Xây dựng phần mở rộng kiểm chứng thuộc tính logic thời gian (LTL) trong Java PathFinder

Học viên: Bùi Hoàng Khánh
Hướng dẫn: TS. Trương Anh Hoàng

Nội dung

- Giới thiệu
- Công cụ jpf-ltl
- Logic thời gian tuyến tính(LTL) trong jpf-ltl
- Thực thi tượng trưng với không gian trạng thái lớn
- Ví dụ
- Kết luận

Giới thiệu

- Phát triển phần mở rộng cho Java PathFinder
  - Kiểm chứng thuộc tính login thời gian
  - Thực thi tượng trưng trong không gian trạng thái lớn
- Các vấn đề cần giải quyết
  - Tích hợp thực thi tượng trưng vào cài đặt *DDFS*
  - Tổ chức lại mã nguồn và thêm một số mệnh đề nguyển tử còn thiếu của *jpf-ltl*
  - Cài đặt một số ví dụ cho chương trình

Công cụ jpf-ltl

- Được xây dựng trên JPF để kiểm chứng các chương trình Java thỏa mãn công thức LTL hay không
  - Đọc công thức LTL từ chú thích (annotation), dịch công thức này sang automat
  - Kiểm chứng chương trình hữu hạn với thực thi tượng trưng
  - Kiểm chứng chương trình vô hạn
- Các vấn đề cần làm thêm
  - Kiểm chứng chương trình có không gian trạng thái lớn với thực thi tượng trưng
  - Thêm các mệnh đề quan hệ nguyên tử còn thiếu

LTL trong jpf-ltl

- Cú pháp LTL trong *jpf-ltl*
  - Toán tử
    - *Λ*, &&, *V*, ||, ->, *!*, *[]*, <>, *U*
  - Mệnh đề nguyên tử
    - Biến: *packageName.ClassName.field*
    - Mệnh đề quan hệ: *(x+y)*z – 3.0 > u/5*
    - Phương thức: *package.Class.methodName(int, float)*
  - Ví dụ
    - @LTLSpec("[](<>Test.done() && <>Test.foo())")

LTL trong jpf-ltl

- Cài đặt một listener để đọc LTL từ chú thích (annotation)
  - gov.nasa.jpf.ltl.infinite.SymbolicLTLListener

Thực thi tượng trưng trong không gian vô hạn trạng thái

- DDFS tìm kiếm một vòng lặp có chứa trạng thái kết thúc
  - DFS 1: Duyệt qua các trạng thái đến trạng thái kết thúc E
  - Phase 2: Từ E, duyệt tiếp để tìm một trạng thái một lặp chứa trạng thái E
- DDFS được cài đặt làm chiến lược tìm kiếm trong JPF
  - gov.nasa.jpf.ltl.infinite.DDFSearch

Thực thi tượng trưng trong không gian vô hạn trạng thái

```
75    public void dfs1() {
76        recordVisit(dfs1Stack);
77        recordVisit(dfs1Table);
78
79        while (true) {
80            if (forward()) {
81                if (seenBefore(dfs1Table, 1)) {
82                    backtrack();
83                }
84                else {
85                    recordVisit(dfs1Stack);
86                    recordVisit(dfs1Table);
87
88                    if (isPropertyViolated()) {
89                        break;
90                    }
91                }
92            }
93            else {
94                if (inAcceptingState() && dfs2()) {
95                    prop.setViolated();
96                    break;
97                }
98                if (depth == 0) {
99                    terminate();
100                   break;
101               }
102
103               dfs1Stack.pop();
104               backtrack();
105           }
106       }
107   }
```

Thực thi tượng trưng trong không gian vô hạn trạng thái

- Kiểm tra trạng thái trong thực thi tượng trưng
  - Chia nhỏ trạng thái: kết hợp biểu thức nguyên tử với biểu thức đường đi
    - Giá trị biểu thức tượng trưng x > 0
    - Biểu thức đường đi x < 10
    - Ta có 2 trạng thái nhỏ là 0 < x < 10 và x >= 10
  - Kiểm tra xếp gộp (subsumption checking)
    - So sánh hình dạng Heap
    - So sánh các biểu thức ràng buộc trong các trạng thái bằng CVC3 solver

Example

```
17 @LTLSpec("[](foo())")
18 public class Raimondi {
19     @Symbolic("true")
20     static int y = 0;
21
22     public static void main(String[] args) {
23         test(0);
24     }
25
26     public static void test(int x) {
27         while (true) {
28             done();
29             if (y != 1) {
30                 foo();
31             }
32         }
33     }
34
35     public static void done() {}
36     public static void foo() {}
37 }
```

```
1 @using jpf-symbc
2
3 target=infinite.symbolic.Raimondi
4 finite=false
5 show_buchi=true
6
7 symbolic.method=infinite.symbolic.Raimondi.test(sym)
8 symbolic.dp=cvc3
9
10 classpath=${jpf-ltl}/build/examples
11
12 sourcepath=${jpf-ltl}/src/examples
13
14 search.class=gov.nasa.jpf.ltl.infinite.DDFSearch
```

Example

```
====================================================== error #1
gov.nasa.jpf.ltl.infinite.Property
Violated LTL property for infinite.symbolic.Raimondi:
        [](foo())

====================================================== snapshot #1
thread index=0,name=main,status=RUNNING,this=java.lang.Thread@0,target=null,priority=5,lockCount=0,suspendCount=0
  call stack:
        at infinite.symbolic.Raimondi.test(Raimondi.java:28)
        at infinite.symbolic.Raimondi.main(Raimondi.java:23)


====================================================== results
error #1: gov.nasa.jpf.ltl.infinite.Property "Violated LTL property for infinite.symbolic.Raimon..."

====================================================== statistics
elapsed time:       0:00:06
states:             new=9, visited=11, backtracked=2, end=0
search:             maxDepth=9, constraints=0
choice generators:  thread=18, data=3
heap:               gc=1, new=266, free=2
instructions:       2891
max memory:         81MB
loaded code:        classes=71, methods=888
```

Kết luận

- Hiểu được các vấn đề liên quan
  - LTL, thực thi tượng trưng, kiến trúc và mở rộng JPF
- Công cụ đã có thể kiểm chứng thuộc tính LTL của các chương trình Java có không gian trạng thái lớn với thực thi tượng trưng
  - Safety, liveness và fairness
- Công việc tiếp theo
  - Cập nhật jpf-ltl để tương thích với JPF mới nhất
  - Cài đặt thêm nhiều ví dụ
  - Áp dụng vào thực tế công việc

Xin cám ơn!