

Grading Uncompilable Programs

Rohit Takhar and Varun Aggarwal

Aspiring Minds

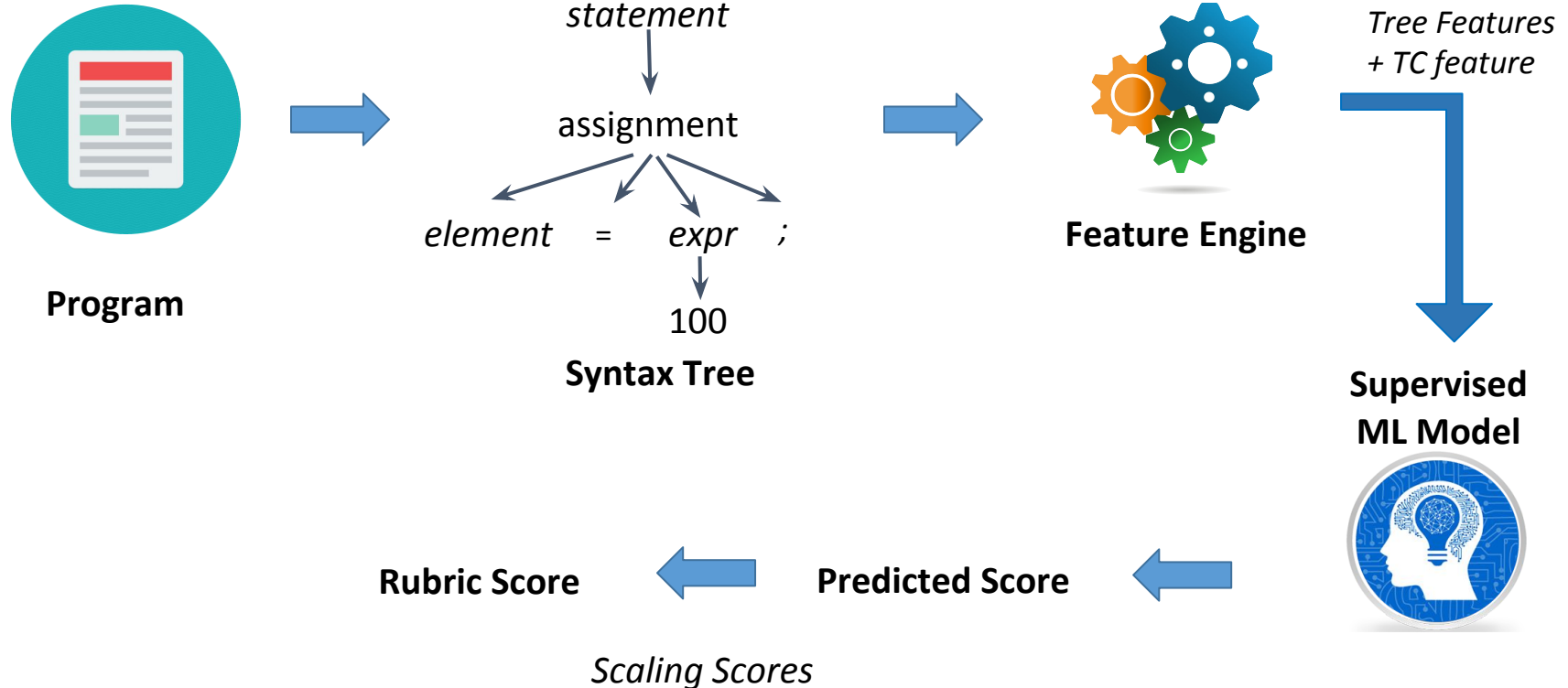
30 January 2019 | IAAI 2019

Motivation: Assessing programs with syntax errors?

- **Rejecting students with uncompiling codes?**
 - **Our study:** Around 16% of uncompileable programs had nearly correct logic
 - Logical ability Vs Language syntax knowledge
- **Traditional test suite based approach**
 - Logical ability - High False Negative rate

First approach to grade uncompileable programs at such a large scale! More than 200k tests till now

Flow diagram for compilable programs



Grammar for expressing features

- **Keywords and Tokens(Counts):**
 - Tokens like for, if, return, break; function calls like print, strlen, strcat; declarations like int, char
 - Operators like various arithmetic, logical, relational operators used
 - Character constants like '\0', ' ', '65', '96'
- **Capturing logical constructs (Interactions)**
 - Control flow structure
 - Data-dependencies
 - Data-dependencies in context of control-flow

TARGET PROGRAM

```
void print(int N){  
  for(i = 1 ; i <= N; i++){  
    print newline;  
    count = i;  
    for(j=0; j < i; j++){  
      print count; count++;  
    }  
  }  
}
```

Counts of control-related keywords/tokens

E.g.

count(**for**) = 2

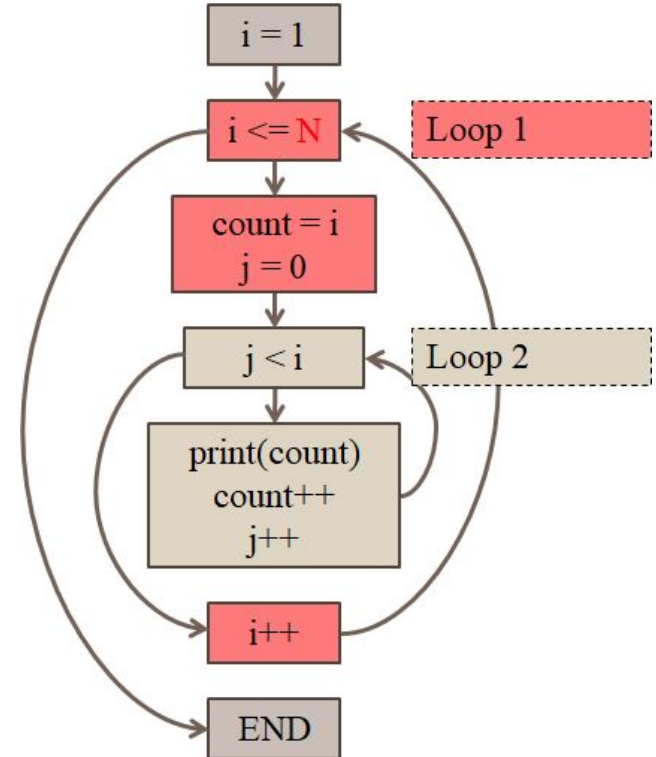
count(**for-in-for**) = 1

count(**while**) = 0

Capture control-context of data-dependencies in groups of expressions

$j < i$ dependant on variable used in $i++$

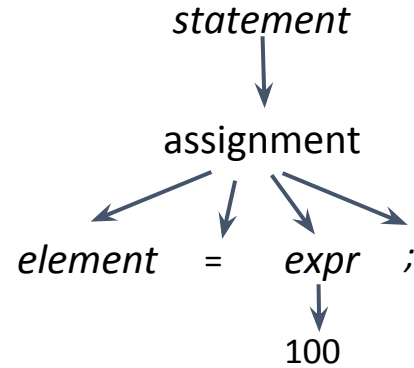
CONTROL FLOW GRAPH



For programs with syntax errors



Code



Syntax Tree



Our Approach

- Feature extraction techniques
 - Make compilable approach
 - Rule based technique
 - Data driven approach
 - Relaxing Rule approach
- Machine learning approach
 - Training supervised models using uncompileable codes
 - Reusing models trained on compilable codes


Make compilable method


- **Rule Based Approach**

- Manually analyzed errors and reasons for the errors.
- Defined rules to fix each type of error, wherever possible.
- For e.g. unbalanced parentheses, missing “;”, missing return statement etc.

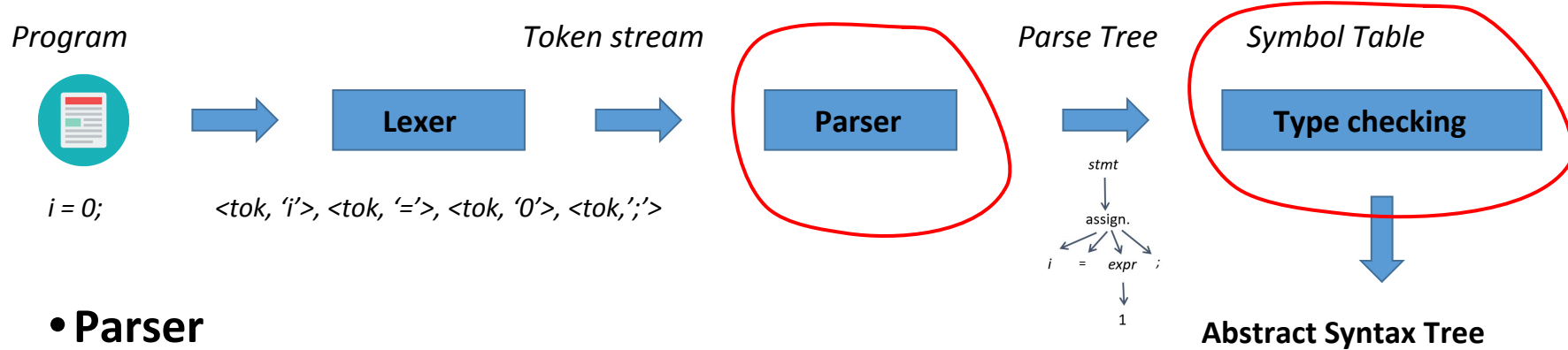
- **Data Driven Approach**

- Picking up patterns from the corpus and fixing the selected part of program.
- Limiting number of fixes - Skipping insertion of logical units like loops, conditions etc.

~48% 

52% 

Relaxing Rules Approach



• Parser

- Skipping the tokens which are violating grammar rules

`int [] arr = new int[];`

Expected **CONSTANT** value

• Type checking

- Relaxing rules in type checking step. For e.g type mismatch, undeclared variable errors, undefined function calls are skipped.

No
feature
loss

```
Stack<char> s = new Stack<char>();  
for(int i = 0 ; i < str.length(); i++){  
    char st = str.charAt(i);  
    if(st == '(' || st == '{' || st == '[')  
        s.push(str.charAt(i));  
    if(s.empty()){  
        s.push(str.charAt(i));  
        continue;  
    } else{  
        char cur = s.top();  
        if(str == ')' && cur == '(') {  
            s.pop();  
        } else if(st == '}' && cur == '{') {  
            s.pop();  
        } else if(st == ']' && cur == '[') {  
            s.pop();  
        }  
    }  
}
```

...

peek() should be used instead of ***top()*** !

```
char strChar[]=st.toCharArray();
String rev = "";
for(int i=0;i<st.length();i++){
    if(st.charAt(i)=='a' || st.charAt(i)=='A'){
        rev=rev+"";
    }else if(st.charAt(i)=='e' || st.charAt(i)=='E'){
        rev=rev+"";
    }else if(st.charAt(i)=='i' || st.charAt(i)=='I'){
        rev=rev+"";
    }else if(st.charAt(i)=='o' || st.charAt(i)=='O'){
        rev=rev+"";
    }else if(st.charAt(i)=='u' || st.charAt(i)=='U'){
        rev=rev+"";
    }else{
        rev = rev + st.charAt(i);
    } return rev;
}
```

Type Mismatch Error

Experiments and Results

- Rubric and Dataset
- Make compilable method Vs rule relaxation method – Which one is better?
- Efficacy of models trained on uncompileable codes
- Reusing models trained on compilable codes

Rubric

Grade	Rubric Definition
1	Code unrelated to given task
2	Appropriate keywords and tokens are present
3	Right control structure exists with missing data dependency
4	Correct with inadvertent errors
5	Completely correct

Dataset

	Task Name	Ques. Id	#UC	#CC
Unseen set	countCacheMiss	24	370	106
	balancedParentheses	132	355	70
Seen set	grayCheck	43	367	175
	transposeMultMatrix	48	407	182
	eliminateVowelString	62	392	182

Error Metric: Pearson correlation coefficient (r) and mean absolute error

Results: RR vs MC method

		MC		RR (MC set)	
Ques. Id	Sample Size (MC)	<i>r</i>	<i>MAE</i>	<i>r</i>	<i>MAE</i>
24	171	0.78	0.44	0.71	0.51
132	165	0.63	0.45	0.65	0.44
43	175	0.86	0.25	0.78	0.49
48	220	0.65	0.54	0.72	0.43
62	198	0.74	0.55	0.58	0.68
Mean		0.73	0.45	0.69	0.51
Median		0.74	0.45	0.71	0.49

MC approach outperforms RR approach!

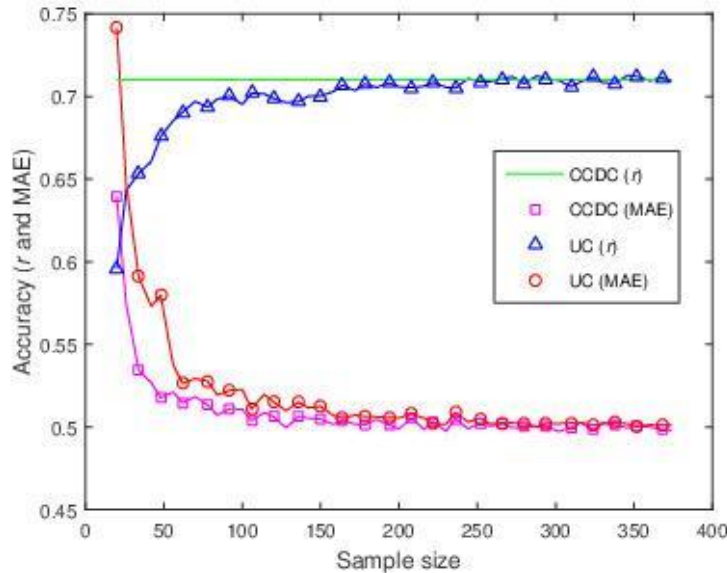
Results: Reusing models trained on compilable codes

	CC Model		CC Model with dist. Fixed		CC Model w/o TC and dist. Fixed		Uncompilable	
Ques. id	<i>r</i>	<i>MAE</i>	<i>r</i>	<i>MAE</i>	<i>r</i>	<i>MAE</i>	<i>r</i>	<i>MAE</i>
24	0.74	0.75	0.74	0.47	0.74	0.44	0.75	0.48
132	0.75	0.84	0.75	0.41	0.74	0.41	0.68	0.45
43	0.72	0.68	0.72	0.47	0.79	0.46	0.79	0.43
48	0.60	0.77	0.60	0.65	0.64	0.60	0.62	0.56
62	0.56	1.00	0.56	0.58	0.64	0.52	0.69	0.57
Mean	0.67	0.81	0.67	0.52	0.71	0.49	0.71	0.50
Median	0.72	0.77	0.72	0.47	0.74	0.46	0.69	0.48

Reusing models trained on compilable codes

Training set	Features for model training	Distribution Norms	Mean		Median	
			r	MAE	r	MAE
Uncompilable Codes	Tree features	Uncompilable Codes	0.71	0.50	0.69	0.48
Compilable Codes	Tree features + TC	Compilable Codes	0.67	0.81	0.72	0.77
Compilable Codes	Tree features	Compilable Codes	0.67	0.52	0.72	0.47
Compilable Codes	Tree features	Uncompilable Codes	0.71	0.49	0.74	0.46

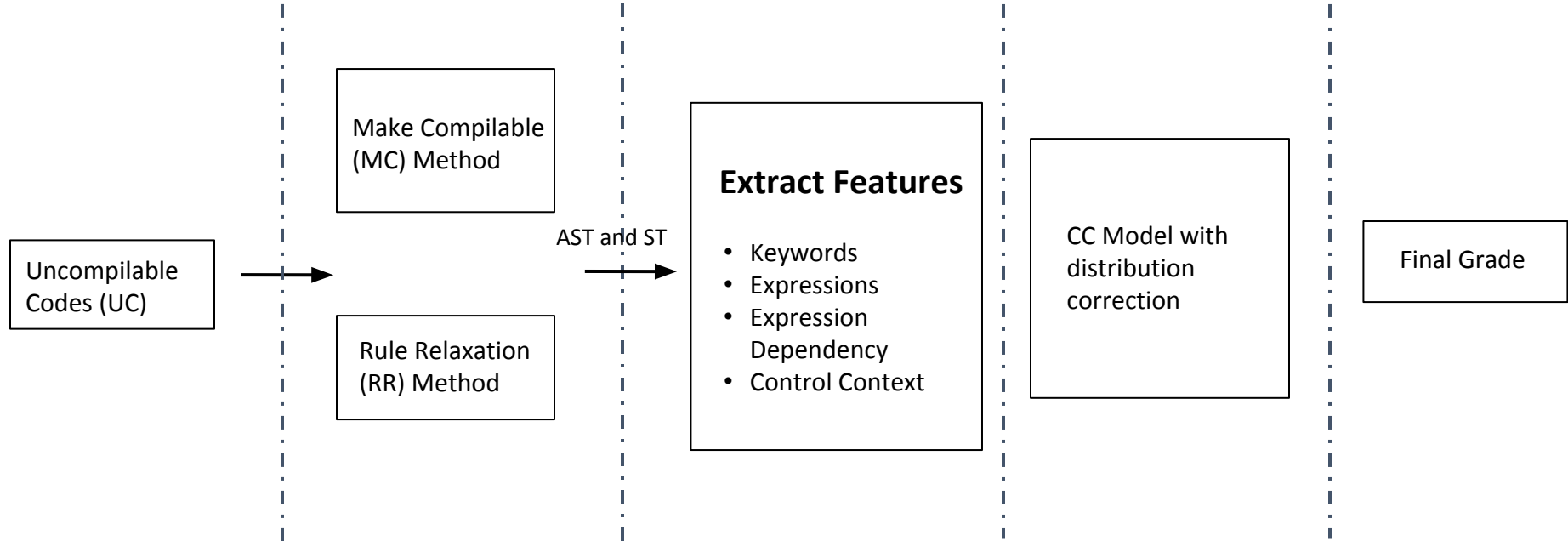
Distribution Correction – How much data do we need?



- **Scalability:** Models trained on compilable codes can be used for grading uncompileable codes

Need around 200 uncompileable codes graded for distribution correction!

System Flow



Conclusion

- A system to grade uncompileable programs
- An innovative feature extraction approach is proposed for uncompileable programs.
- We propose and demonstrate machine learning techniques to lower the need of human-graded data to build models.

Thank You!

Case studies

- Large multinational company based out of China
 - Total 29600 students evaluated on our programming platform
 - 26% more candidates shortlisted for the interview
 - 19% more candidates were hired
- IT services company in India
 - Improved the throughput by 30-40%.

Distribution of Programming Ability Scores

	Rubric Definition	Compilable	Uncompilable
1	Code unrelated to given task	3361	1979
2	Appropriate keywords and tokens are present	3264	2125
3	Right control structure exists with missing data dependency	2547	1440
4	Correct with inadvertent errors	2955	1017
5	Completely correct	3828	-
>=3	Selected for Interview	9330	2457
	Hired	2986	565