Many tools stand to benefit from models that accurately capture the statistics of natural programs.

Choosing the right representation for programs is important because it determines what information is available to models.

Previous Work: Programs are Token Sequences

Running Example:

```java
class Sagan extends Astronaut {
  void beforeBed() {
    Planet favorite = Planet.NEPTUNE;
  }
}
class Astronaut {
  void view(Planet p) {...}
}
```

Previous work has considered only tokenized representations of programs. This makes it possible to borrow techniques from the vast literature on natural language processing.

- n-grams (Hindle et al., ICSE 2012)
- topic modeling + part of speech analysis (Nguyen et al., FSE 2013)

Our Approach: Programs are Typed Syntax Trees

But programs, unlike natural sentences, are objects of a synthetic formal system specified by a well-defined grammar and typing rules. We are developing models that assign probabilities to these mathematical objects — typed syntax trees — directly.

This choice makes it easier to structure our models, incorporate information available during parsing and type checking, and ensure that only valid expressions have non-zero probability.

The probability distribution we are interested in is:

$$P(e | \tau, \rho, \Gamma) = \sum_{\phi \in \Phi} P(e | \phi, \tau, \rho, \Gamma) P(\phi | \tau, \rho)$$

This factorization can be understood as a Bayesian network:

Defining and parameterizing the conditional distribution functions for the form and expression determine a particular language model.

We are implementing our statistical model of Java using the Eclipse JDT, which performs the necessary parsing and tracking of the typing context, and will serve as the basis of future applications.

This project is called Syzgy and our current codebase is openly developed and licensed on Github:

http://github.com/cyrus-/syzgy

To build an initial model of Java expressions, we made the following choices:

- An expression can have the role of a top-level statement, an argument to a method call, the target of a dot operation (e.x), the right-hand side of an assignment, or some other role.
- The typing context tracks the types of local variables, the classes and interfaces that have been imported, their inheritance relationships and the type signatures of their methods and fields.
- The distribution of forms is determined empirically. Here, # indicates a count parameter: the number of expressions in the training set that have the specified characteristics, summing over omitted characteristics.

Variables are drawn from a uniform distribution over all variables of the right type in scope.

Numerical literals are drawn from a double exponential (Laplacian) distribution with additional weight given to literals seen in the training corpus.

Field accesses are drawn by independently considering the target expression and then drawing from a uniform distribution over its fields with additional weight given to fields seen during training.

Other forms follow similar patterns: subexpressions are considered mutually independent and observations in the training corpus increase the weight of an expression against a simple/uniform baseline distribution.

An early implementation of this model predicts expressions as well as or better than the state-of-the-art as of 2012.