

# CIS 842: Specification and Verification of Reactive Systems

## Lecture INTRO-Depth-Bounded: Depth-Bounded Depth-first Search

Copyright 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

- Understand the principle of depth-bounded depth-first search
- Understand how to modify the core DFS algorithm to implement a depth-bounded search

## Outline

- Using Bogor's output to determine the max stack size (number of steps in longest execution path) encountered during the depth-first search
- Setting Bogor's depth bound
- Rationale for using depth-bounded search
- The consequences (unsoundness) of performing bounded searches
- Bogor options for finding minimal length counter-examples
- The core DFS algorithm extended to included depth-bounded search

## Bogor Output

Running a model-check of SumToN with  $N = 5$

[Time: 3315ms, Depth: 395] Error found: Assertion failed

Total memory before search: 5,864,216 bytes (5.59 Mb)

Total memory after search: 6,676,048 bytes (6.37 Mb)

Total search time: 3325 ms

State count: 12127

Matched state count: 16393

Max depth: 1922

*...deepest stack depth reached during search*

*...depth in computation tree (i.e., transition stack) where assertion violation was found (i.e., number of steps in error trace)*

## Error Trace Length

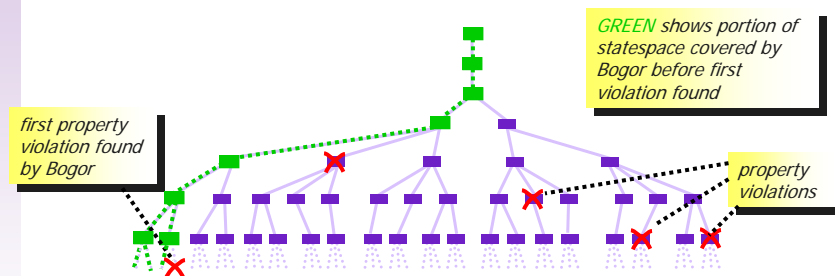
- Model-check SumToN with  $N = 5$
- From Bogor's output, we can see that it has found a execution trace that violates the assertion and that the trace is 395 steps long.
  - Having to reason about how the assertion can be violated along a trace of 395 steps is quite painful!
  - You have previously discovered a much shorter violating trace using Bogor's simulation mode.
  - Does this mean that the Bogor analyzer is not very useful?
    - Not at all!!
- We will see in a little bit how to tell Bogor to search for shorter violating traces (as well as minimal length violating traces) .

CIS 842: INTRO: Depth-bounded DFS

5

## Error Trace Length

- In general, a system may have many different traces that lead to the same property violation.



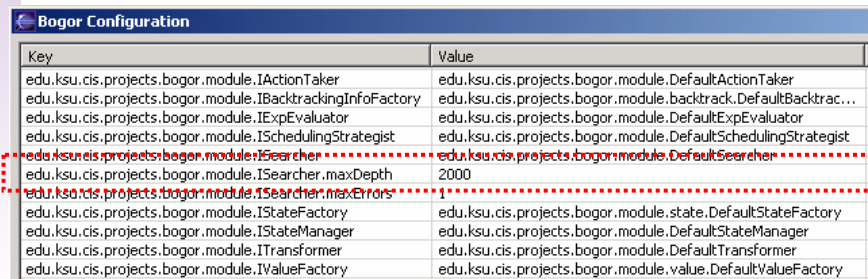
- Because Bogor does a depth-first search (instead of a bread-first search), the first violating trace that it finds is usually not of minimal length.

CIS 842: INTRO: Depth-bounded DFS

6

## Setting Bogor's Depth Bound

- Users can set a bound on the depth of Bogor's search (i.e., entries in Bogor's depth-first stack)



Key	Value
edu.ksu.cis.projects.bogor.module.IActionTaker	edu.ksu.cis.projects.bogor.module.DefaultActionTaker
edu.ksu.cis.projects.bogor.module.IBacktrackingInfoFactory	edu.ksu.cis.projects.bogor.module.backtrack.DefaultBacktrac...
edu.ksu.cis.projects.bogor.module.IExpEvaluator	edu.ksu.cis.projects.bogor.module.DefaultExpEvaluator
edu.ksu.cis.projects.bogor.module.ISchedulingStrategist	edu.ksu.cis.projects.bogor.module.DefaultSchedulingStrategist
edu.ksu.cis.projects.bogor.module.ISearcher	edu.ksu.cis.projects.bogor.module.DefaultSearcher
edu.ksu.cis.projects.bogor.module.ISearcher.maxDepth	2000
edu.ksu.cis.projects.bogor.module.ISearcher.maxErrors	1
edu.ksu.cis.projects.bogor.module.IStateFactory	edu.ksu.cis.projects.bogor.module.state.DefaultStateFactory
edu.ksu.cis.projects.bogor.module.IStateManager	edu.ksu.cis.projects.bogor.module.DefaultStateManager
edu.ksu.cis.projects.bogor.module.ITransformer	edu.ksu.cis.projects.bogor.module.DefaultTransformer
edu.ksu.cis.projects.bogor.module.IValueFactory	edu.ksu.cis.projects.bogor.module.value.DefaultValueFactory

Choose the "Configure Bogor" option, then Add/Edit to set the value for the *ISearcher.maxDepth* attribute.

CIS 842: INTRO: Depth-bounded DFS

7

## Setting Bogor's Depth Bound

- This is often useful...
  - ...after a counterexample has been found and you want to see if a shorter one exists.
    - look at Bogor's output to see the size, then rerun Bogor with an appropriate depth bound (i.e., one smaller than the size of the counter-example).
  - ...before a counterexample has been found and Bogor is taking too long or is running out of memory.

CIS 842: INTRO: Depth-bounded DFS

8

# Setting Bogor's Depth Bound

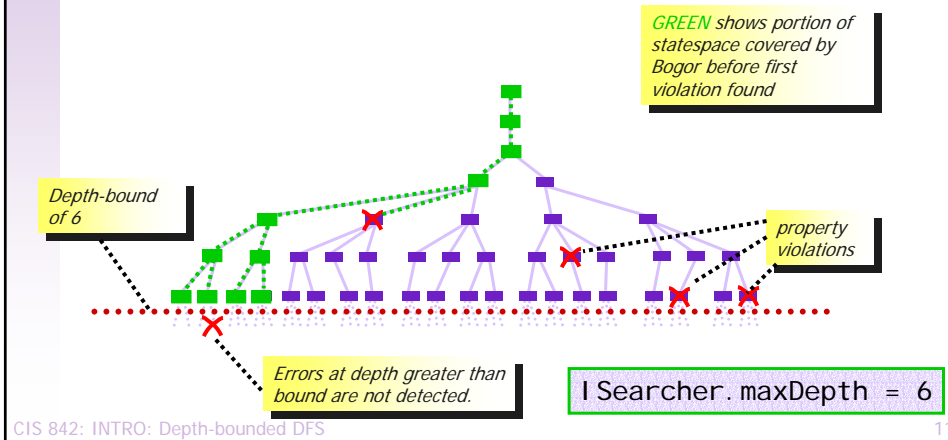
- Be careful!
  - when you bound Bogor's search, Bogor will not be exploring part of the system state-space, and the omitted part may contain property violations that you want to detect.
  - If Bogor tells you that there are no violations, but you have bounded the search, you *cannot assume that the system has no violations*. You only know that Bogor has found no violations in the part of the state-space that it searched.
  - Bogor displays "Max depth reached!!!" to let you know that the depth bound prevented it from searching the complete state-space.

## Checking SumToN with N = 5

System	Transitions	States	Matched	Max. Depth	Errors	Time
SumToN	556	278	279	15	1	0:0:0
SumToN	276	135	142	11	0	0:0:0
SumToN	556	278	279	15	1	0:0:0
SumToN	297	156	142	12	1	0:0:0
SumToN	26	27	0	26	1	0:0:0
SumToN	26	27	0	26	1	0:0:0
SumToN	26	27	0	26	1	0:0:0
SumToN	26	27	0	26	1	0:0:0
SumToN	26	27	0	26	1	0:0:0
SumToN	28519	12127	16393	1922	1	0:0:3
SumToN	61	36	26	6	0	0:0:0
SumToN	276	135	142	11	0	0:0:0
SumToN	297	156	142	12	1	0:0:0

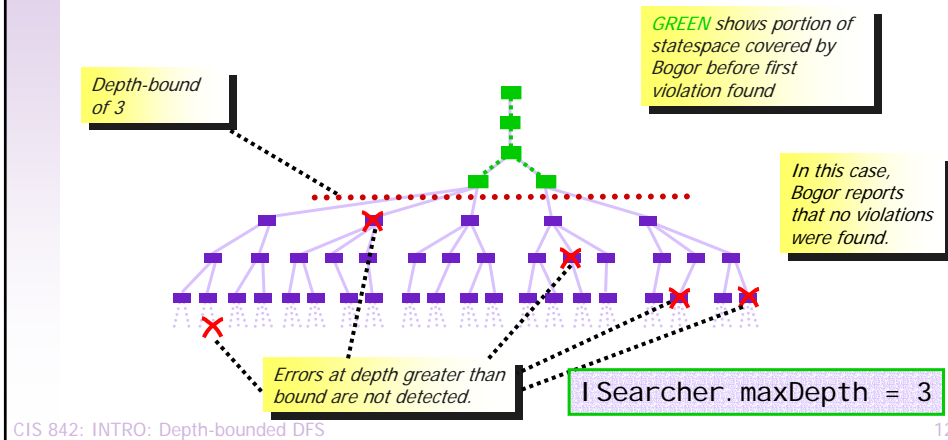
# Bounded Depth-first Search

- When analyzing a system and given a depth bound as a command-line argument, Bogor will backtrack when that depth is reached.



# Bounded Depth-first Search

- When analyzing a system and given a depth bound as a command-line argument, Bogor will backtrack when that depth is reached.



## For You To Do...

- Pause the lecture...
- Edit `SumToN.bi r` and change the assertion to `x != 7`.
- Use Bogor to find an error trace of minimal length.
  - start with a depth bound that allows an error
  - successively choose smaller versions of the bound until Bogor reports no error
  - determine a bound B such that running SPIN with bound B-1 reveals no errors, but running with B reveals an error
- How does this error trace compare to the one (i.e., size and state vectors encountered) to the error trace that you discovered earlier using Bogor's guided simulation mode?
- Note that there may be multiple minimal length error traces.

## Assessment

- Minimal length error traces can be found with the `| Search.maxDepth` option for Bogor.
- This is somewhat tedious for the user.
- SPIN provides two other options to find shorter traces...
  - `pan.exe -i`
    - finds a minimal length path by successively rerunning with bound set to length-of-current-violating-trace - 1 (can be costly!)
  - `pan.exe -l`
    - similar to the option above but faster (a form of binary search is used), but approximate (sometimes minimal error trace is not found)
- We will eventually incorporate functionality similar to this in Bogor (it's not difficult!)

# Summary

- Bogor, like most other model-checkers, provides for a depth-bounded search
- A depth-bounded search can be useful when...
  - ...after a counterexample has been found and you want to see if a shorter one exists.
    - look at Bogor's output to see the size, then rerun Bogor with an appropriate depth bound (i.e., one smaller than the size of the counter-example).
  - ...before a counterexample has been found and Bogor is taking too long or is running out of memory.
- However, a depth-bounded search is technically unsound since it gives rise to an under-approximation of the system's behavior
  - In other words, an error may actually lie in an area that is outside of the area searched in the model-check