

CIS 842: Specification and Verification of Reactive Systems

Lecture Specifications: Progress Properties

Copyright 2001-2004, Matt Dwyer, John Hatcliff, and Robby. The syllabus and all lectures for this course are copyrighted materials and may not be used in other course settings outside of Kansas State University in their current form or modified form without the express written permission of one of the copyright holders. During this course, students are prohibited from selling notes to or being paid for taking notes by any person or commercial firm without the express written permission of one of the copyright holders.

Objectives

- To understand the essential difference between safety and liveness properties
- To understand the algorithm used to check for progress properties

Cyclic Behavior

- It is clear from looking at nearly all of our examples that systems can cycle indefinitely
 - e.g., dining philosophers
- This is a characteristic of reactive systems
- We will want to be able to characterize the fact that we expect the system to eventually perform some action

Safety Properties

Are fundamentally about not reaching certain undesirable states

Progress ...

- Intuitively means that the system eventually will *do* something
- From **every state** we should be able to make progress

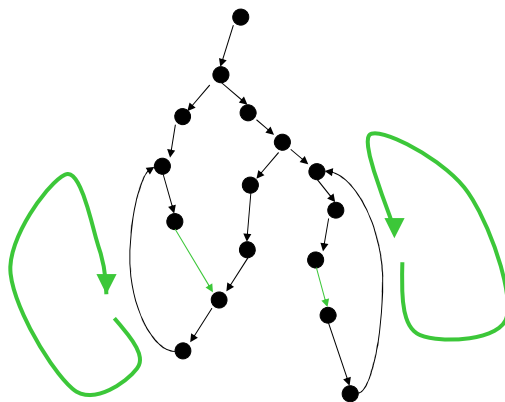
Specifying Progress

- A simple way to designate progress is to name labels of actions that should eventually be performed
- For example
 - {Philosopher1.eating, Philosopher2.eating}
- Property states
 - From all states in the system, eventually a (all) progress labeled action (s) will be executed

CIS 842: Spec Basics and Observables

5

Progress

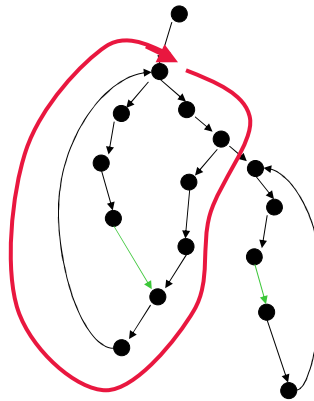


CIS 842: Spec Basics and Observables

6

Progress Violations

- A cyclic behavior on which no progress label occurs



CIS 842: Spec Basics and Observables

7

Checking for Progress

- Reachability works well for predecessors of progress actions
- Cycle detection works well for successors of progress actions
- Need to combine both checks in a single algorithm

CIS 842: Spec Basics and Observables

8

Recall : DFS Algorithm

```
1 seen := {s0}
2 pushStack(s0)
3 DFS(s0)

DFS(s)
4 workSet(s) := enabled(s)
5 while workSet(s) is not empty
6   let  $\alpha \in \text{workSet}(s)$ 
7   workSet(s) := workSet(s) \ { $\alpha$ }
8   s' :=  $\alpha(s)$ 
9   if s'  $\notin$  seen then
10    seen := seen  $\cup$  {s'}
11    pushStack(s')
12    DFS(s')
13    popStack()
end DFS
```

CIS 842: Spec Basics and Observables

9

Nested DFS Algorithm

```
1 seen := {s0}
2 pushStack(s0)
3 DFS(s0)

DFS(s)
4 workSet(s) := enabled(s)
5 while workSet(s) is not empty
6   let  $\alpha \in \text{workSet}(s)$ 
7   workSet(s) := workSet(s) \ { $\alpha$ }
8   s' :=  $\alpha(s)$ 
9   if s'  $\notin$  seen then
10    seen := seen  $\cup$  {s'}
11    DFS(s')
12    if  $\alpha$  is not progress then
13.1      NDFS(s', s')
13.2
14    popStack()
end DFS
```

Deleted stack maintaining
statements for brevity

CIS 842: Spec Basics and Observables

10

Nested DFS Algorithm

```
NDFS(s, seed)
14 workSet2(s) := enabled(s)
15 while workSet2(s) is not empty
16   let  $\alpha \in \text{workSet2}(s)$ 
17   if  $\alpha$  is progress
18     return
19   workSet2(s) := workSet2(s) \ { $\alpha$ }
20    $s' := \alpha(s)$ 
21   if  $s' = \text{seed}$  then
22     Non-progress cycle detected
23   if  $s' \notin \text{seen}'$  then
24      $\text{seen}' := \text{seen}' \cup \{s'\}$ 
25   NDFS( $s'$ , seed)
end DFS
```

similar to DFS (with
parate data structures)

For You To Do

- Take the dining philosophers example, with eating progress labels and apply the nested DFS algorithm to it
- Do you find an error?