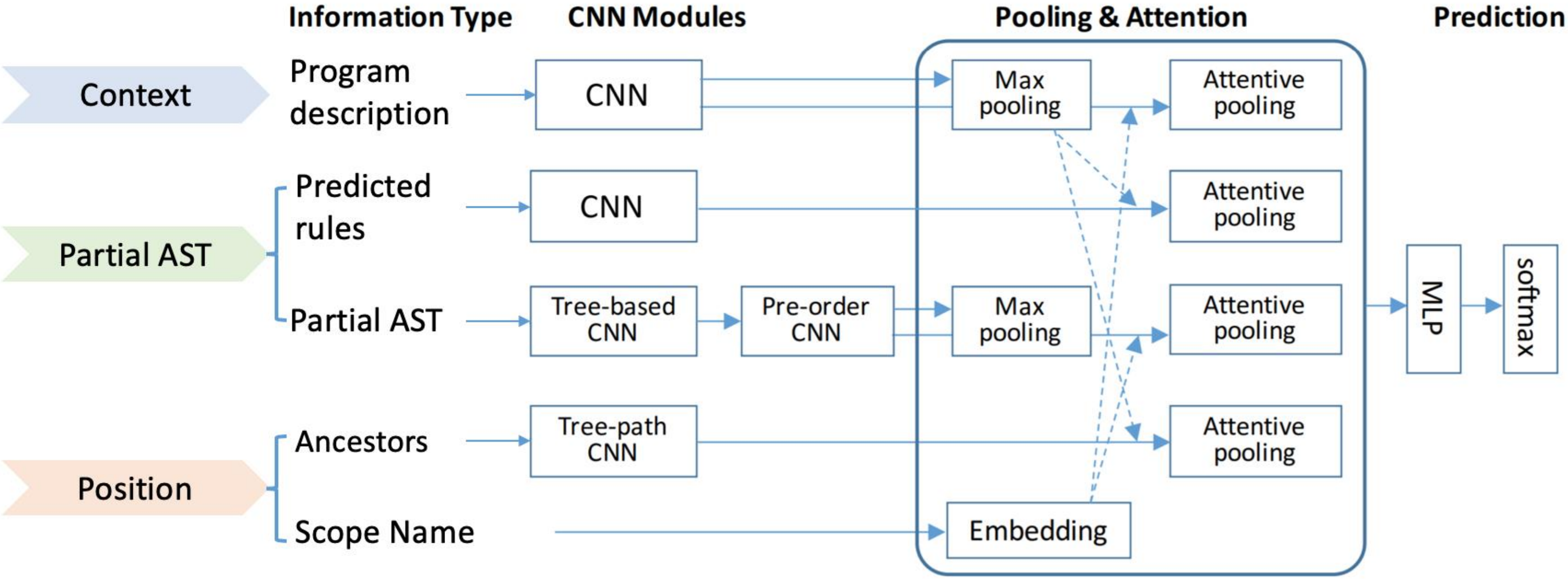
```
class AcidicSwampOoze(MinionCard):
    def __init__(self):
        super().__init__("Acidic Swamp Ooze", 2,
            CHARACTER_CLASS.ALL, CARD_RARITY.COMMON,
            battlecry=Battlecry(Destroy(), WeaponSelector(EnemyPlayer())))

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

L2S Configuration

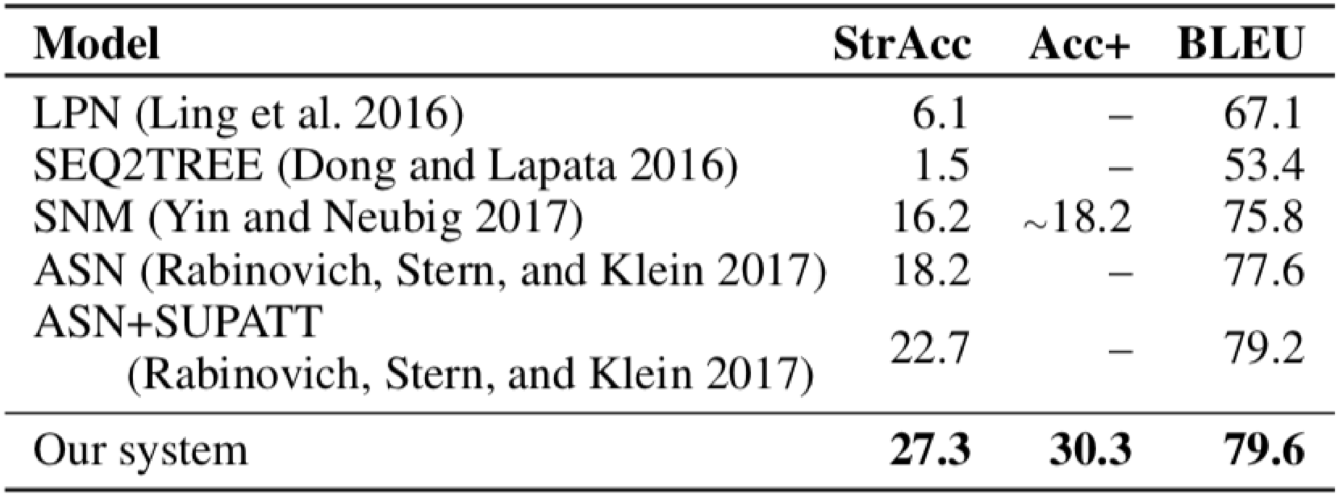
- Expansion rules
  - Top-down
- Machine learning
  - A CNN-based network
- Constraints
  - Size constraints
- Search algorithm
  - Beam search

A CNN-based Network Architecture



Results

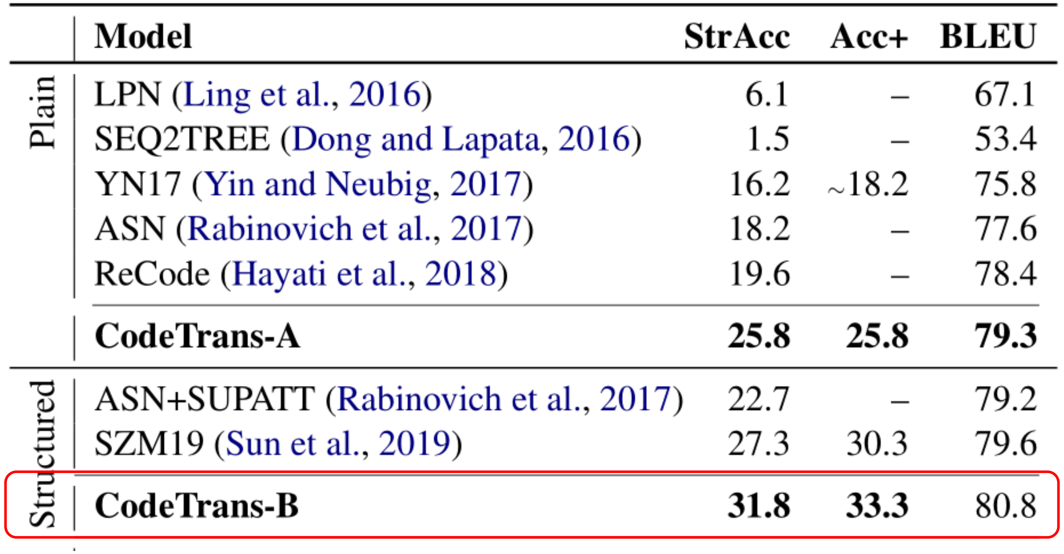
Benchmark: HearthStone



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 the CNN with a 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

Conclusion

- Program Estimation: to find the most probable program under a context
- L2S: combining four tools to solve program estimation
- Why worked?
  - Machine learning to estimate probability
  - Expansion rules and constraints to confine the space
  - Search algorithms to locate the best program
- Better to combine the tools we have

参考资料

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