The Raychev-Vechev-Yahav Model

Given a program with some designated areas removed, how well can a natural language model predict the removed code?

- First explored in their paper “Code Completion with Statistical Language Models.”
- Each variable *v*, along with an execution of the program, yields a sequence of events of form (*func, S*), where *func* is invoked in the execution, and *S* is the set of places where *v* appears as a parameter of *func.*
- We can thus turn any program into a set of sentences with words corresponding to events.

Example

```
frame = screen.getFrame();
frame.resize(screen);
…
val = frame.getContents();
this.setHeader(0, val);
…
printf(“%s”, frame.name);
…
int x = val.toInt();
frame.setID(x,x);
…
x = x * 9;
Uint y = factorial(x);
Exit(0);
```

becomes

frame
(getFrame, (-1))
(resize, (0))
(getContents, (0))
(setID, (0))

val
(getContents, (-1))
(setHeader, (2))
(toInt, (0))

screen
(getFrame, (0))
(resize, (1))

x
(toInt, (-1))
(setID, (1, 2))
(factorial, (1))

y
(factorial, (-1))

Limitations

- The Raychev-Vechev-Yahav Model has major difficulties with loops.
  - Loops that can run for an arbitrary number of iterations will yield infinitely many sentences.
  - Even when bounded, this inflates the frequency of n-grams in loops.
- The results aren't reported as the likelihood of the correct completion.
  - Instead, the actual code's placement in a list of top suggestions is given.
  - It's not immediately clear what parts of the model are effective.

Modifications

- To address the loop issues, we proposed generating the sentences by walking along the program's parse tree.
  - *levels*: statements at a certain variable scope form sentences
  - *cfs*: an acyclic version of the CFG is used
- To better illuminate the likelihood of code, we generalized from the original trigram model.
  - The *call model*, which generates the function call to test, was modified to use a MEMM.
  - We also reported results for unigram and bigram.

Implementation

- The original paper restricted its attention to Android's Java API.
- Our corpus was thus taken from GitHub projects in this area.
- We had to make our own program analysis tool.
  - We used the PLYJ Java parser to get raw parse trees, and then performed significant modification to yield a simplified representation.
- We also included a random baseline prediction.

Performance

- We usually didn't get the right completion…
  - We had difficulty with static fields—without detailed type analysis, we had to treat them like constants, and couldn't dynamically place them.
  - The model was weighted towards short sentences—with the exception of fields, many variables only generate one or two events. As a result, the model assigned greater likelihood to completions that only used constants.
  - As in the previous model, we couldn't handle expressions in function calls.
- However, we did find out the most important elements of the model.

Numbers

call model	**3**	1.3801695908	1.4446105766	1.471347531
	2	1.3801695908	1.4446105766	1.471347531
	1	1.3794449235	1.4479395829	1.4694658828
	MEMM	1.2973448348	1.3617000792	1.3816577024
levels, max		**1**	**2**	**3**
			var model	

call model	**3**	1.3502175602	1.3991302954	
	2	1.3675182533	1.3854038403	
	1	1.3671579219	1.3908003662	
	MEMM	1.2811909286	1.3039491778	
levels, random baseline		**1**	**2**	
			var model	

call model	**3**	1.2526546072	1.2952245955	1.310383277
	2	1.2526546072	1.2930782423	1.3067884675
	1	1.252504953	1.2877275064	1.2984496871
	MEMM	N/A	N/A	N/A
CFS, max		**1**	**2**	**2**
			var model	

call model	**3**	1.2376532236	1.2616989967	
	2	1.2263363548	1.2873724219	
	1	1.2234551087	1.2464570138	
	MEMM	N/A	N/A	
CFS, random baseline		**1**	**2**	
			var model	

Conclusions

- The limited information available from raw code severely limited predictive abilities.
  - We could include additional information about the Android API, although the results would then not generalize.
  - A more powerful program analysis tool would help.
- The MEMM showed great promise.
  - It performed on par with the other models. However, it had a minimal number of features.
  - It also wasn't yet considering previous words during tag computation.
- We hope to continue this work!