1 Algorithm Overview

1.1 Network construction

As we mentioned before, the network models the given java libraries. We aim to model method calls by which types are activated by the given sequence, and how many variables of the given type are generated.

Given a set of java libraries, we use Soot to parse the signature of each method in each library. We collect a set of types involved in the given signatures, and each type will be a node in our network. Then, we add transition node that represents each method. Each transition node of a method will have in edges from its input types and out edge to its return type.

One important variation to handle in object-oriented programming languages is polymorphism of types. To handle polymorphism, in addition to the original set of methods, we add an additional transition from any subclass to its superclass, therefore any subclass can be considered as its superclass by an additional transition.

1.2 Reachability analysis

The reachable path of the graph is defined as a subset of the transitions (or methods) activated, such that the return type is activated by the method calls. We find the set by modeling the problem as an SAT problem.

Variables: for each of the n methods, we create a variable m_i, true if and only if the method is activated. For each of the k type, we create a variable t_i, true if and only if the type is activated. Define Input(i) be the number of inputs of type i.

Constraint #1: each method requires all its input types all activated.
The corresponding SAT constraint:

$$\forall i < n, m_i \rightarrow \bigwedge_{t_j \in in(m_i)} t_j$$

Constraint #2: each non-input type requires at least one method that generates this type to be activated.
The corresponding SAT constraint:

$$\forall i < ki \notin inputs, t_i \rightarrow \bigvee_{m_j \in in(t_i)} m_j$$

Constraint #3: all input types and the return type should be activated.
The corresponding SAT constraint: t_{return} true and $\forall t_i \in inputs, t_i$ true.

Constraint #4: by our hypothesis, each return value should be used at least once. To make sure each return value is used at least once, for each type, the number of times of usage should be more than the number of variables generated of that type.
Specifically,

$$\forall i < k, Input(i) + \sum_{m_j \in in(t_i)} m_j \leqslant \sum_{m_j \in in(t_i)}$$

With all these constraints, in most cases, we can gaurantee that there exists at least one sequence of method calls such that we can find a compilable program, and there is no unused variable.