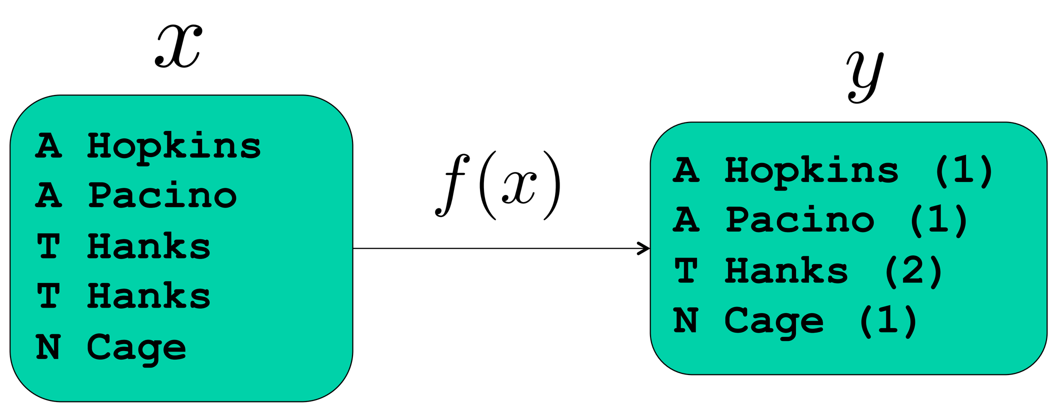
# A Machine Learning Framework for Programming by Example

Aditya Krishna Menon, Omer Tamuz, Sumit Gulwani, Butler Lampson, Adam Kalai

## Motivation: text processing

In Programming by Example, a user describes a task by providing an example of its operation. We'll focus on text processing tasks, such as:




which is meant to express the string transformation (or program)

```
f(x) = dedup(concat(x," ",
          concat("(",count(x,x),")")))
```

**Q**: Given (*x, y*), can we learn *f*?

**A**: Given a library of subroutines – e.g. `dedup`, `concat` – we could search over compositions of them.

**Catch**:
- Naïve search is not scalable!
- How to rank all consistent *f*'s?

## Our approach

We propose an ML framework to speed up search *f* by:
- Representing a program as the derivation of some PCFG
- Defining clues that link features of (*x, y*) to likely PCFG rules
- Learning each clue's reliability, thus determining the PCFG probabilities

In the above example, the most likely grammar probabilities could be e.g.:

| Rule | Prob. |
|---|---|
| LIST→split(x,DELIM) | 0.3 |
| LIST→concat(CAT,CAT,CAT) | 0.3 |
| LIST→dedup(LIST) | 0.2 |
| LIST→count(LIST,LIST) | 0.2 |

With these probabilities in hand, we search over programs in order of how their grammar probability i.e. the most probable consistent *f* is chosen.

## PCFG representation

We define a PCFG where each rule corresponds to a particular subroutine. A program is a trace under the PCFG, viz. composition of rules (i.e. subroutines). The probability of a program for a given input *z* = (*x,y*) thus depends on the probabilities of its constituent rules:

$$\Pr[f|z;\theta] = \prod_{r\in\mathcal{R}_f} \Pr[r|z;\theta]$$

## Clues

Clues connect features of the input to the grammar rules they suggest may be useful. This injects domain knowledge into the problem.

| | |
|---|---|
| String *s* in input but not output? | E → *s* |
| Duplicates in input but not output? | LIST → { E } |
| Numbers on each input but not output line? | LIST → dedup(LIST) |
| … | LIST → count(LIST) |
| | … |

Formally, a clue is a function that takes as input the example (*x,y*), and returns a subset of grammar rules.

## Probability model

We use a log-linear model for the PCFG probabilities. The model posits that the probability of a grammar rule is proportional to the reliability of all clues that suggest that rule.

$$\Pr[r|z;\theta] \propto \exp\left(\sum_{i:r\in c_i(z)} \theta_i\right)$$

The parameters $\theta$ are estimated by maximizing the log-likelihood of the training data.

## System usage

Once the system is trained, we may apply it to a new input (*x, y*) as follows:

1. Evaluate each clue on (*x, y*).
2. Using the probability model, assign probabilities to the grammar rules.
3. Enumerate over programs in decreasing order of probability, and return the first consistent with (*x,y*).

## Experimental setup

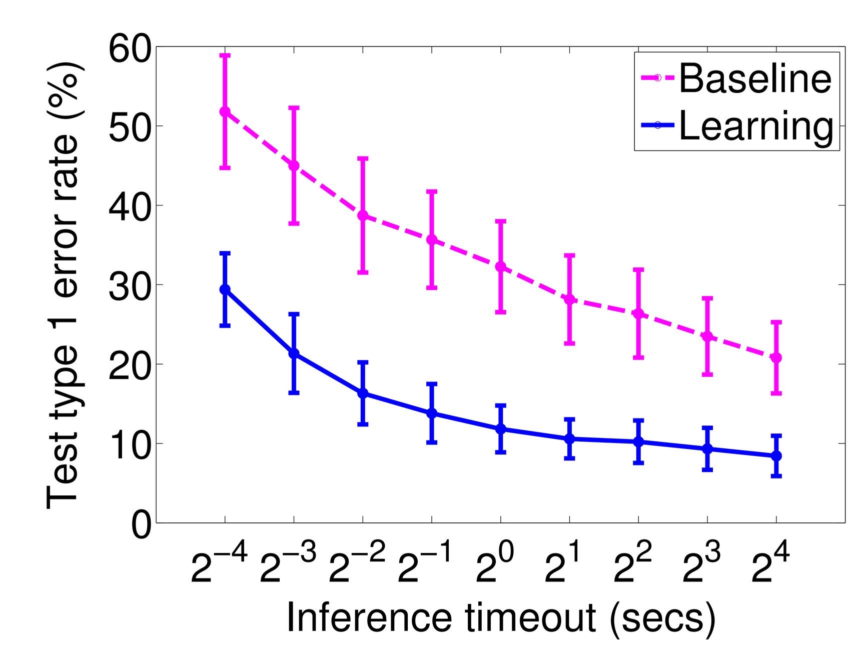
We developed a prototype of our learning approach, based on a library of around 100 base subroutines and around 100 clues.

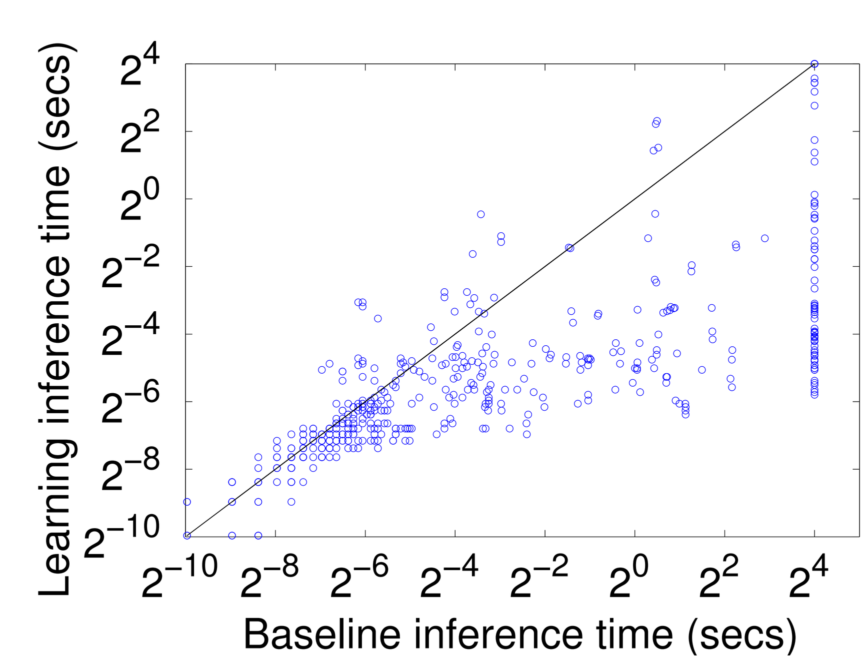
We evaluated the learning method on a corpus of 280 examples, largely based on queries from Excel help forums. Sample cases:

| Input | Output |
|---|---|
| Adam Ant\t1A St. \t90113 | 90113 |
| 28/6/2010 | June the 28th 2010 |
| 612 Australia | case 612: return Australia; |

## Experimental results

Learning dramatically lowers the error rate and inference time compared to naïve search.




Further, learning is able to learn more nested function compositions than the naïve search.