PROLOG BASED APPROACH TO EVALUATE JAVA PROGRAMS

Ananta Kumar Das

IIIT-B

Content

- What is Prolog
- Properties of Java Program
- Java To Prolog conversion
- Program evaluation using Prolog
- Examples
- Planned Enhancement

Prolog

- A logic programming language
- A prolog program consists of *facts* and *rules*
- Fact → states a certain tuple of values satisfies a predicate *unconditionally*.
- Rule → is a Prolog statement which gives conditions under which tuples satisfy a predicate.

- **Example 1 – A Directed Graph**
  - edge(a,b).
  - edge(a,e).
  - edge(b,d).
  - edge(b,c).
  - edge(c,a).
  - edge(e,b).

Facts

```
edge(Node1,Node2) :-
        edge(Node1,SomeNode),
        edge(SomeNode,Node2).
```

Rule

Properties of Java Programs

- Few properties of a Java Program –
  - Use Recursion to find factorial of Number
  - Class A extends class B
  - Method M1 overloads/overrides method M2
  - Encapsulation is implemented in Program
- Intended to evaluate programs written by students
- Concentrating on properties other than runtime properties and programming logic.

Java To Prolog conversion (using JTransformer)

JTransformer represents the Abstract Syntax Tree (AST) of Java and information about projects, directories and files by Prolog facts that we call **Program Element Facts** (**PEFs**). The image below illustrates the idea.

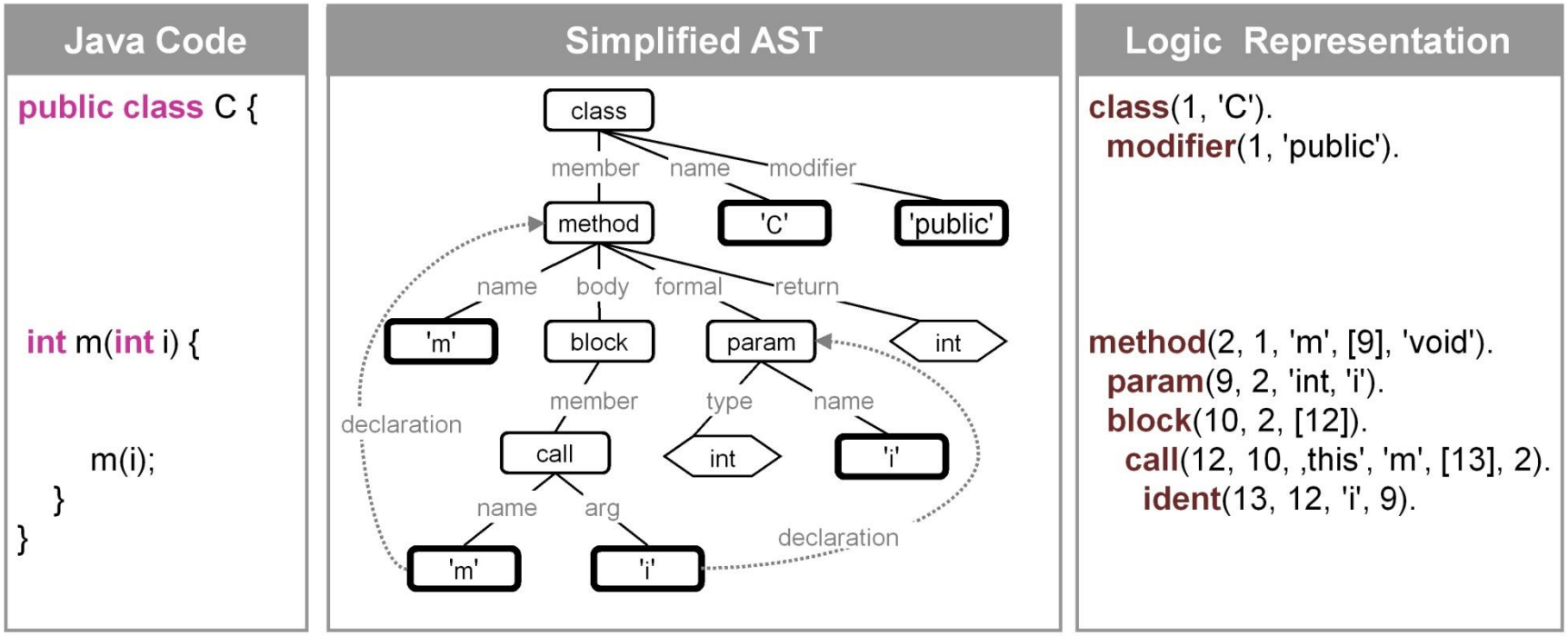
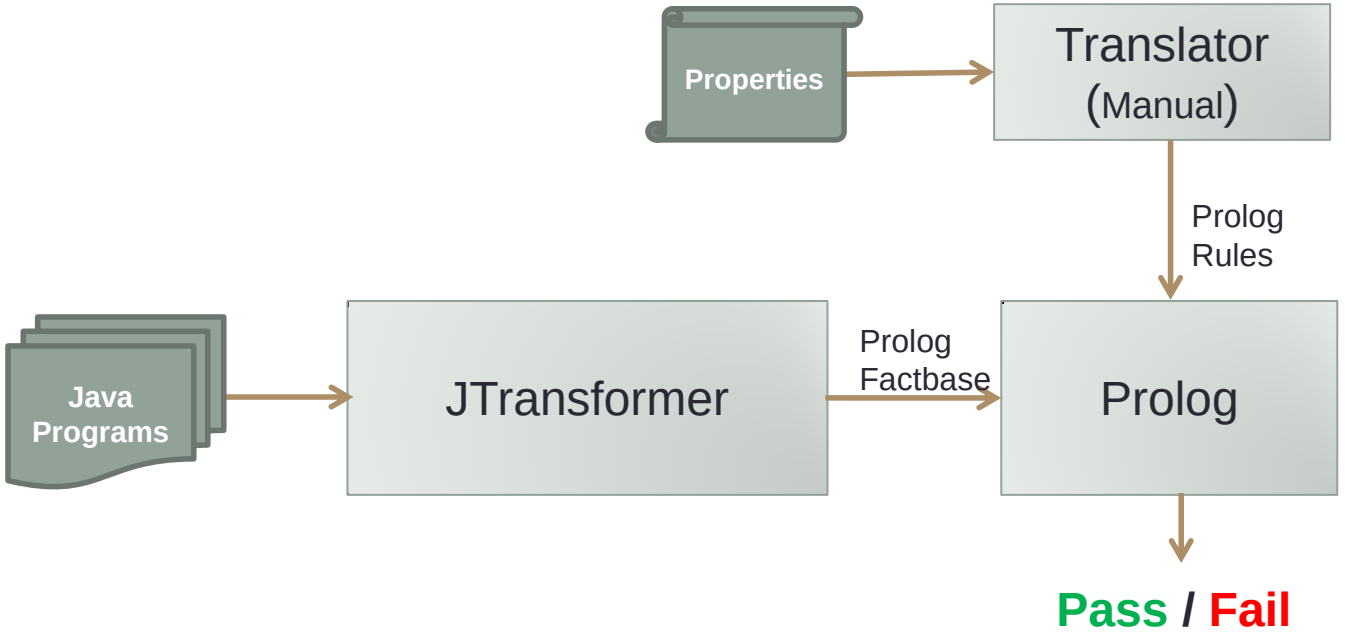


Fig : Representation of Java abstract syntax tree by Program Element Facts.

Program evaluation using Prolog



- JTransformer translates the Java program to a Prolog knowledge base.
- We (manually) translate the question into a Prolog query.
- Prolog runs the query over the knowledge base to check satisfiability.

Example (Find recursive methods)

4.1. Example Java Code

```
public class SubClass extends BaseClass{
  public static void main(String []s){test1(0);}
  static void test1(int i){test1(0);}
  static void test2(int i){test2(0);}
}
```

JTransformer

4.2. FACTBASE of code 2.1.

```
classT(ClassT,CompilationUnitT,'SubClass',[],[ConstructorT,MethodT,MethodT_1,MethodT_2]),

  methodT(MethodT,ClassT,main,[ParamT],BasicTypeT,[],[],BlockT),
        paramT(ParamT,MethodT,ArrayTypeT,s),
        blockT(BlockT,MethodT,MethodT,[ExecT]),
              execT(ExecT,BlockT,MethodT,CallT),
                    callT(CallT,ExecT,MethodT,null,[LiteralT],MethodT_1,[],BasicTypeT),
                    literalT(LiteralT,CallT,MethodT,BasicTypeT_1,'0'),

  methodT(MethodT_1,ClassT,test1,[ParamT_1],BasicTypeT,[],[],BlockT_1),
        paramT(ParamT_1,MethodT_1,BasicTypeT_1,i),
        blockT(BlockT_1,MethodT_1,MethodT_1,[ExecT_1]),
              execT(ExecT_1,BlockT_1,MethodT_1,CallT_1),
                    callT(CallT_1,ExecT_1,MethodT_1,null,[LiteralT_1],MethodT_1,[],BasicTypeT),
                    literalT(LiteralT_1,CallT_1,MethodT_1,BasicTypeT_1,'0'),

  methodT(MethodT_2,ClassT,test2,[ParamT_2],BasicTypeT,[],[],BlockT_2),
        paramT(ParamT_2,MethodT_2,BasicTypeT_1,i),
        blockT(BlockT_2,MethodT_2,MethodT_2,[ExecT_2]),
              execT(ExecT_2,BlockT_2,MethodT_2,CallT_2),
                    callT(CallT_2,ExecT_2,MethodT_2,null,[LiteralT_2],MethodT_2,[],BasicTypeT),
                    literalT(LiteralT_2,CallT_2,MethodT_2,BasicTypeT_1,'0'),
```

4.3. Prolog Code to find if Method is recursive

```
findrecursiveMethod(MethodName) :-
        methodT(MethodT_1,ClassT,MethodName,[ParamT_1],BasicTypeT,[],[],BlockT_1),
        paramT(ParamT_1,MethodT_1,BasicTypeT_1,i),
        blockT(BlockT_1,MethodT_1,MethodT_1,[ExecT_1]),
        execT(ExecT_1,BlockT_1,MethodT_1,CallT_1),
        callT(CallT_1,ExecT_1,MethodT_1,null,[LiteralT_1],MethodT_1,[],BasicTypeT),
        literalT(LiteralT_1,CallT_1,MethodT_1,BasicTypeT_1,'0').
```

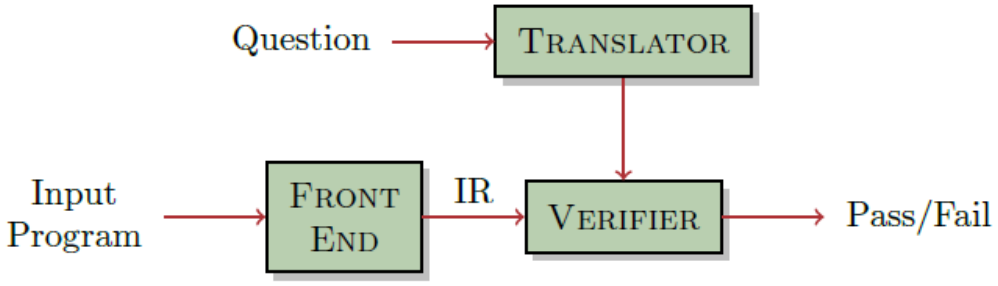
SWI-Prolog

4.4. Output

```
?- findrecursiveMethod(A).

        Correct to: "hello:findrecursiveMethod(A)"? yes
        A = test1 ;
        A = test2.
```

Planned Enhancement (Generalized Static Analysis Framework)



1. **Specification Language** ($L_S : S \in L_S$)

   A specification language which allows a formal declarative method of writing the question.

2. **Question to Specification Translator** ($T : Q \rightarrow S$)

   A translator that reads a question written natural language and translates that into a specification written in a formal logic (e.g. first order predicate logic or Prolog query).

3. **Verification Engine** ($V : S \times IR \rightarrow boolean$)

   A verification engine that can analyze the IR to check if satisfies the specification S.

Links

- https://www.cpp.edu/~jrfisher/www/prolog_tutorial/contents.html
- http://www.learnprolognow.org/
-
- https://sewiki.iai.uni-bonn.de/research/jtransformer/api/java/prologast
- https://sewiki.iai.uni-bonn.de/research/jtransformer/api/java/jtransformer

THANK YOU