

Depth first search

Franck van Breugel

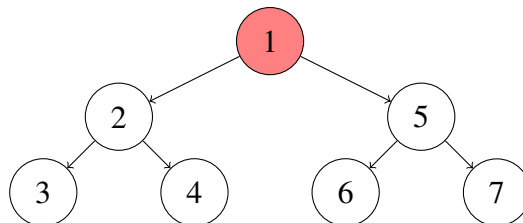
February 11, 2019

1 Constructor

Implement the constructor of the **DFSearch** class.

```
public class DFSearch extends Search {  
    /**  
     * Initialize this search.  
     *  
     * @param config JPF's configuration.  
     * @param vm JPF's virtual machine.  
     */  
    public DFSearch(Config config, VM vm) {  
  
    }  
}
```

2 Forward and backtrack



For the above state space, provide the sequence of calls to **forward** and **backtrack** and the value returned by them corresponding to depth first search started in the top most state.

3 Search

Implement a basic **search** method using **forward** and **backtrack** and loops.

```
public void search() {
```

```
}
```

4 New states

```
public boolean isNewState()
```

tests whether the current state has not been visited before.

Incorporate the **isNewState** method into the **search** method of the **DFSearch** class.

```
public void search() {
```

```
}
```

5 End states

```
public boolean isEndState()
```

tests whether the current state is a final state.

Incorporate the **isEndState** method into the **search** method of the **DFSearch** class.

```
public void search() {
```

```
}
```

6 Ignored states

```
public boolean isIgnoredState()
```

tests whether the current state can be ignored in the search.

States can, for example, be ignored by using in the system under test the method `ignoreIf(boolean)` of JPF's class `Verify` which is part of the package `gov.nasa.jpf.vm`.

Incorporate the `isIgnoredState` method into the `search` method of the `DFSearch` class.

```
public void search() {
```

```
}
```

7 Done

Other components of JPF can end a search by setting the attribute `done` of the class `Search` to true.

Modify the `search` method of the `DFSearch` class to incorporate the `done` attribute.

```
public void search() {
```

```
}
```

8 Request backtrack

Other components of JPF can request a search to backtrack by means of the method

```
public boolean checkAndResetBacktrackRequest ()
```

Modify the **search** method of the **DFSearch** class to incorporate the **checkAndResetBacktrackRequest** method.

```
public void search() {
```

```
}
```

9 Depth

The **Search** class contains the attribute **depth** that can be used to keep track of the depth of the search. It is initialized to zero.

Override the **forward** method of the **Search** class to keep track of the depth.

```
protected boolean forward() {
```

```
}
```

Override the **backtrack** method of the **Search** class to keep track of the depth.

```
protected boolean backtrack() {
```

```
}
```

10 Depth limit

JPF can be configured to limit the depth of the search by setting the JPF property **search.depth_limit**. The default value of **search.depth_limit** is **Integer.MAX_VALUE**. The **Search** class provides the method **getDepthLimit** which returns the maximal allowed depth of the search.

We introduce the following method in the **DFSearch** class.

```
private boolean checkDepthLimit() {  
    return this.depth < this.getDepthLimit();  
}
```

Incorporate **checkDepthLimit** into **forward**.

```
protected boolean forward() {
```

```
}
```

11 Memory usage limit

The JPF property `search.min_free` captures the minimal amount of memory, in bytes, that needs to remain free. The default value is $1024 \ll 10 = 1024^2 = 1,048,576B \approx 1MB$. By leaving some memory free, JPF can report that it ran out of memory and provide some useful statistics instead of simply throwing an `OutOfMemoryError`. The method `checkStateSpaceLimit` of the class `Search` checks whether the minimal amount of memory that should be left free is still available.

Modify the `search` method of the `DFSearch` class to limit the memory usage.

```
public void search() {
```

```
}
```

12 Multiple errors?

The JPF property `search.multiple_errors` tells us whether the search should report multiple errors (or just the first one). The `forward` method also checks whether any property is violated after the unexplored transition has been traversed. If a violation has been detected then the attribute `done` is set to true if and only if JPF has been configured to report at most one error.

The method **hasPropertyTermination** of the class **Search** checks whether a violation was encountered during the last transition. The method returns true if and only if a violation was encountered and the attribute **done** is set to true.

Modify the **search** method of the **DFSearch** class to take **search.multiple_errors** into account.

```
public void search() {
```

```
}
```

13 Notification of start and finish

A search should also notify listeners of particular events by invoking to the methods of the interface **SearchListener**, which can be found in the package **gov.nasa.jpjpf.search**. The **Search** class contains a number of **notify** methods.

Modify the **search** method of the **DFSearch** class to incorporate following notifications.

- **notifySearchStarted**
- **notifySearchFinished**

```
public void search() {
```

```
}
```

14 Notification of forward, backtrack and having fully explored a state

Incorporate following notifications into the **forward** and **backtrack** method.

- **notifyStateAdvanced**
- **notifyStateBacktracked**
- **notifyStateProcessed**

```
protected boolean forward() {  
  
  
  
  
  
  
  
  
  
}  
  
protected boolean backtrack() {}  
  
  
  
  
  
  
  
  
  
}
```

15 Notification of constraints being violated

Override the **checkStateSpaceLimit** method and modify the **checkDepthLimit** method to incorporate **notifySearchConstraintHit (String)** to notify the following.

- "memory limit reached"
- "depth limit reached"


```
public boolean checkStateSpaceLimit() {
```

```
}
```

```
public boolean checkDepthLimit() {
```

```
}
```

16 Notification of property violations

Immediately after an invocation of the **forward** method of the **Search** class, an invocation of the **getCurrentError** method of the **Search** class returns **null** if and only if no property violation has been detected.

Modify the overridden **forward** method of the **DFSearch** class to include an invocation of the **notifyPropertyViolated** method.

```
protected boolean forward() {
```

```
}
```