Probabilistic Model Checking of Randomized Java Code

Syyeda Zainab Fatmi, Xiang Chen, Yash Dhamija, Maeve Wildes, Qiyi Tang, and Franck van Breugel

July 12, 2021

Motivation

The Miller-Rabin primality test determines whether a number given as input is prime. The algorithm may erroneously report that the input number is prime.

Motivation

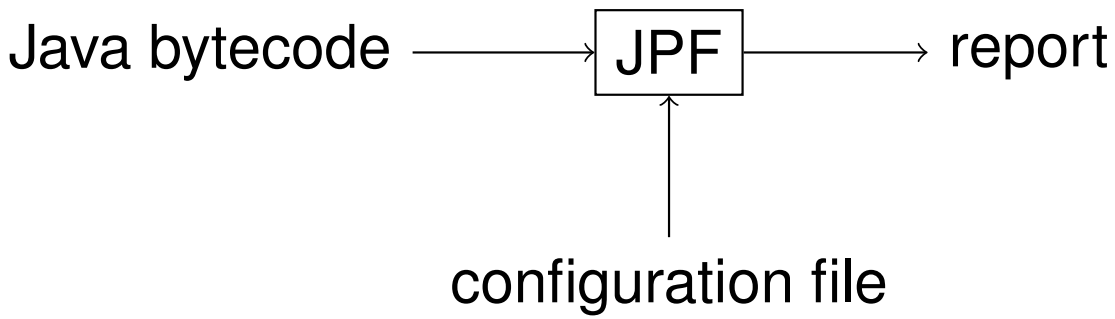
The Miller-Rabin primality test determines whether a number given as input is prime. The algorithm may erroneously report that the input number is prime.

Monte Carlo algorithms are randomized algorithms that may produce incorrect results with a small probability.

Table of Contents

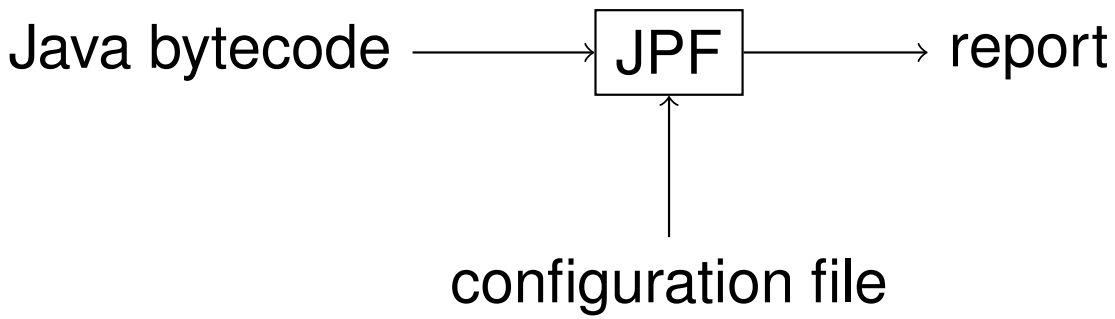
Java PathFinder

Java PathFinder (JPF) is the most popular model checker for Java code.

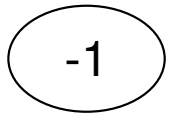


Java PathFinder

Java PathFinder (JPF) is the most popular model checker for Java code.



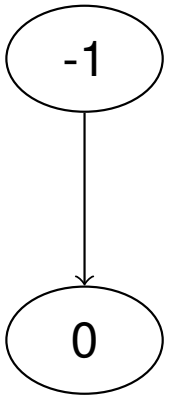
However, JPF cannot handle probabilistic properties.



Running JPF on Java code can be viewed as building on the fly a model of the code.

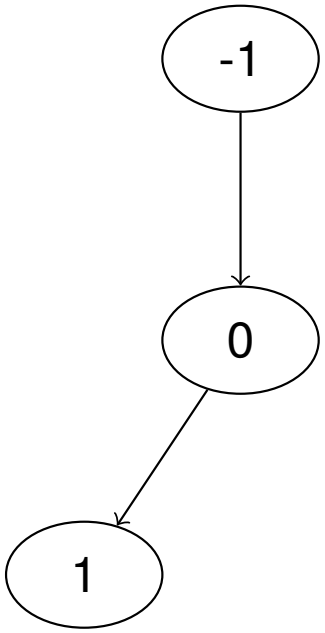
Java PathFinder

Running JPF on Java code can be viewed as building on the fly a model of the code.



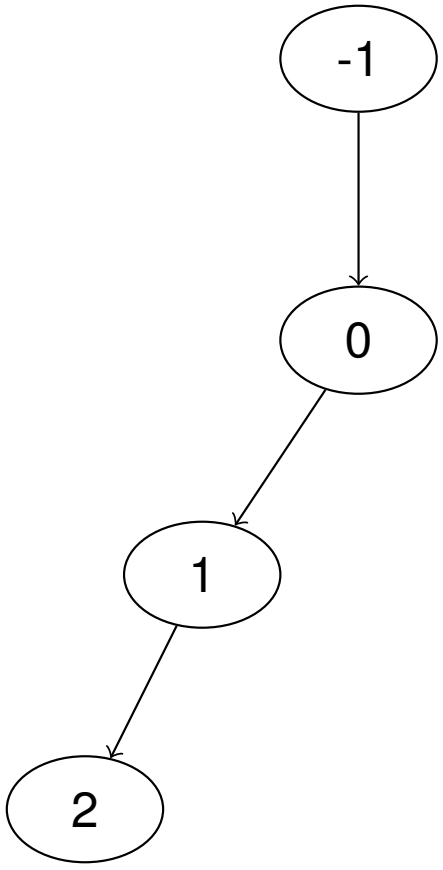
Java PathFinder

Running JPF on Java code can be viewed as building on the fly a model of the code.



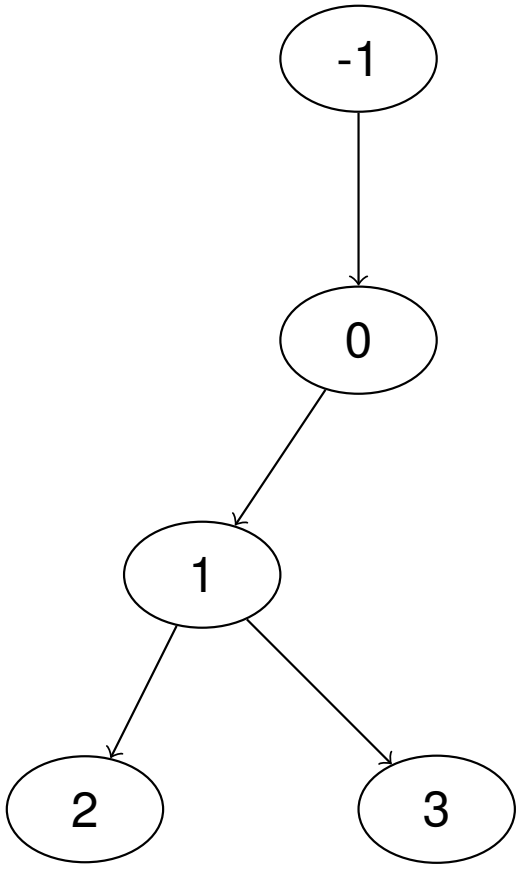
Java PathFinder

Running JPF on Java code can be viewed as building on the fly a model of the code.



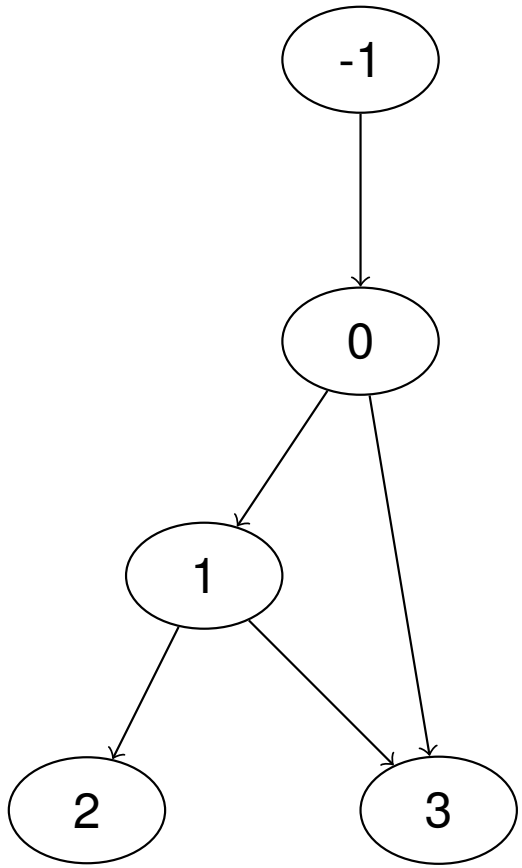
Java PathFinder

Running JPF on Java code can be viewed as building on the fly a model of the code.



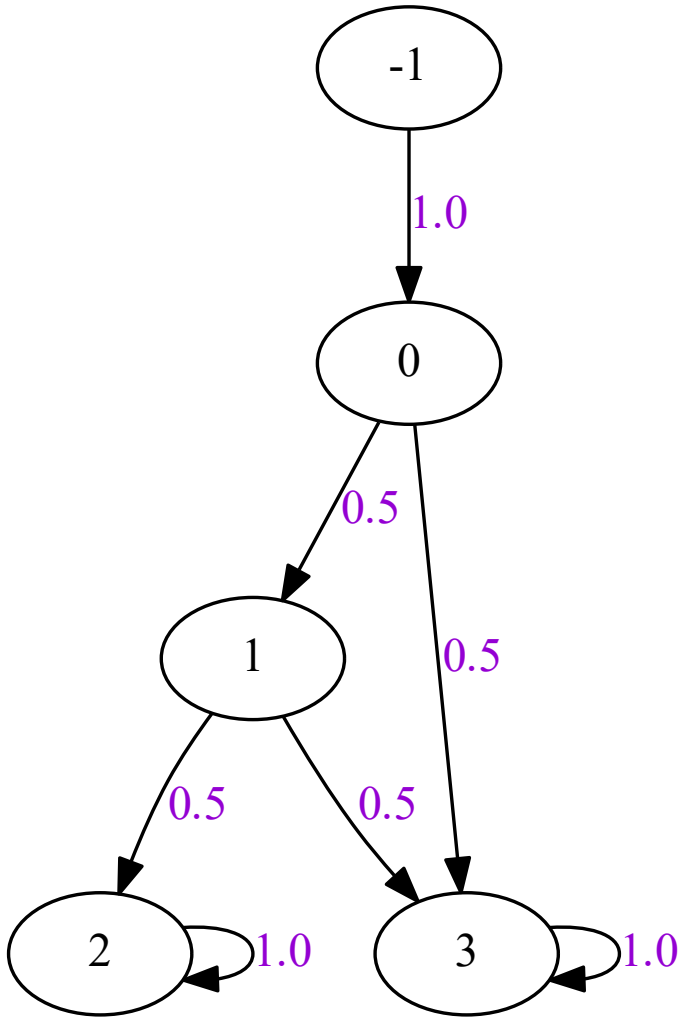
Java PathFinder

Running JPF on Java code can be viewed as building on the fly a model of the code.



jpf-probabilistic

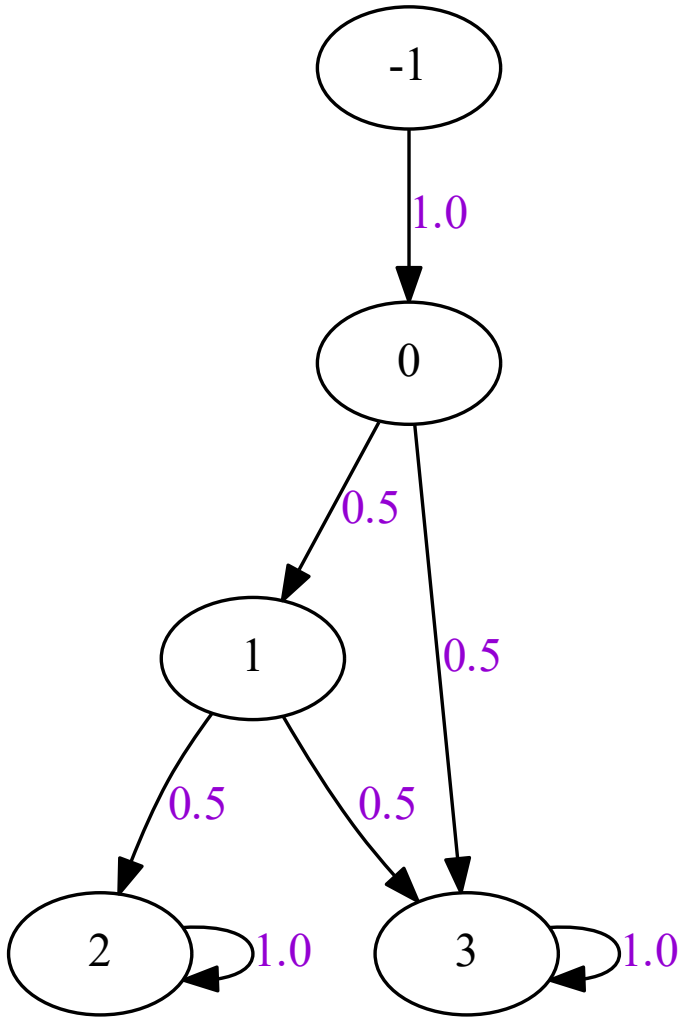
jpf-probabilistic assigns probabilities to the transitions of the model, which reflect the random choices in the Java code.



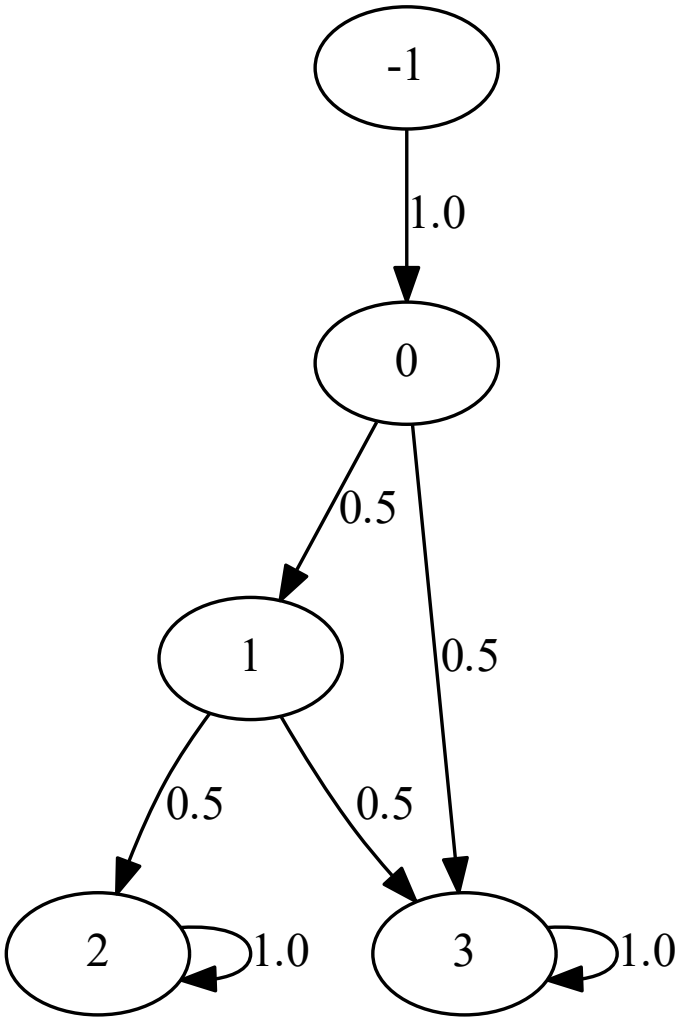
jpf-probabilistic

jpf-probabilistic assigns probabilities to the transitions of the model, which reflect the random choices in the Java code.

This turns the state space into a discrete-time Markov chain (DTMC).



jpf-probabilistic




```
1  5 7
2  -1 0 1.0
3  0 1 0.5
4  1 2 0.5
5  2 2 1.0
6  1 3 0.5
7  3 3 1.0
8  0 3 0.5
```

jpf-probabilistic

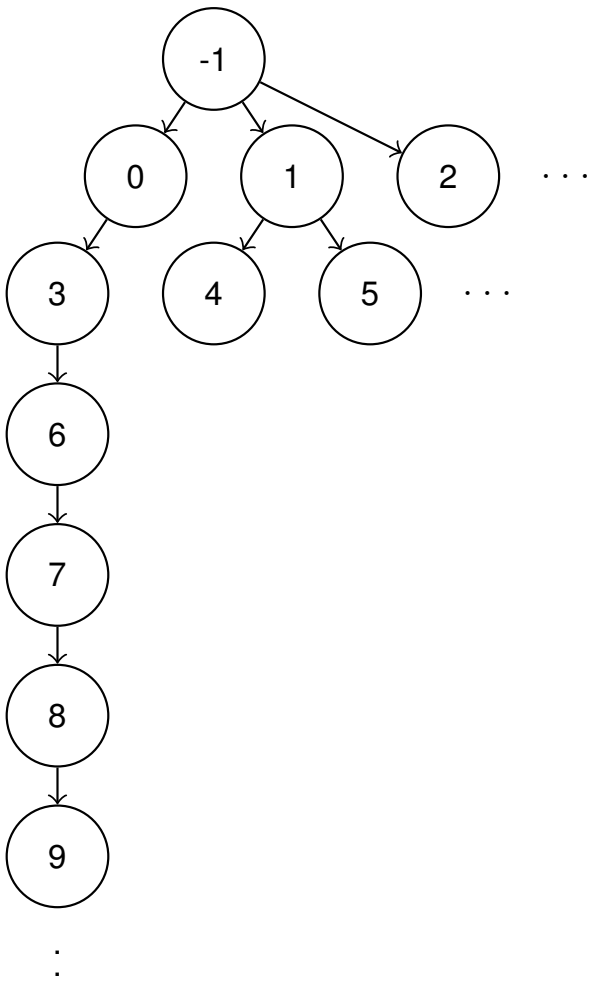
To traverse the state space, JPF supports

- depth-first search (DFS) and
- breadth-first search (BFS).

jpf-probabilistic

To traverse the state space, JPF supports

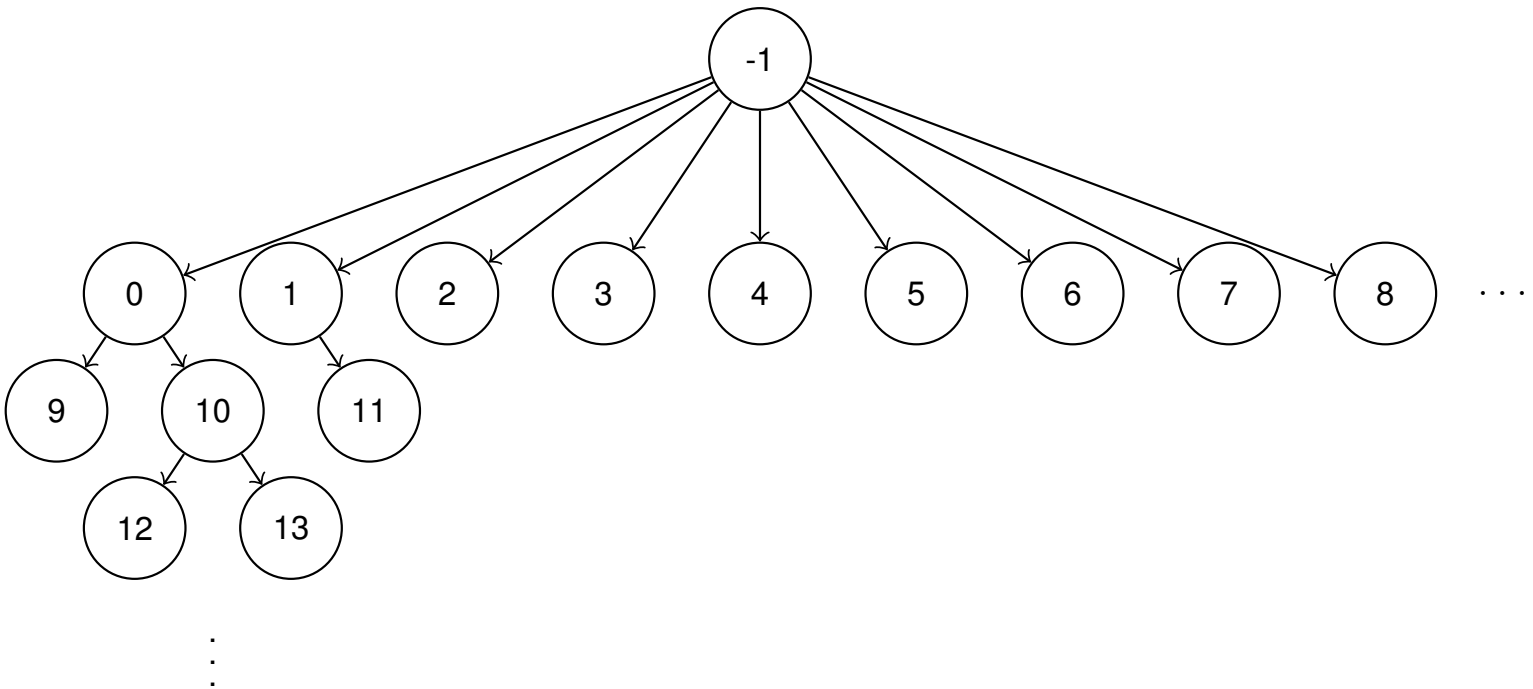
- depth-first search (DFS) and
- breadth-first search (BFS).



jpf-probabilistic

To traverse the state space, JPF supports

- depth-first search (DFS) and
- breadth-first search (BFS).



jpf-probabilistic

To traverse the state space, JPF supports

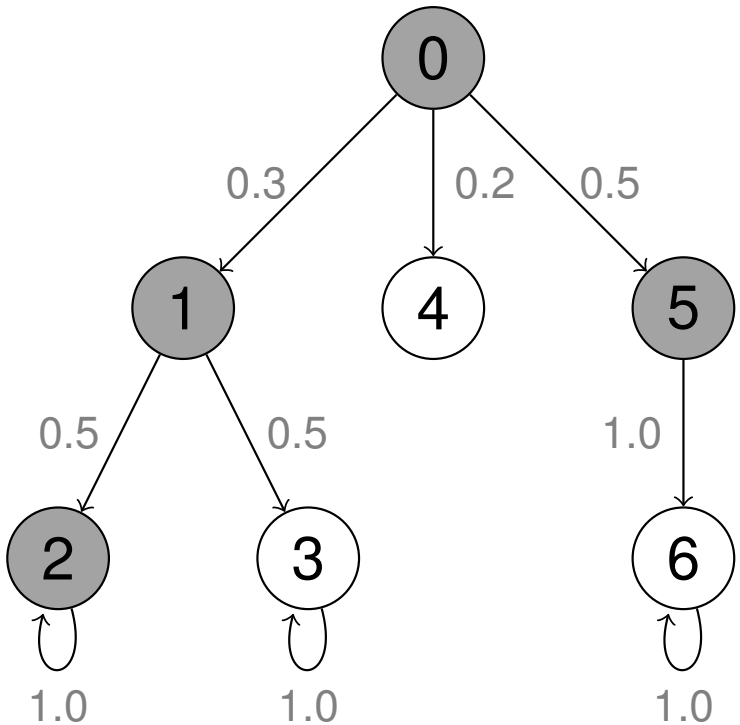
- depth-first search (DFS) and
- breadth-first search (BFS).

jpf-probabilistic provides

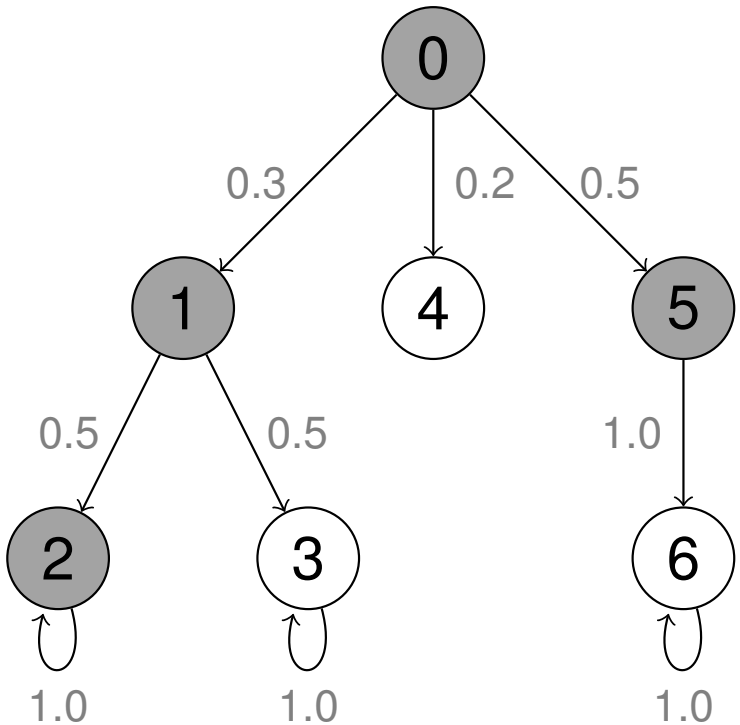
- probability-first search (PFS),
- random search (RS),
- softmax search (SMS), and
- ϵ-greedy search (EGS).

jpf-probabilistic

1. PFS
   always chooses a state whose path along which it is discovered has the highest probability.

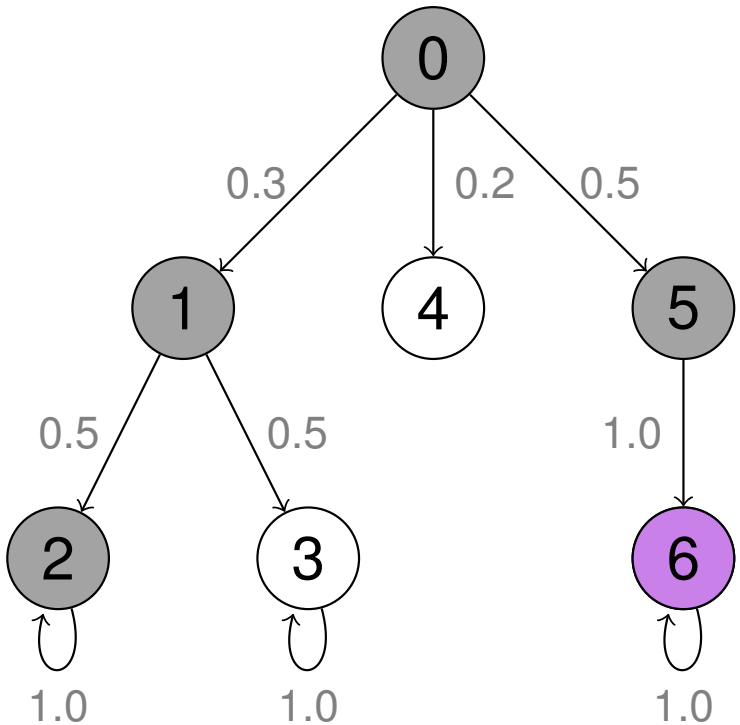


1. PFS
   always chooses a state whose path along which it is discovered has the highest probability.



$$p(s_3) = 0.15$$
$$p(s_4) = 0.2$$
$$p(s_6) = 0.5$$

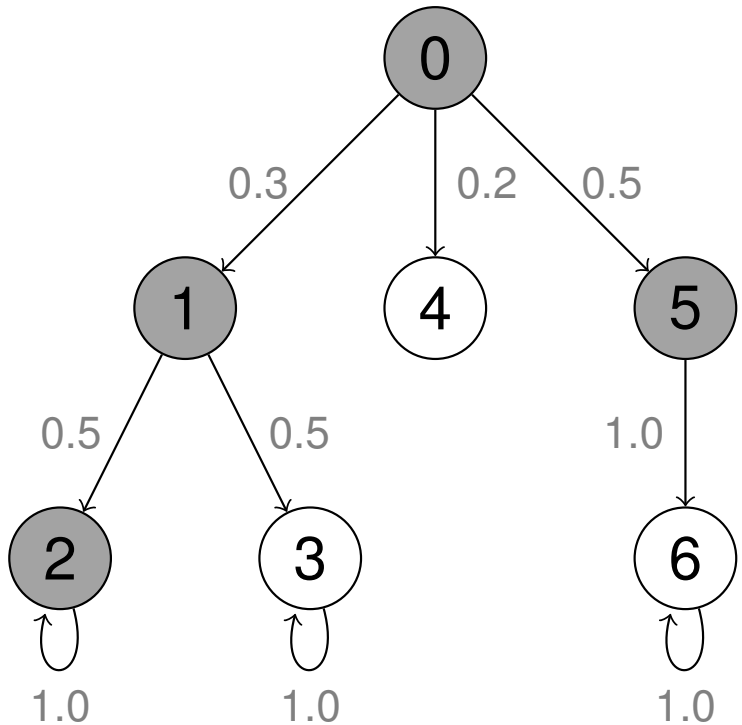
1. PFS
   always chooses a state whose path along which it is discovered has the highest probability.



$$p(s_3) = 0.15$$
$$p(s_4) = 0.2$$
$$p(s_6) = 0.5$$

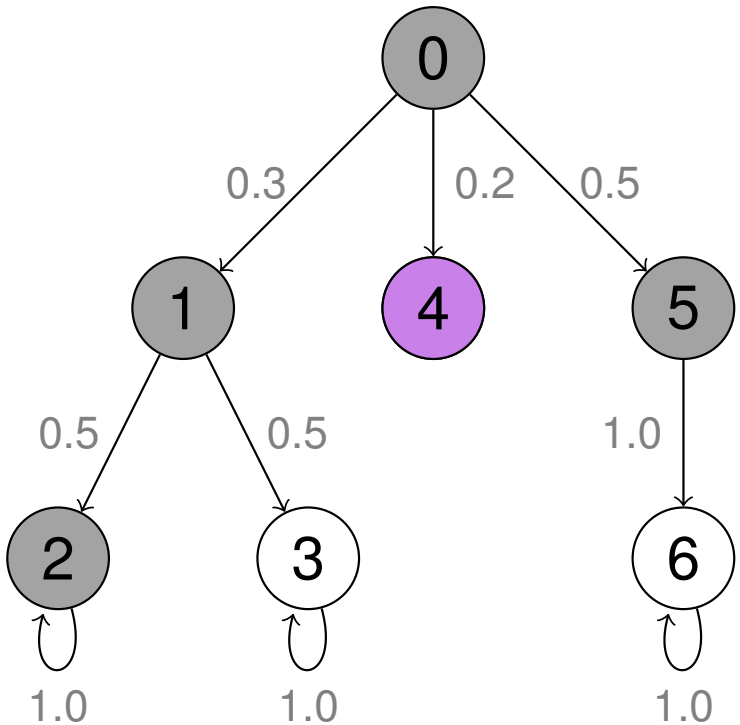
2. RS
   chooses state s_j with probability

$$\frac{p(s_j)}{\sum_{0 \le i \le n} p(s_i)}.$$



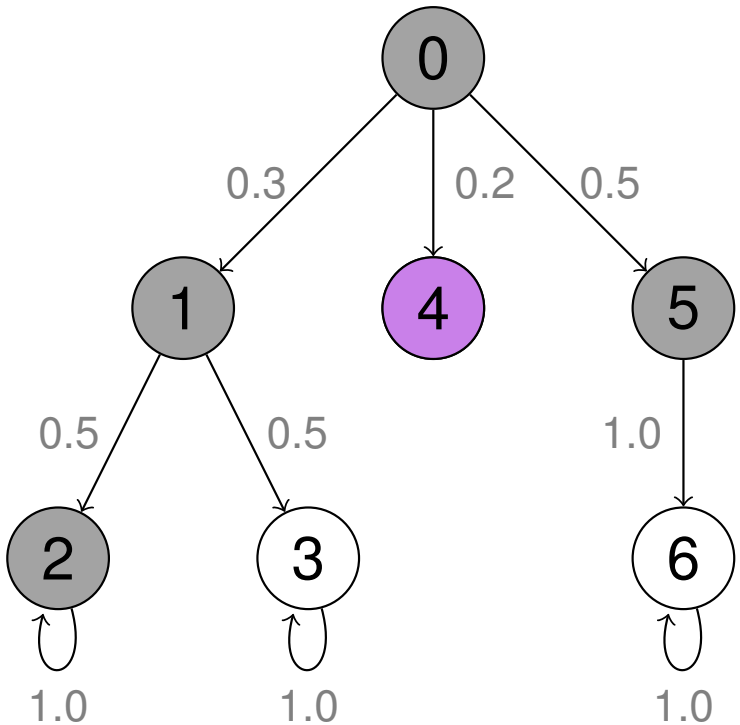
2. RS
   chooses state s_j with probability

$$\frac{p(s_j)}{\sum_{0 \le i \le n} p(s_i)}.$$



2. RS
   chooses state s_j with probability

$$\frac{p(s_j)}{\sum_{0 \le i \le n} p(s_i)}.$$

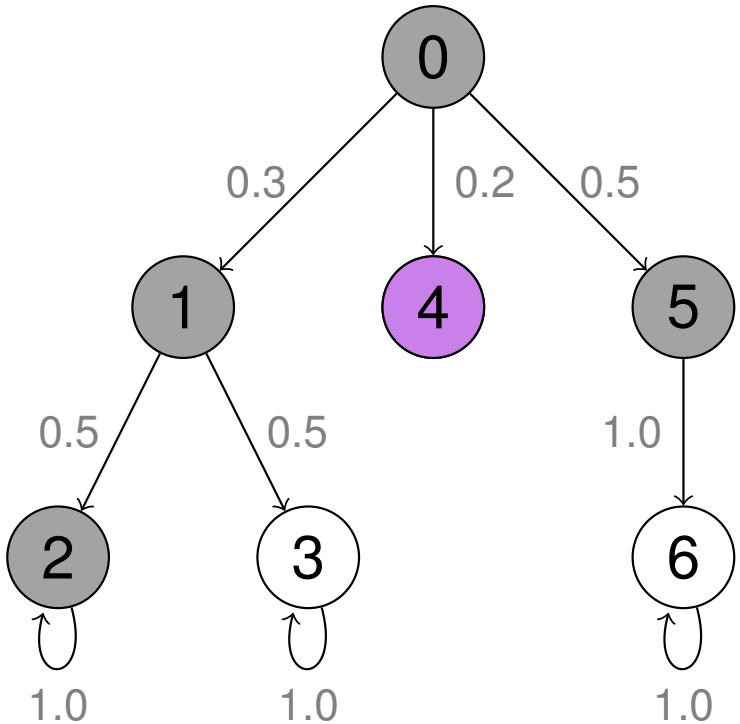


$$p(s_3) = 0.15$$
$$p(s_4) = 0.2$$
$$p(s_6) = 0.5$$

2. RS
   chooses state s_j with probability

$$\frac{p(s_j)}{\sum_{0 \le i \le n} p(s_i)}.$$

   chooses s_4 with probability 0.235

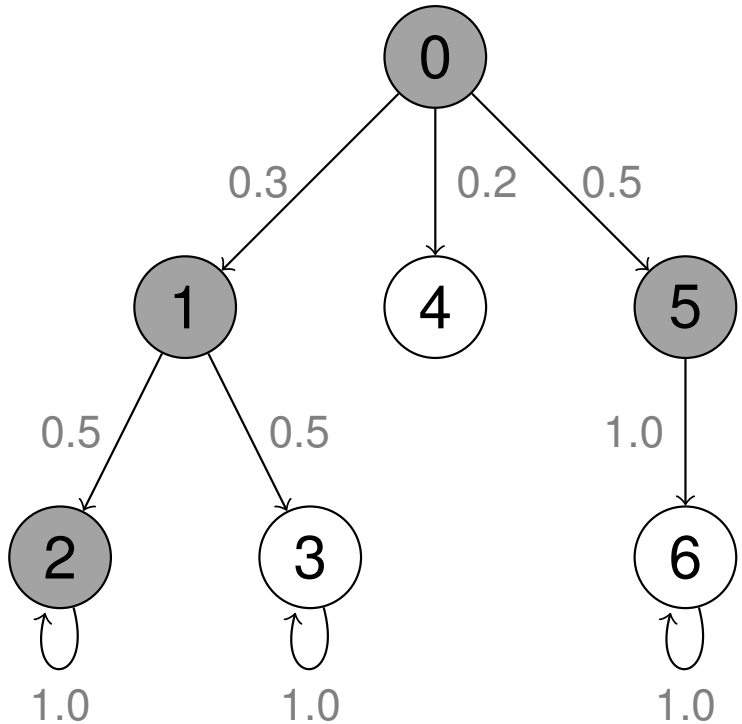


$$p(s_3) = 0.15$$
$$p(s_4) = 0.2$$
$$p(s_6) = 0.5$$

3. SMS
   chooses state s_j with probability

$$\frac{e^{p(s_j)/\tau}}{\sum_{0 \le i \le n} e^{p(s_i)/\tau}},$$

   where τ is the temperature.

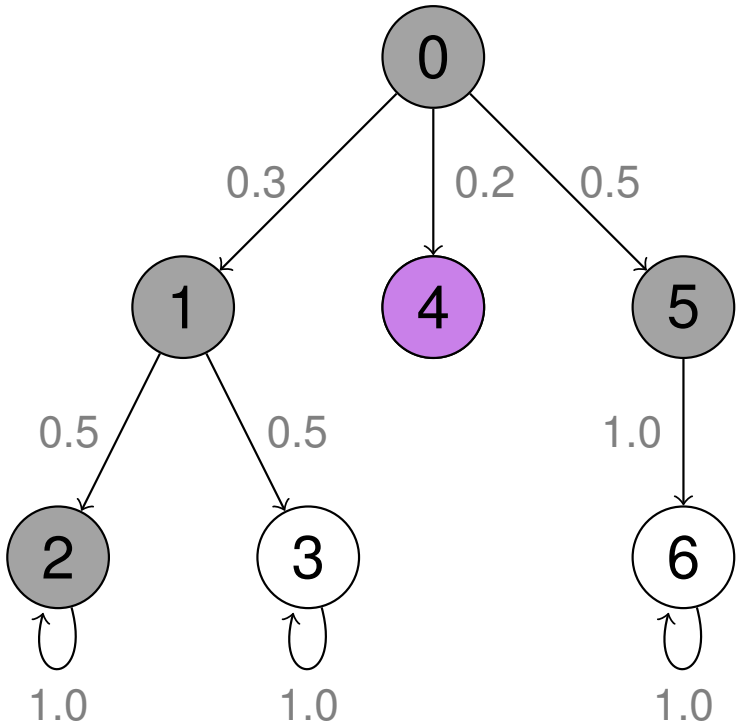


3. SMS
   chooses state s_j with probability

$$\frac{e^{p(s_j)/\tau}}{\sum_{0 \le i \le n} e^{p(s_i)/\tau}},$$

   where τ is the temperature.

   assume $\tau = 2$

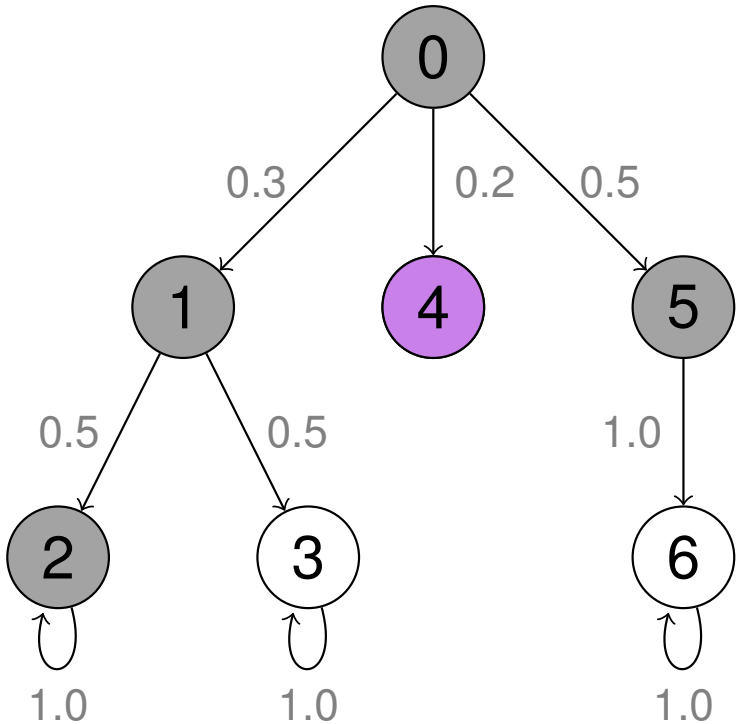


3. SMS
   chooses state s_j with probability

$$\frac{e^{p(s_j)/\tau}}{\sum_{0 \le i \le n} e^{p(s_i)/\tau}},$$

   where τ is the temperature.

   assume $\tau = 2$



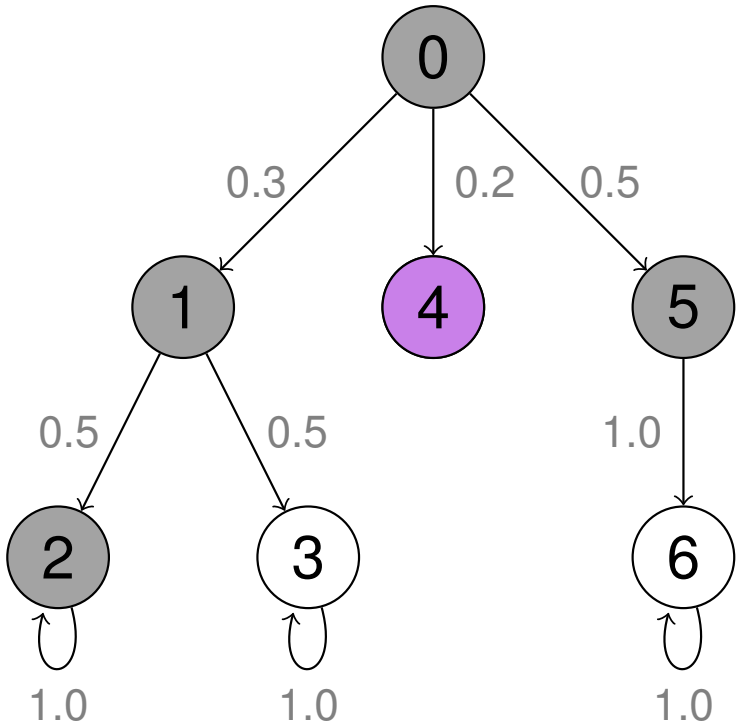
$$p(s_3) = 0.15$$
$$p(s_4) = 0.2$$
$$p(s_6) = 0.5$$

3. SMS
   chooses state s_j with probability

$$\frac{e^{p(s_j)/\tau}}{\sum_{0 \le i \le n} e^{p(s_i)/\tau}},$$

   where τ is the temperature.

   assume $\tau = 2$, chooses s_4 with probability 0.319

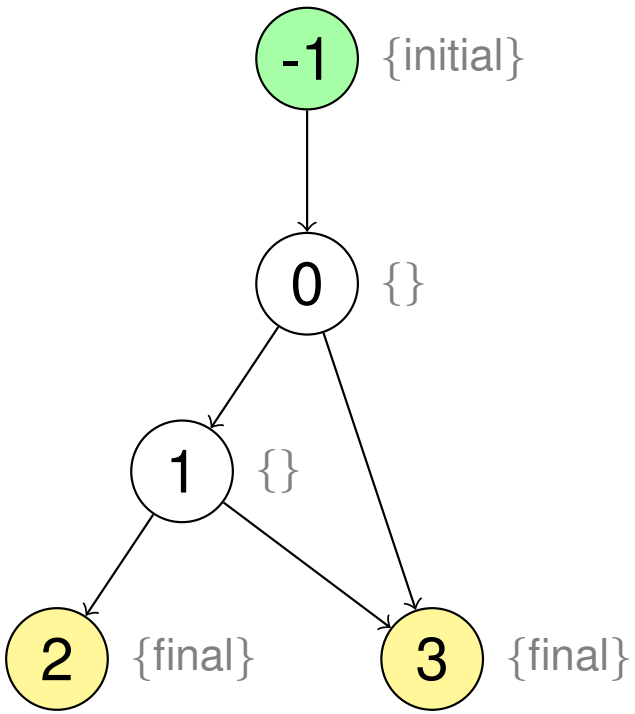


$$p(s_3) = 0.15$$
$$p(s_4) = 0.2$$
$$p(s_6) = 0.5$$

4. EGS
   behaves like PFS with probability $1 - \epsilon$ and behaves like RS and with probability ϵ.

jpf-label

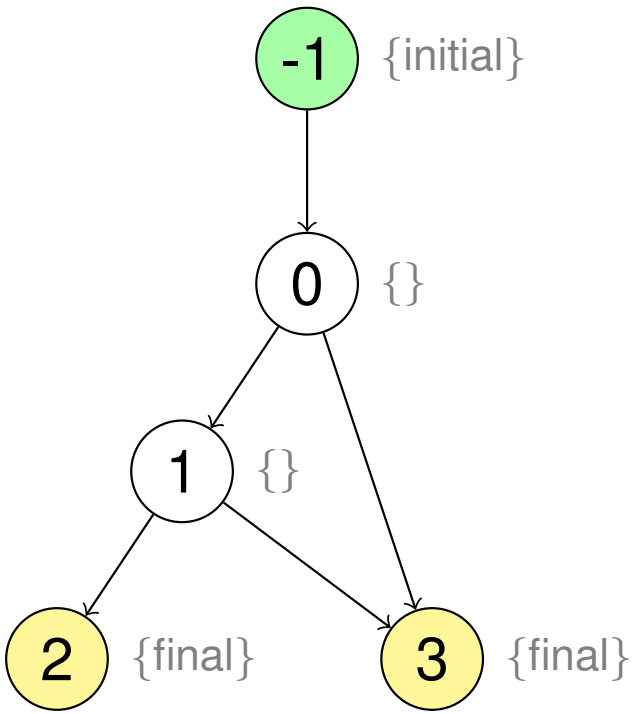
To capture simple known facts about the modelled software, the states are usually labelled with a set of atomic propositions.



jpf-label

To capture simple known facts about the modelled software, the states are usually labelled with a set of atomic propositions.

These atomic propositions may be used to express properties of the code.

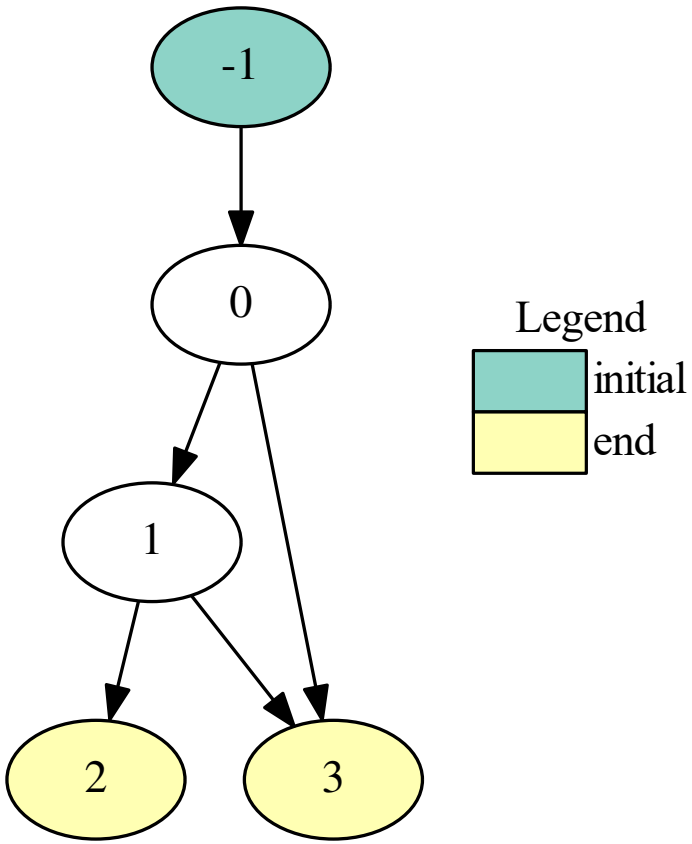


jpf-label

jpf-label allows users to easily label states with atomic propositions, by defining custom labelling functions.

jpf-label

jpf-label allows users to easily label states with atomic propositions, by defining custom labelling functions.




```
1  0="init" 1="end"
2  -1: 0
3  2: 1
4  3: 1
```

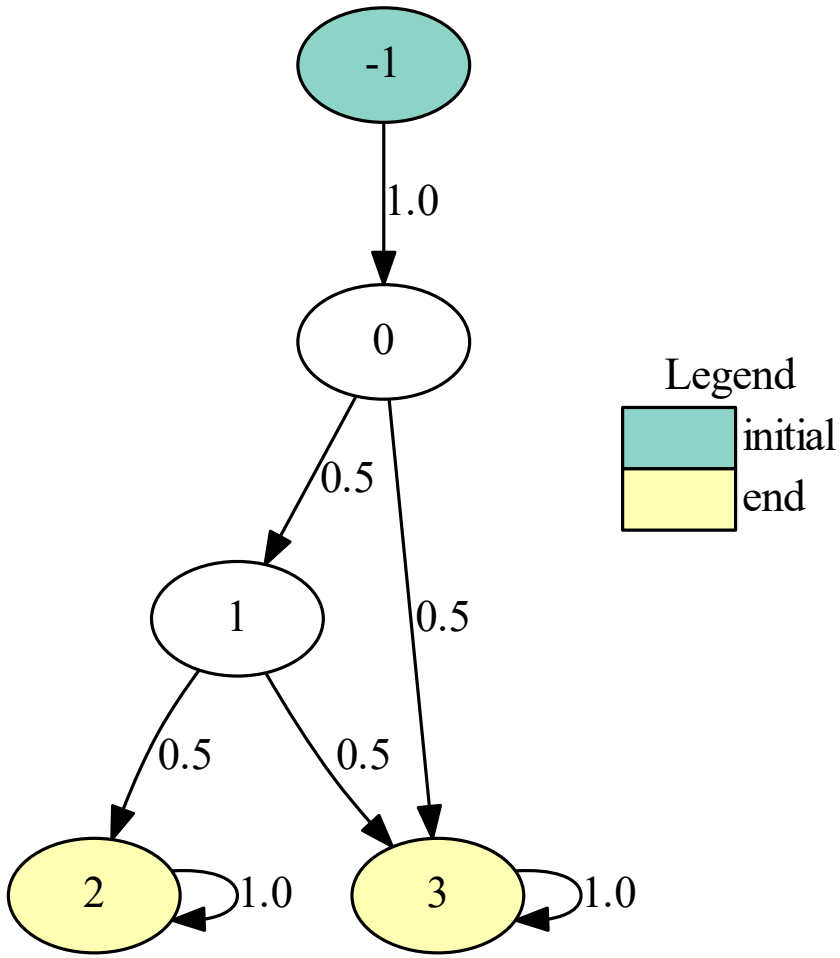
jpf-label

We provide twelve different ways to label states, including:

- initial and final states,
- boolean and integer static fields,
- boolean and integer local variables,
- method invocations,
- method returns and the values returned, and
- thrown exceptions and the exception types.

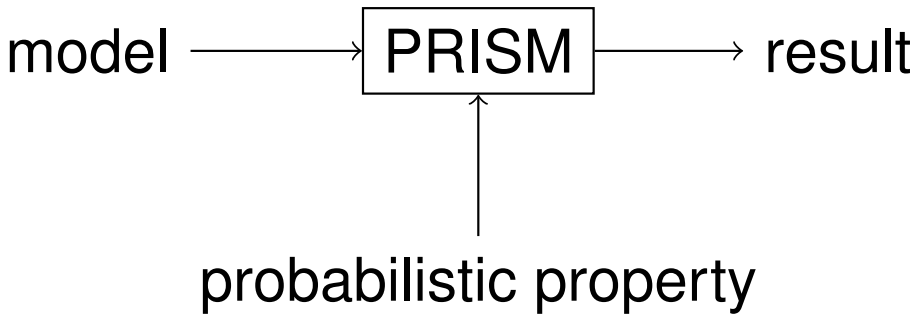
Our extensions

JPF, extended with jpf-probabilistic and jpf-label, can construct a labelled Markov chain that models the given Java code.



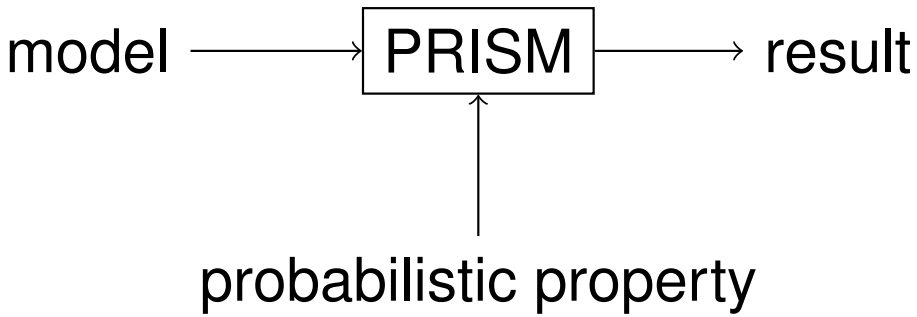
PRISM

PRISM is the most popular probabilistic model checker.



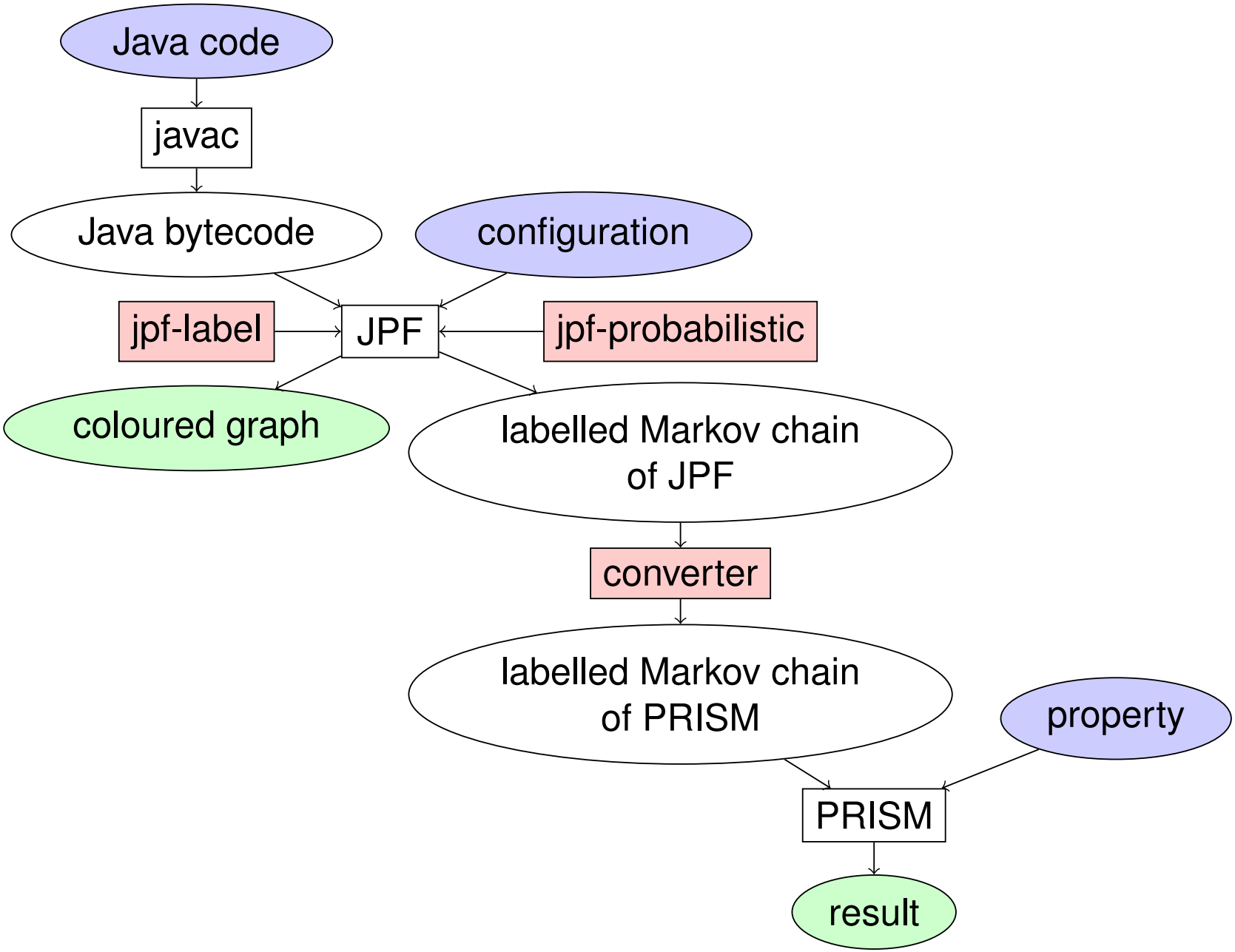
PRISM

PRISM is the most popular probabilistic model checker.



However, some algorithms cannot be easily translated into PRISM's input language.

Our tool


Java code
javac
Java bytecode
configuration
jpf-label
JPF
jpf-probabilistic
coloured graph
labelled Markov chain
of JPF
converter
labelled Markov chain
of PRISM
property
PRISM
result

Miller-Rabin Primality Test

Let us revisit our example and apply our tool to the problem.

Conclusion

Our extensions of JPF, jpf-label and jpf-probabilistic,

- provide an easy way to the label the states
- assign probabilities to the transitions
- introduces new search strategies

Conclusion

Our extensions of JPF, jpf-label and jpf-probabilistic,

- provide an easy way to the label the states
- assign probabilities to the transitions
- introduces new search strategies

Our tool

- builds a bridge between the model checkers JPF and PRISM
- is accompanied by numerous realistic randomized algorithms

Thank You

Our tool can be found on GitHub at:

github.com/javapathfinder/jpf-label
github.com/javapathfinder/jpf-probabilistic