南京大學
NANJING UNIVERSITY

Steering Symbolic Execution to Less Traveled Paths

You Li

Joint with Zhendong Su, Linzhang Wang, Xuandong Li

SEG Nanjing University

Background and Motivations

- Testing is important, but can be ineffective
  - Software is complex with large or infinite state space
  - Manual testing is tedious and ad hoc
  - Random testing is not systematic

- Symbolic execution is promising
  - Systematically explores a program
  - Generates test cases with high coverage

Symbolic Execution

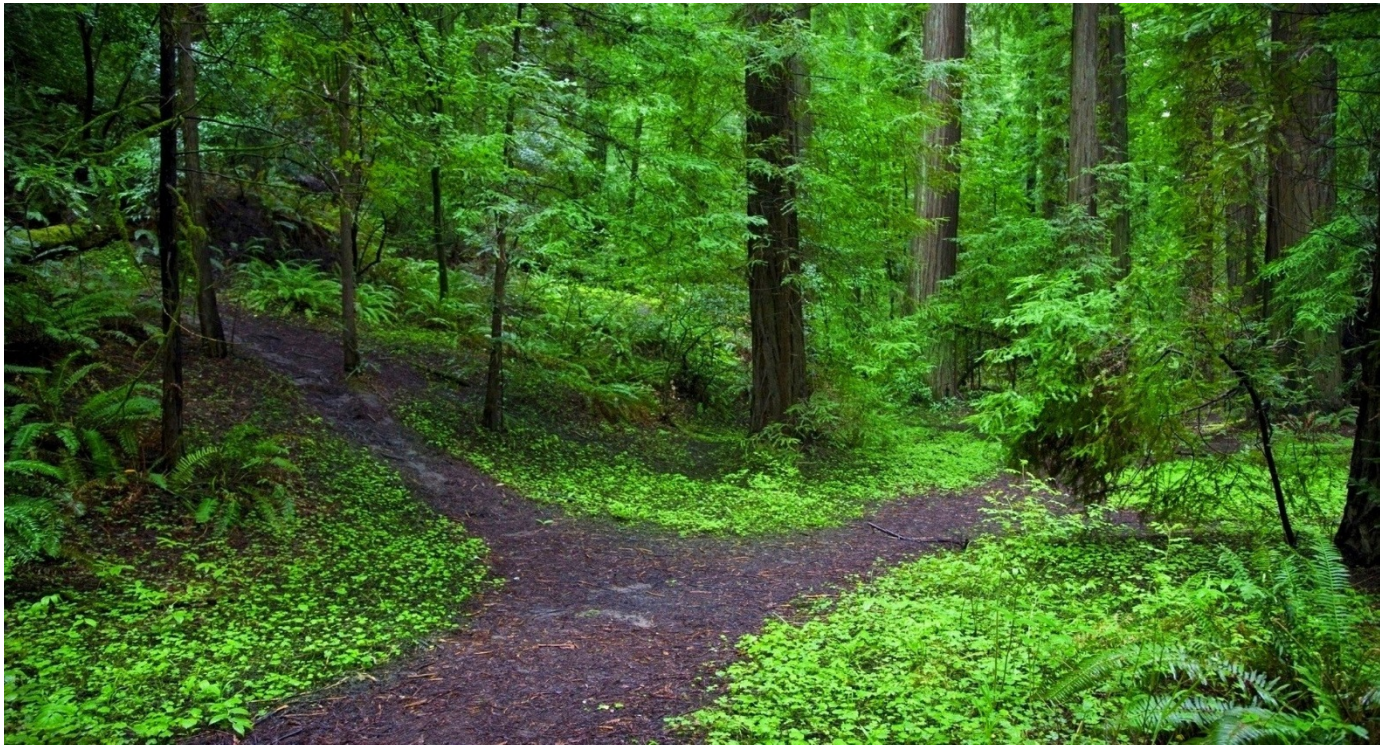
- Uses symbolic values for inputs to explore a program
- Forks at branch conditions
- Follows both directions by updating path constraints
- Solves path constraints to generate test cases

Main Challenges

- Complex constraints
- Path explosion

Goal: Guide Symbolic Execution to Profitable Paths

Key issue: How to guide toward profitable paths?



Less Traveled Paths

- Benefits
  - Cover the program better
  - Locate more bugs

- Difficulties
  - Define "footprints"
  - Use "footprints" to guide path exploration

Subpath-Guided Path Exploration

- How to define “footprints”?
  - *Length-n* Subpath Program Spectra

- How to use "footprints" to guide path exploration?
  - Subpath-Guided Search (SGS)

Program Spectra

- Program profiling
  - Counting different program execution events
- Profiling of different events provides various program spectra
  - Branch Hit Spectra
  - Branch Count Spectra
  - Complete Path Spectra
  - Path Spectra
  - Path Count Spectra

Length-n Subpath Program Spectra

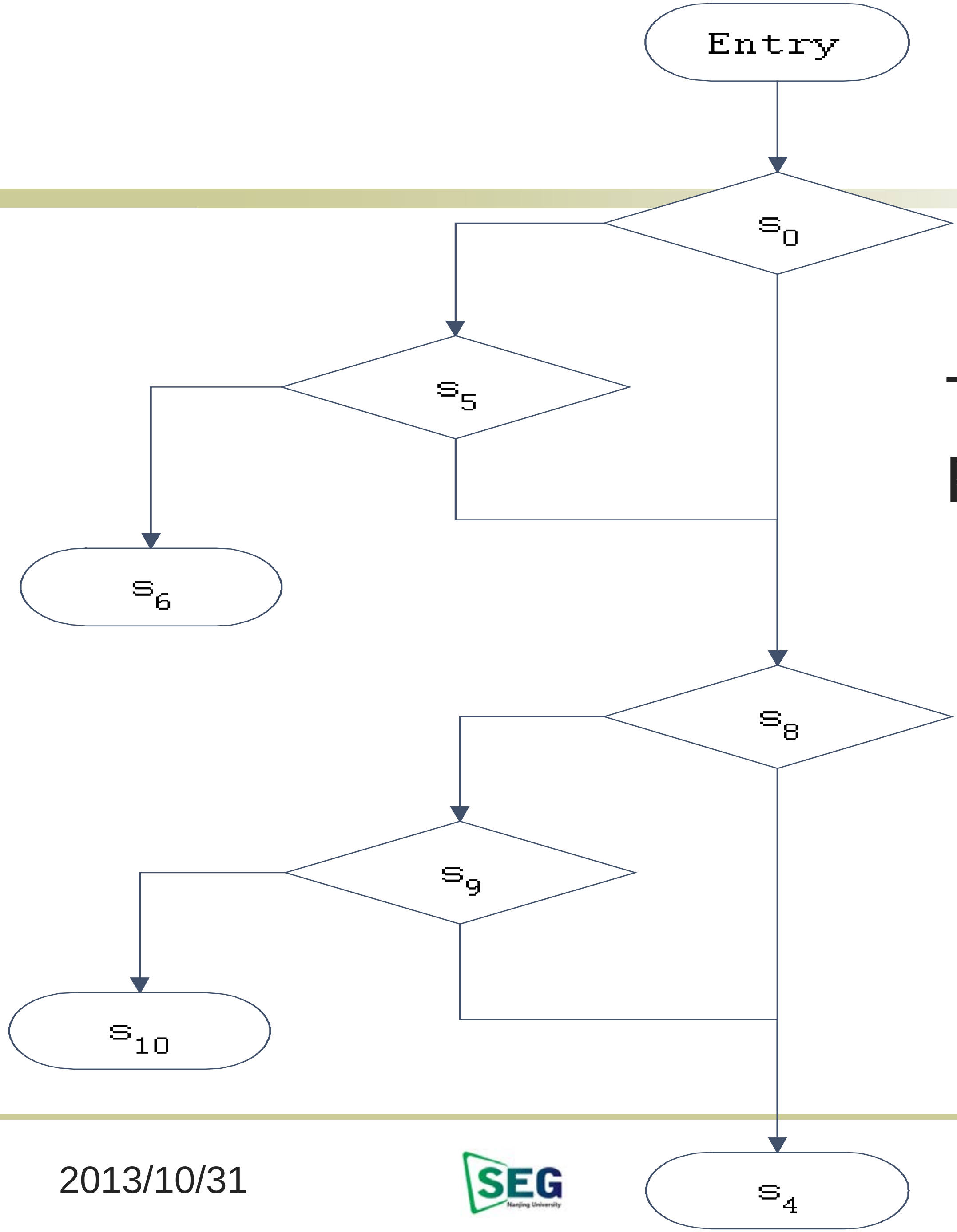
- Each subpath has *n* branches
- Contiguous sub-sequences of execution paths
- Varying n leads to a spectrum of modeling precision
- Fills the gap between branch coverage & complete path coverage

Subpath Guided Search (SGS)

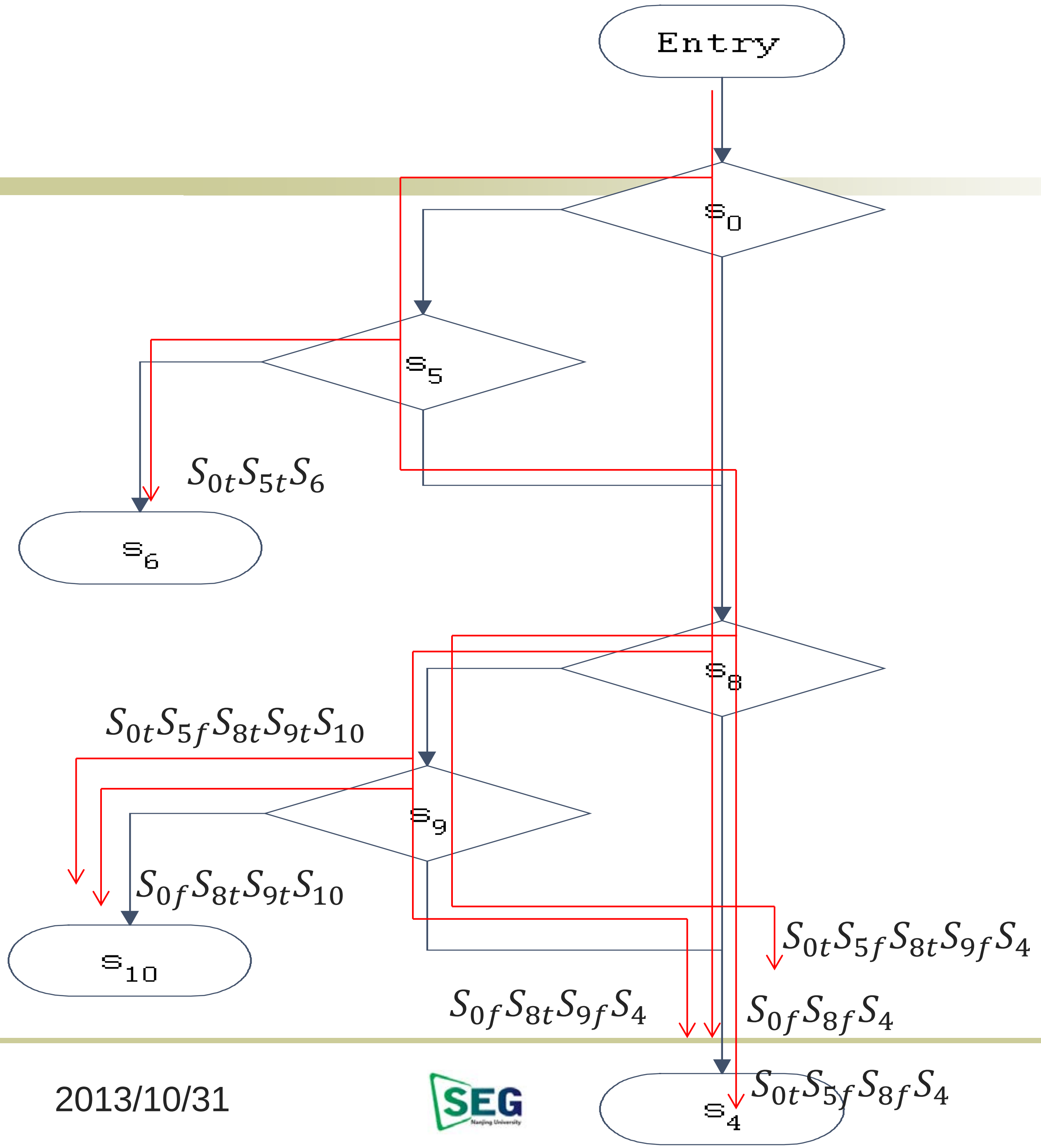
- Maintain a structure $e =< \pi_n, f >$
  - π_n is a length n subpath
  - f is the frequency of π_n
- For each execution, track the most recent length-n path segment
- Pick a pending execution with the lowest f to explore next
  - Break ties randomly

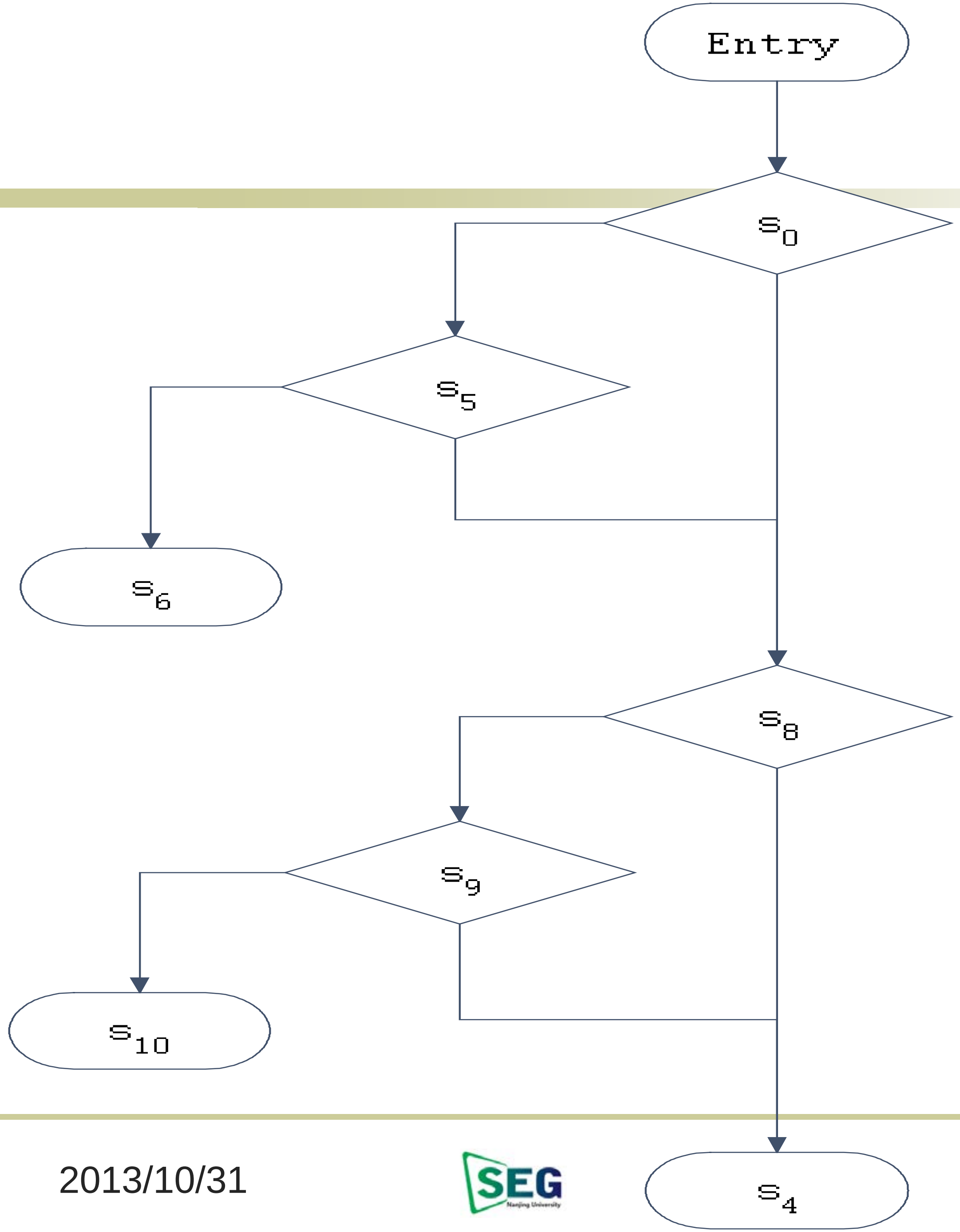
Example

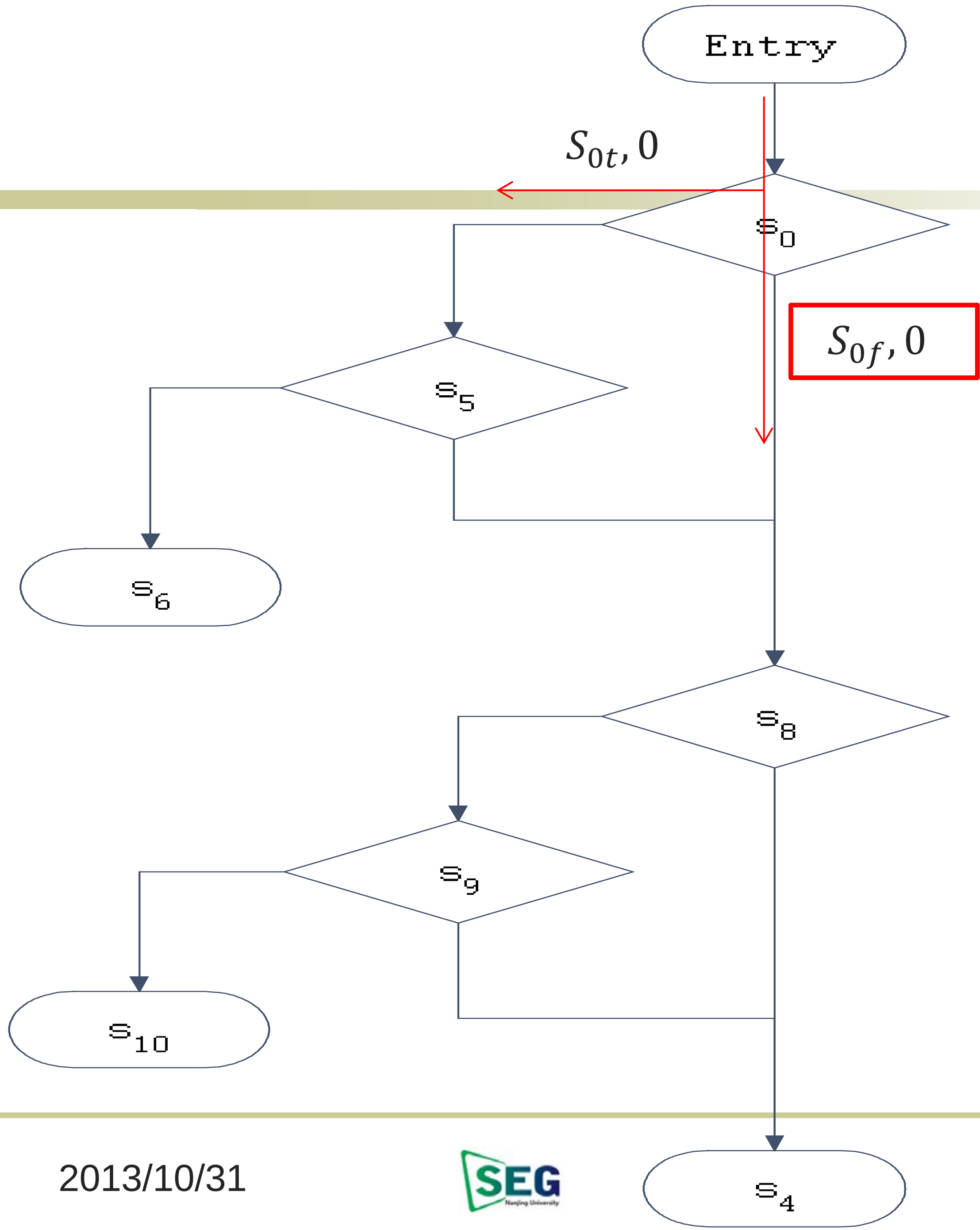
```
     main (x, y) {
s0:      if  (x > y)
s1:          x = f(x);
         else
s2:          ;
s3:      g (x, y);
s4:      return;
     }

     int f (a) {
s5:      if  (a > 0)
s6:          ABORT;
         else
s7:          return -a;
     }

     g (a, b) {
s8:      if  (a == 0)
s9:          if  (b == 0)
s10:             ABORT;
             else
s11:             ;
         else
s12:         print a/b;
s13:     return;
     }
```



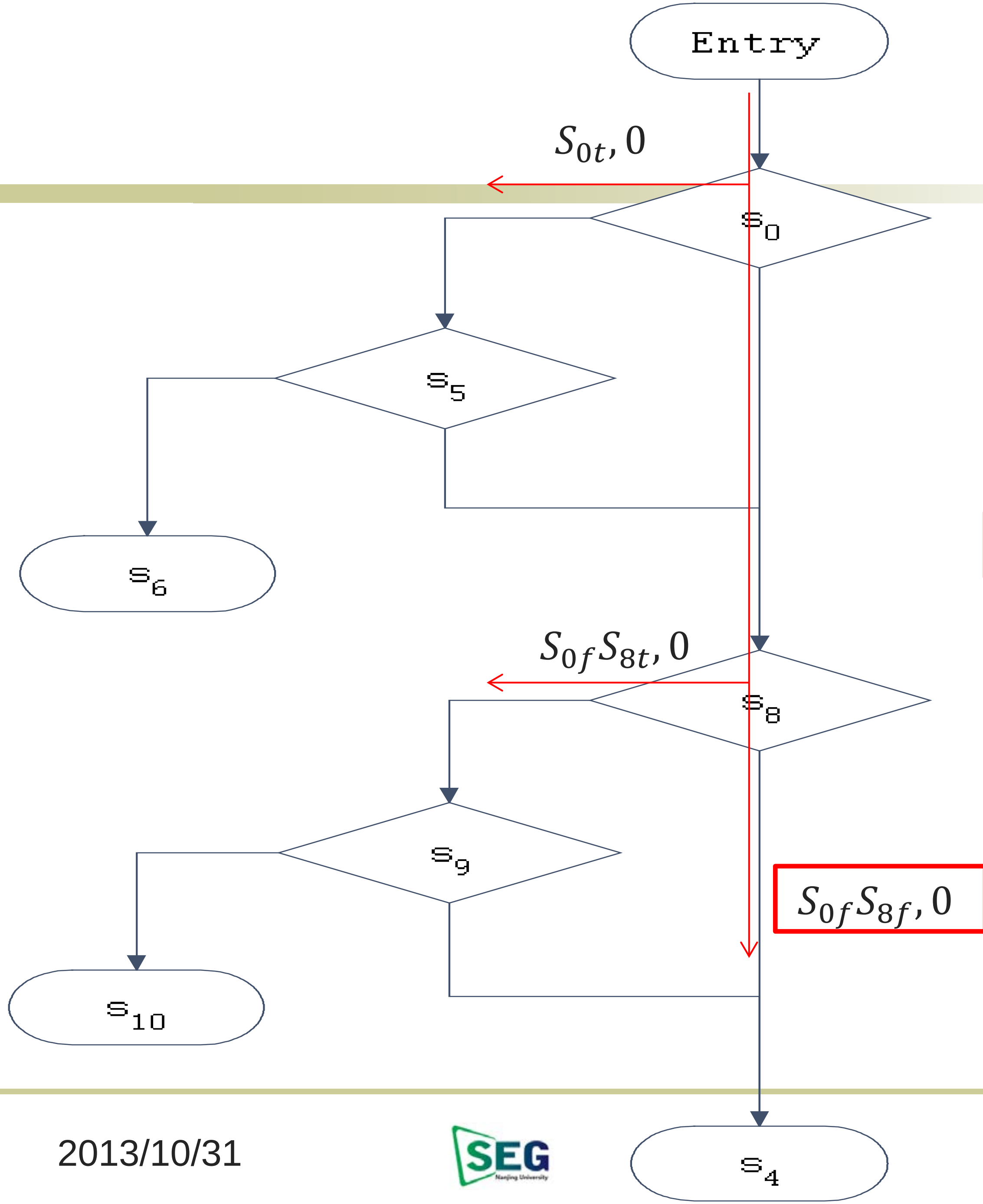
Total Number of Paths: 7


Entry
S_0
S_5
$S_{0t}S_{5t}S_6$
S_6
S_8
$S_{0t}S_{5f}S_{8t}S_{9t}S_{10}$
S_9
$S_{0f}S_{8t}S_{9t}S_{10}$
S_{10}
$S_{0t}S_{5f}S_{8t}S_{9f}S_4$
$S_{0f}S_{8t}S_{9f}S_4$
$S_{0f}S_{8f}S_4$
$S_{0t}S_{5f}S_{8f}S_4$
S_4

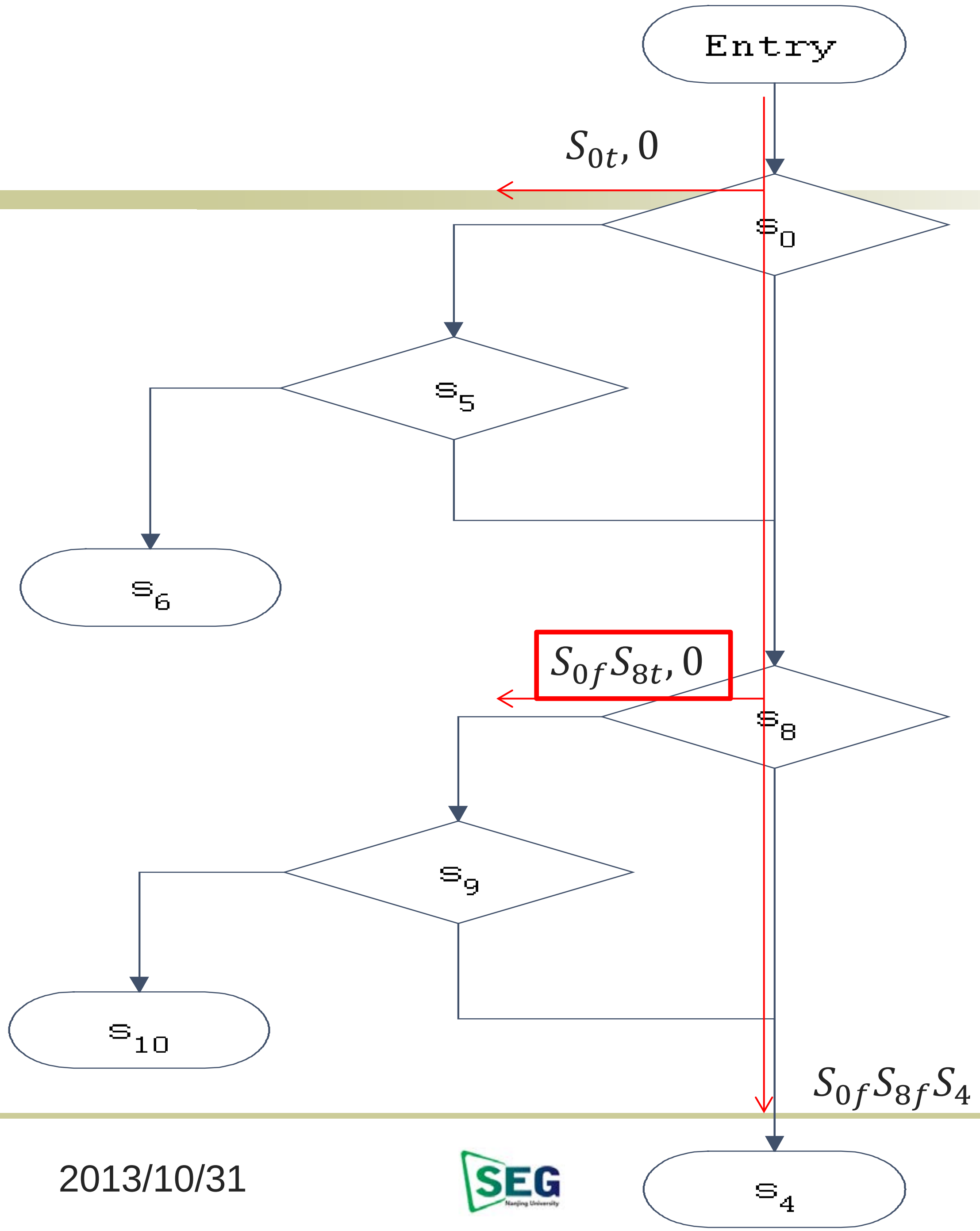

Entry
S_0
S_5
S_6
S_8
S_9
S_{10}
S_4



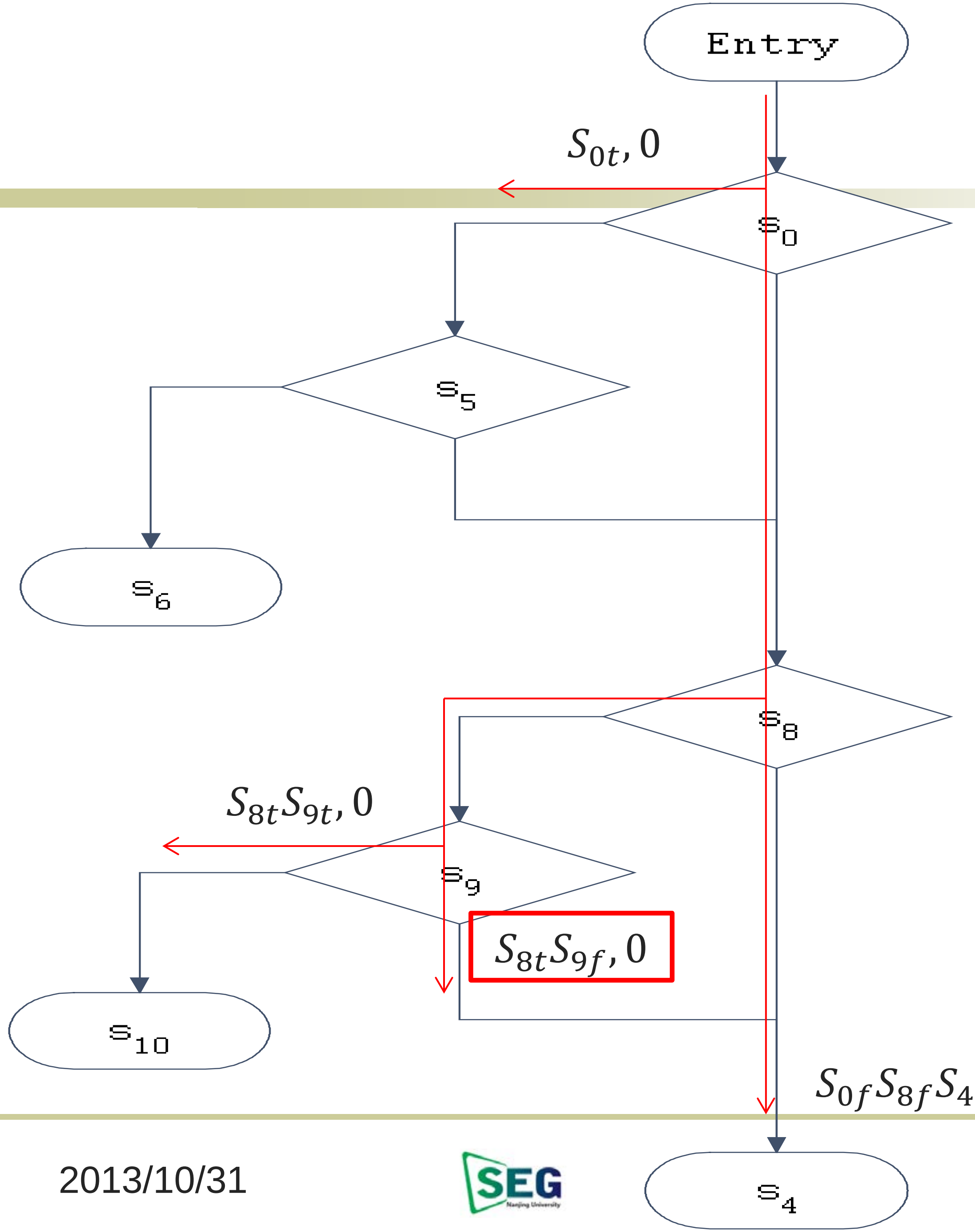
Sub-path Frequency
$S_{0t}, 0$ *
$S_{0f}, 0$ *



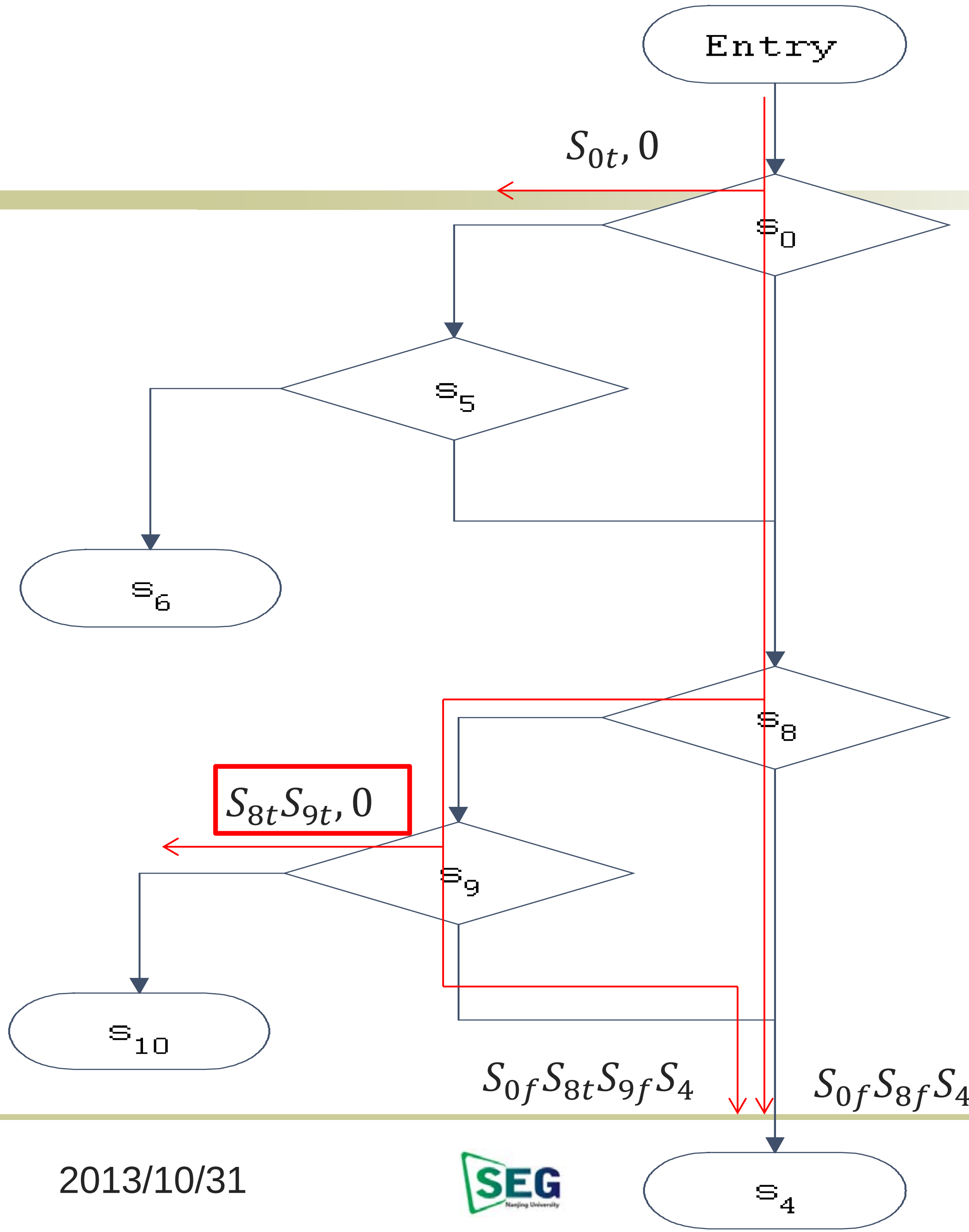
Sub-path Frequency
$S_{0t}, 0$ *
$S_{0f}, 1$
$S_{0f}S_{8t}, 0$ *
$\boxed{S_{0f}S_{8f}, 0}$ *



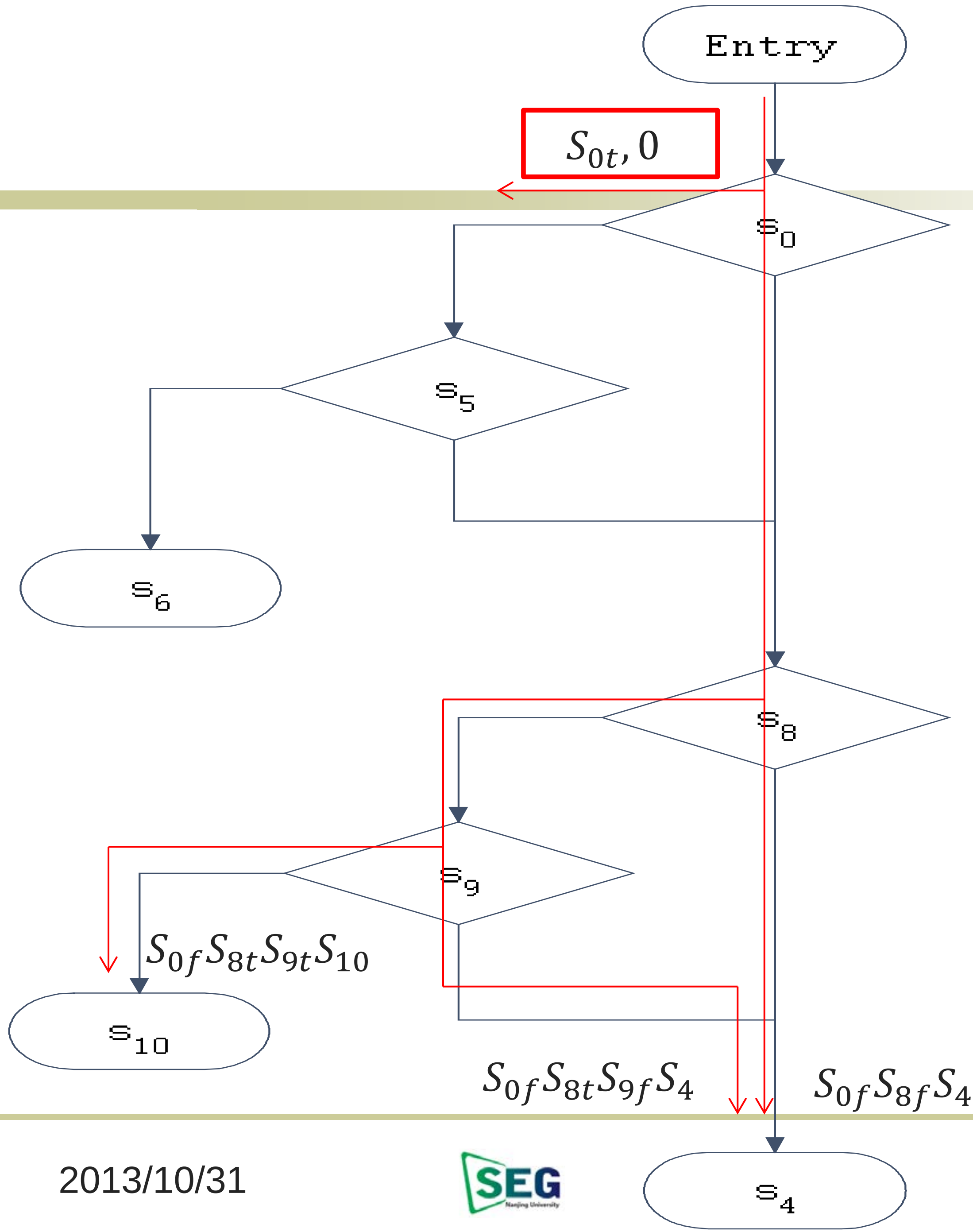
Sub-path Frequency
$S_{0t}, 0$ *
$S_{0f}, 1$
$S_{0f}S_{8t}, 0$ *
$S_{0f}S_{8f}, 1$



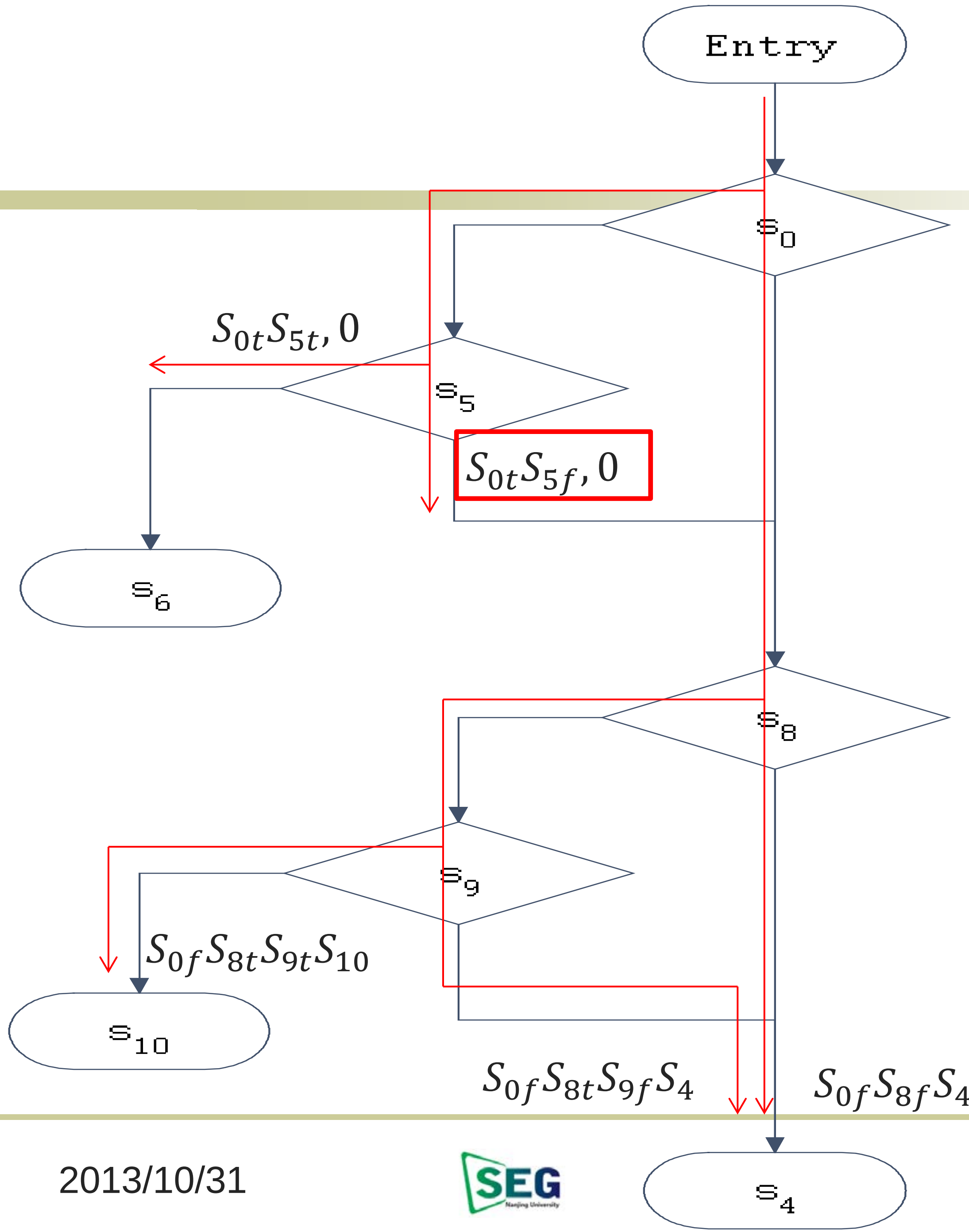
Sub-path	Frequency
S_{0t}	0 *
S_{0f}	1
$S_{0f}S_{8t}$	1
$S_{0f}S_{8f}$	1
$S_{8t}S_{9t}$	0 *
$S_{8t}S_{9f}$	0 *



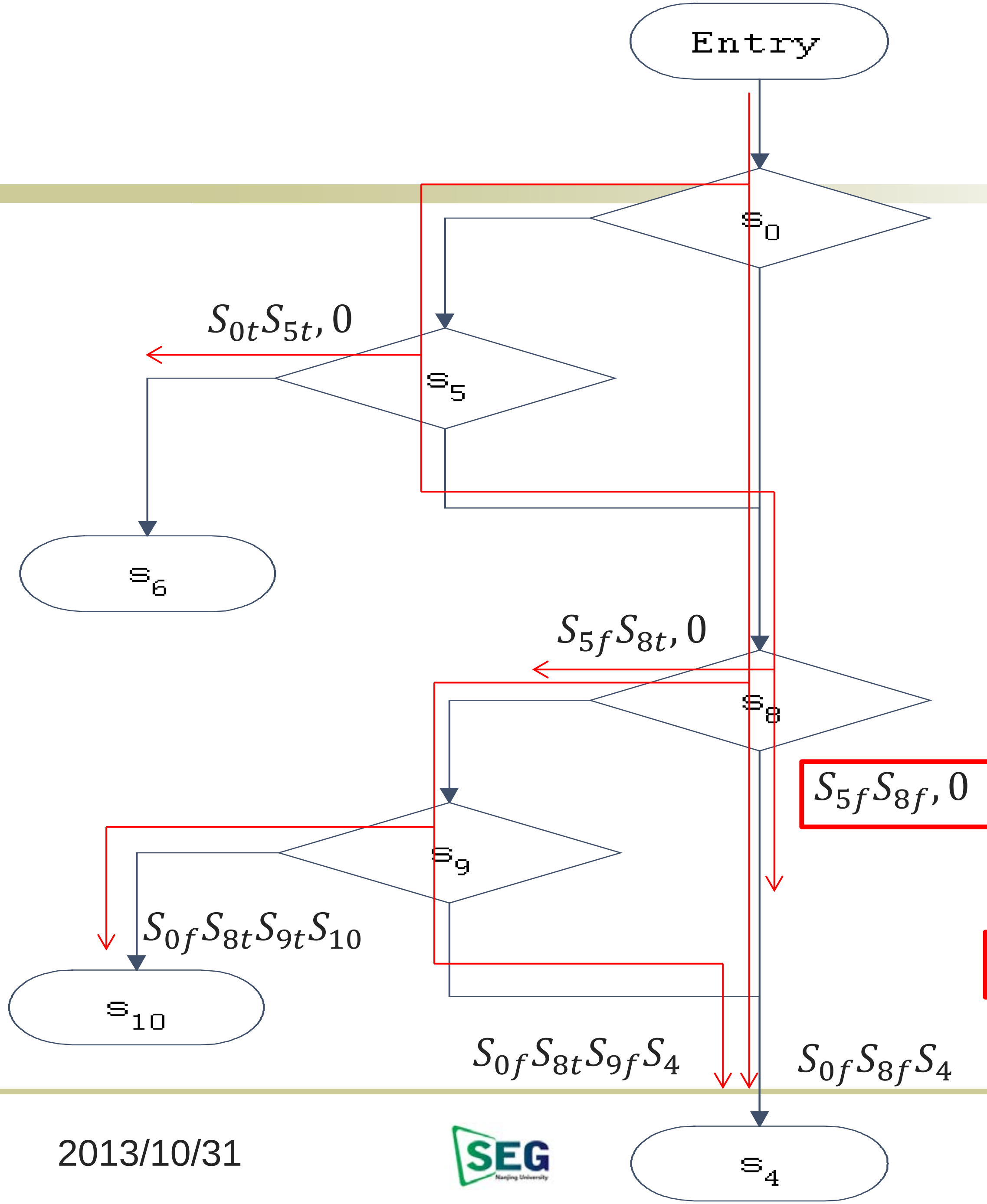
Sub-path Frequency
$S_{0t}, 0$ *
$S_{0f}, 1$
$S_{0f}S_{8t}, 1$
$S_{0f}S_{8f}, 1$
$S_{8t}S_{9t}, 0$ *
$S_{8t}S_{9f}, 1$



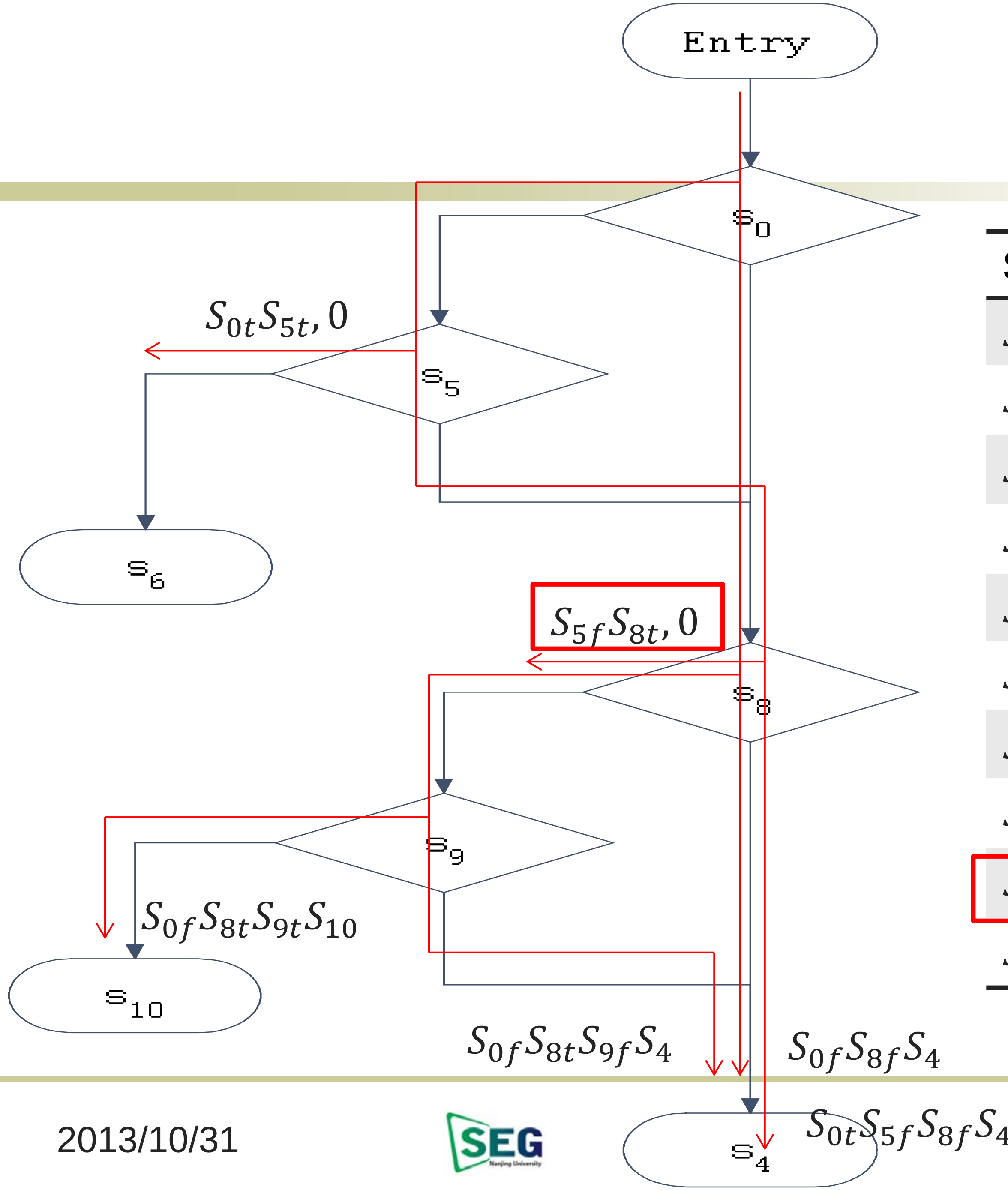
Sub-path Frequency
$S_{0t}, 0$ *
$S_{0f}, 1$
$S_{0f}S_{8t}, 1$
$S_{0f}S_{8f}, 1$
$S_{8t}S_{9t}, 1$
$S_{8t}S_{9f}, 1$



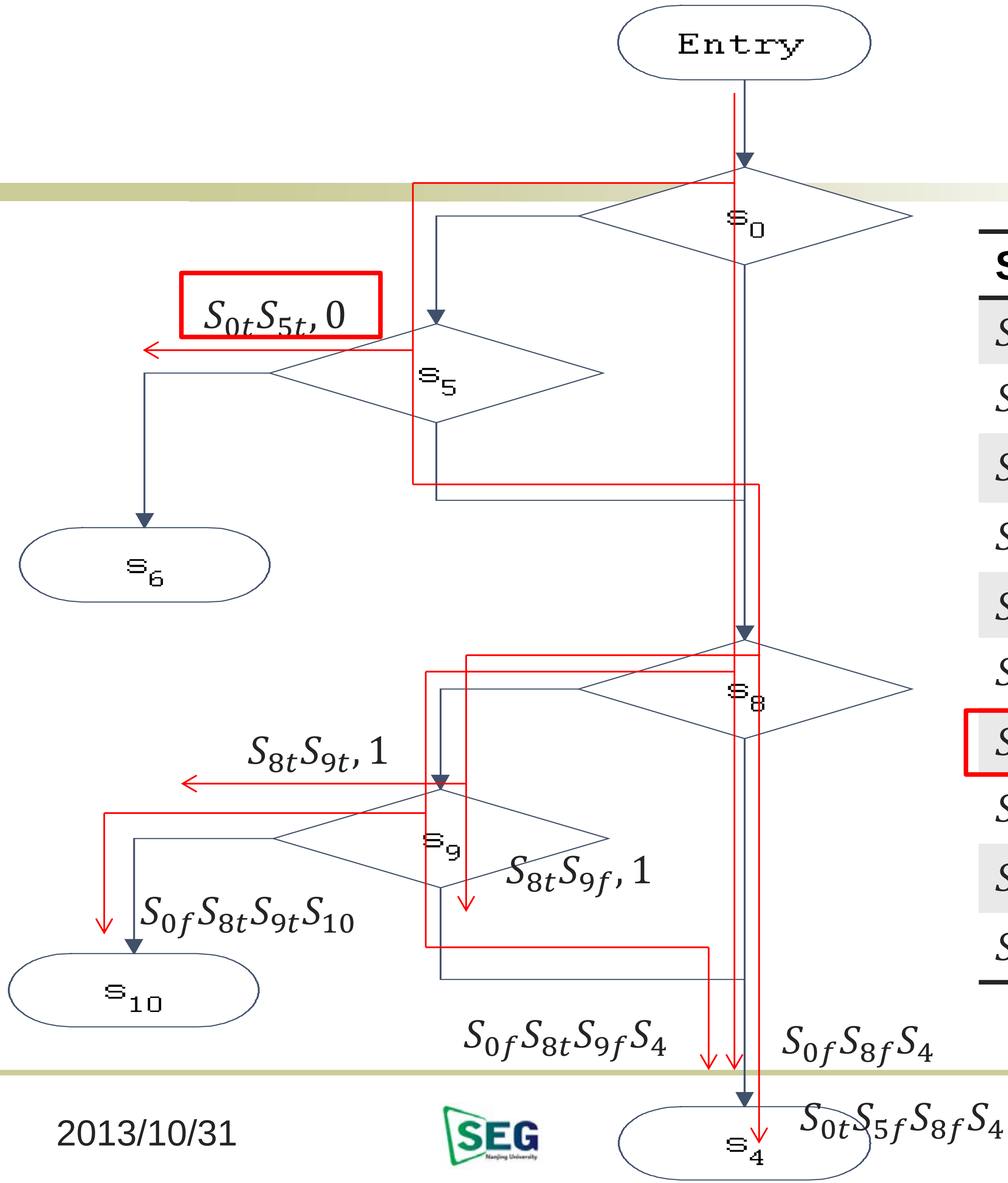
Sub-path Frequency
$S_{0t}, 1$
$S_{0f}, 1$
$S_{0f}S_{8t}, 1$
$S_{0f}S_{8f}, 1$
$S_{8t}S_{9t}, 1$
$S_{8t}S_{9f}, 1$
$S_{0t}S_{5t}, 0$ *
$\boxed{S_{0t}S_{5f}, 0 \ *}$



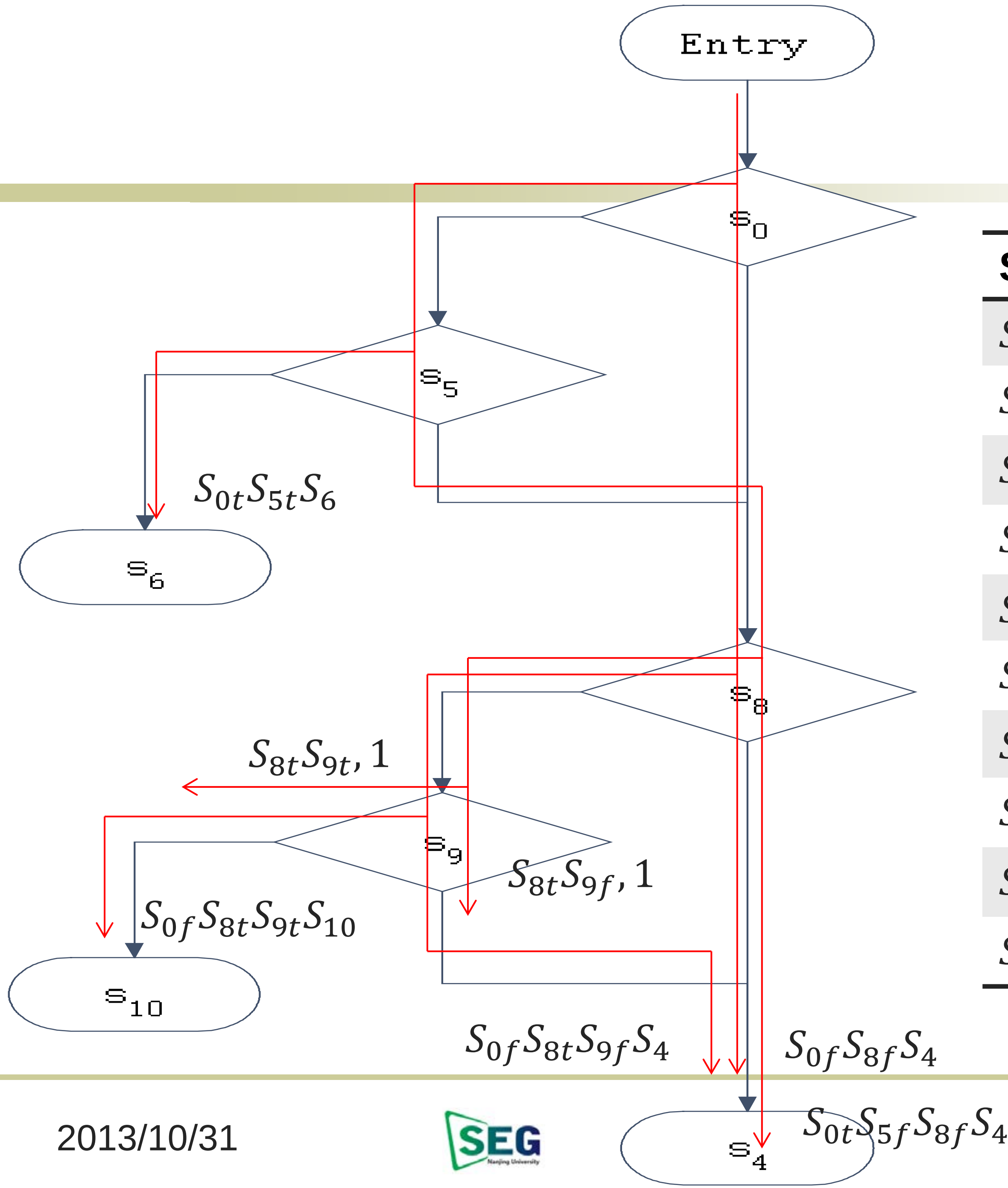
Sub-path Frequency
$S_{0t}, 1$
$S_{0f}, 1$
$S_{0f}S_{8t}, 1$
$S_{0f}S_{8f}, 1$
$S_{8t}S_{9t}, 1$
$S_{8t}S_{9f}, 1$
$S_{0t}S_{5t}, 0$ *
$S_{0t}S_{5f}, 1$
$S_{5f}S_{8t}, 0$ *
$\boxed{S_{5f}S_{8f}, 0 \text{ *}}$



Sub-path	Frequency
S_{0t}	1
S_{0f}	1
$S_{0f}S_{8t}$	1
$S_{0f}S_{8f}$	1
$S_{8t}S_{9t}$	1
$S_{8t}S_{9f}$	1
$S_{0t}S_{5t}$	0 *
$S_{0t}S_{5f}$	1
$\boxed{S_{5f}S_{8t}}$	$\boxed{0\ *}$
$S_{5f}S_{8f}$	1



Sub-path Frequency
$S_{0t}, 1$
$S_{0f}, 1$
$S_{0f}S_{8t}, 1$
$S_{0f}S_{8f}, 1$
$S_{8t}S_{9t}, 1$ *
$S_{8t}S_{9f}, 1$ *
$S_{0t}S_{5t}, 0$ *
$S_{0t}S_{5f}, 1$
$S_{5f}S_{8t}, 1$
$S_{5f}S_{8f}, 1$



Sub-path Frequency
$S_{0t}, 1$
$S_{0f}, 1$
$S_{0f}S_{8t}, 1$
$S_{0f}S_{8f}, 1$
$S_{8t}S_{9t}, 1$
$S_{8t}S_{9f}, 1$
$S_{0t}S_{5t}, 1$
$S_{0t}S_{5f}, 1$
$S_{5f}S_{8t}, 1$
$S_{5f}S_{8f}, 1$

Evaluation: Research Questions

- What impact do different choices of *n* have?
- Can they be effectively combined?
- How does our strategy compare to existing strategies?

Evaluation Setup

- Implement SGS in KLEE
- Evaluation subjects: GNU core utilities
- Evaluated search strategies
  - Length-n SGS with varying n (n = 1, 2, 4, 8)
  - Existing strategies implemented in KLEE
- Evaluation metrics
  - How well a program is covered?
  - How effective in locating bugs?

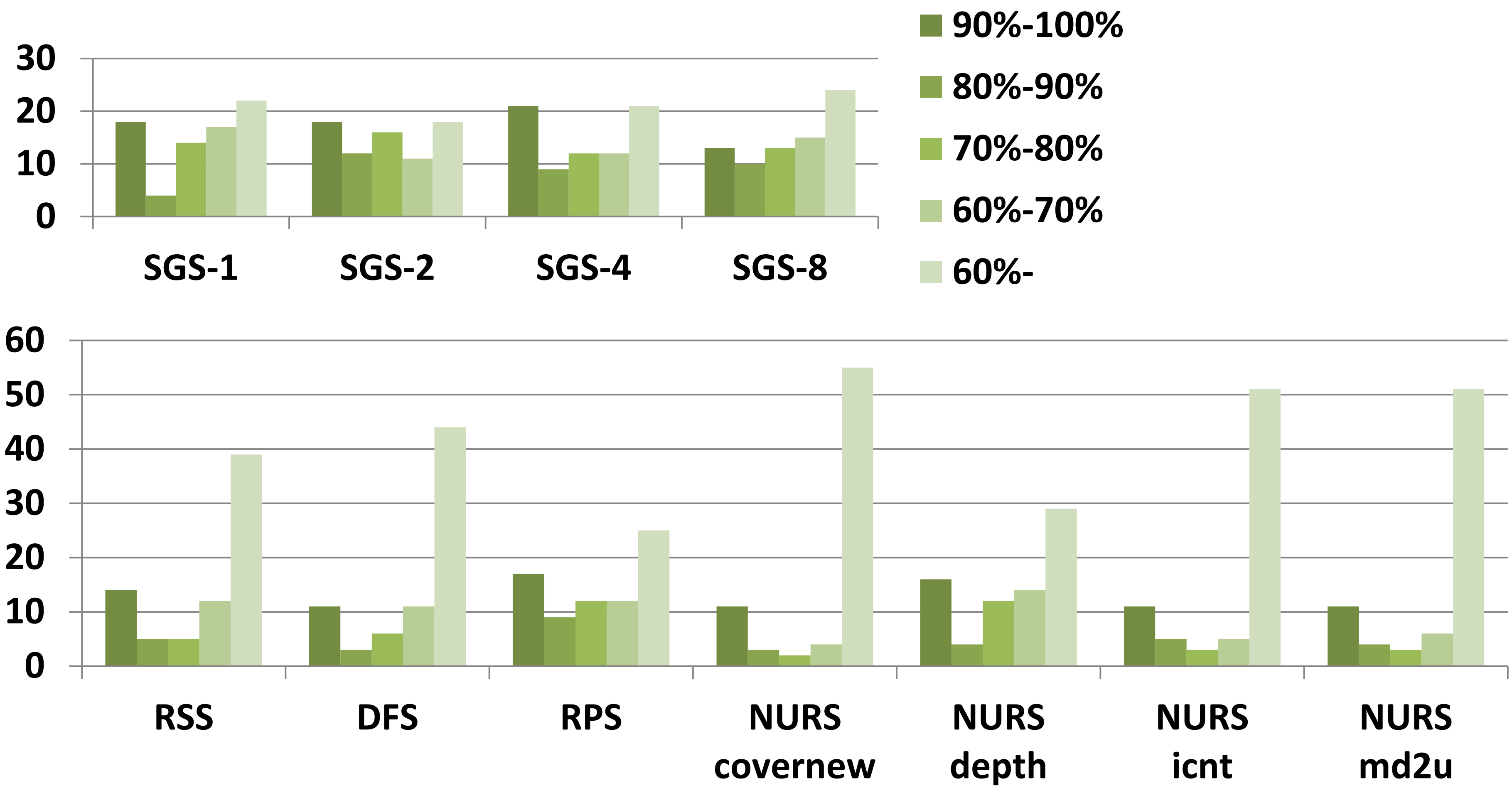
KLEE Strategies

- DFS
- Random State
- Random Path
- Non-Uniform Random Selection
  - covnew
  - depth
  - icnt
  - md2u

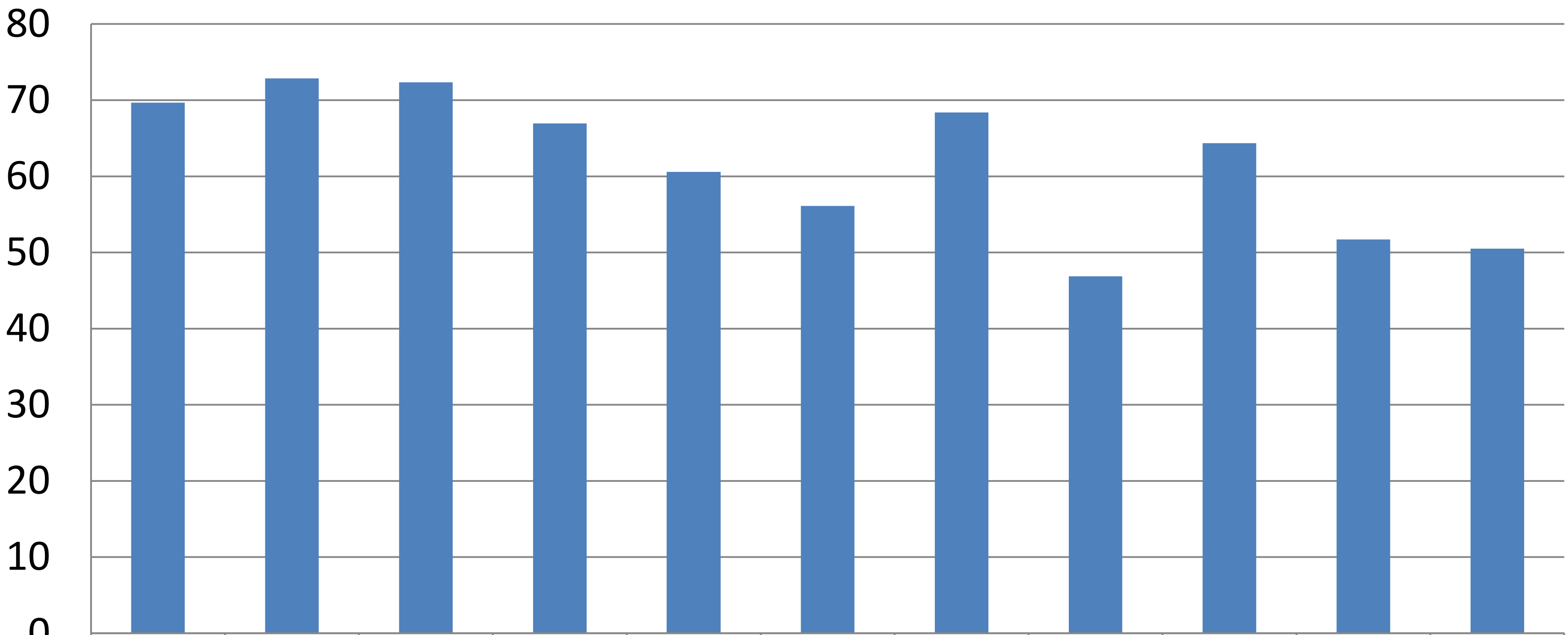
Program Coverage

- 75 programs (2K - 10K LOC in size)
- Run each strategy for 1 hour
- Output test cases exploring new statements or triggering errors
- Re-execute test cases to measure statement coverage

Coverage Distribution

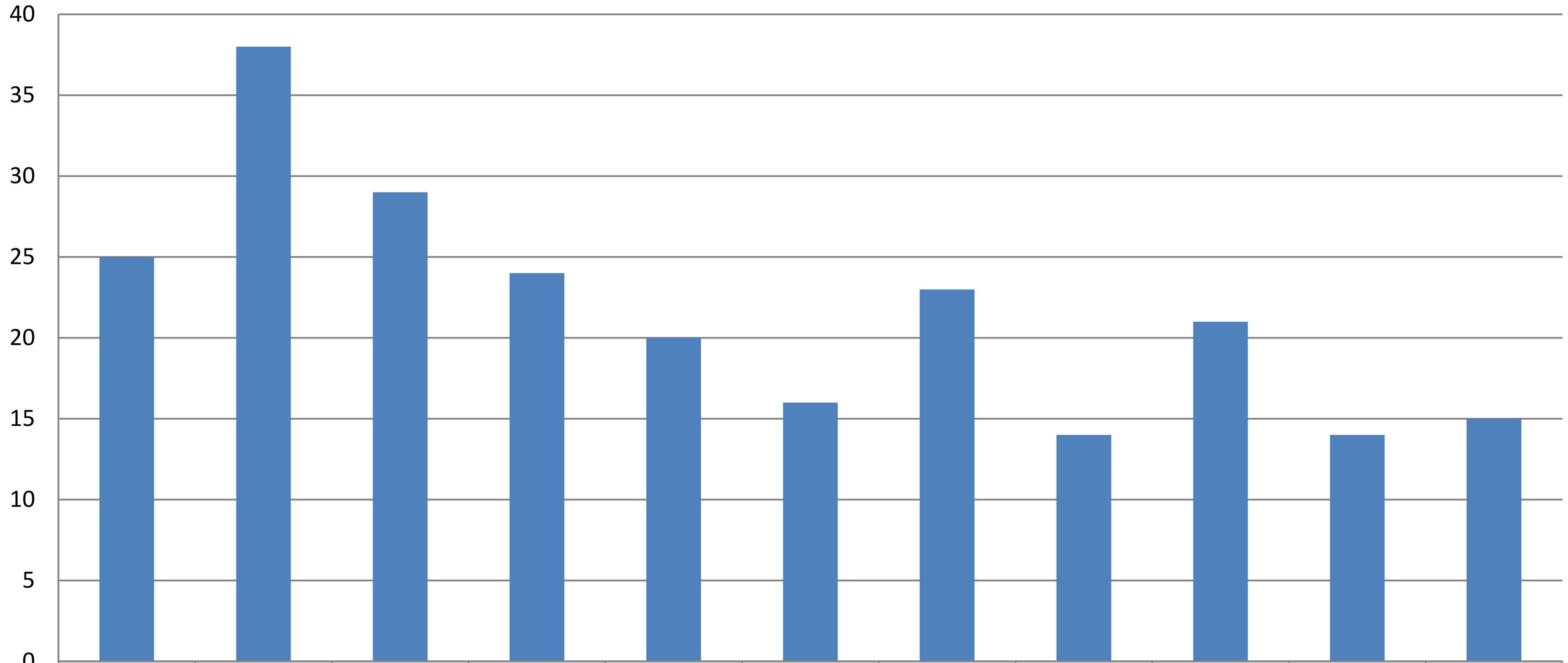


Average Coverage (%)



	SGS-1	SGS-2	SGS-4	SGS-8	RSS	DFS	RPS	NURS cn	NURS depth	NURS icnt	NURS md2u
Avg.	69.67	72.87	72.35	66.95	60.56	56.12	68.38	46.87	64.35	51.72	50.5

"Best" Counts



	SGS-1	SGS-2	SGS-4	SGS-8	RSS	DFS	RPS	NURS cn	NURS depth	NURS icnt	NURS md2u
Best	25	38	29	24	20	16	23	14	21	14	15

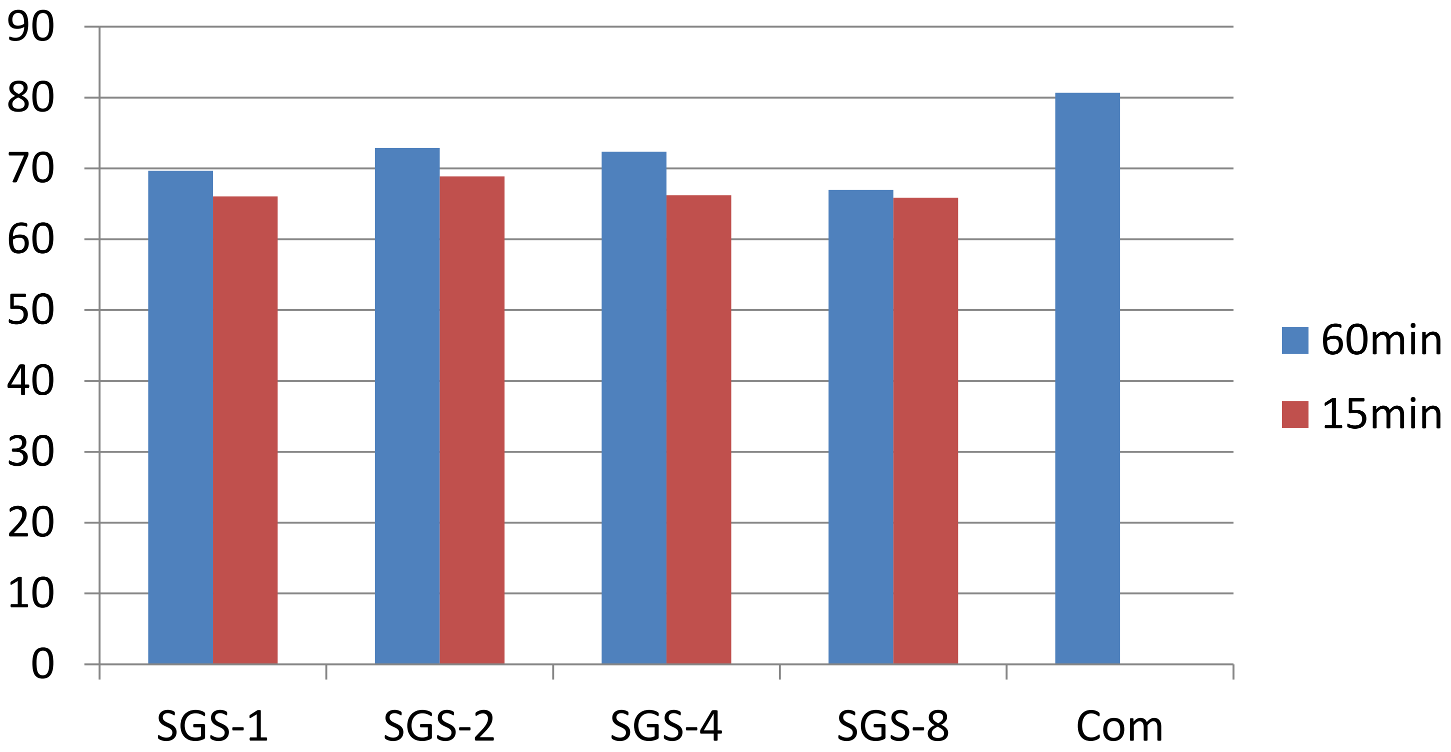
Results Recap

- Result 1: SGS yields higher coverage
- Result 2: No uniform best *n* for SGS

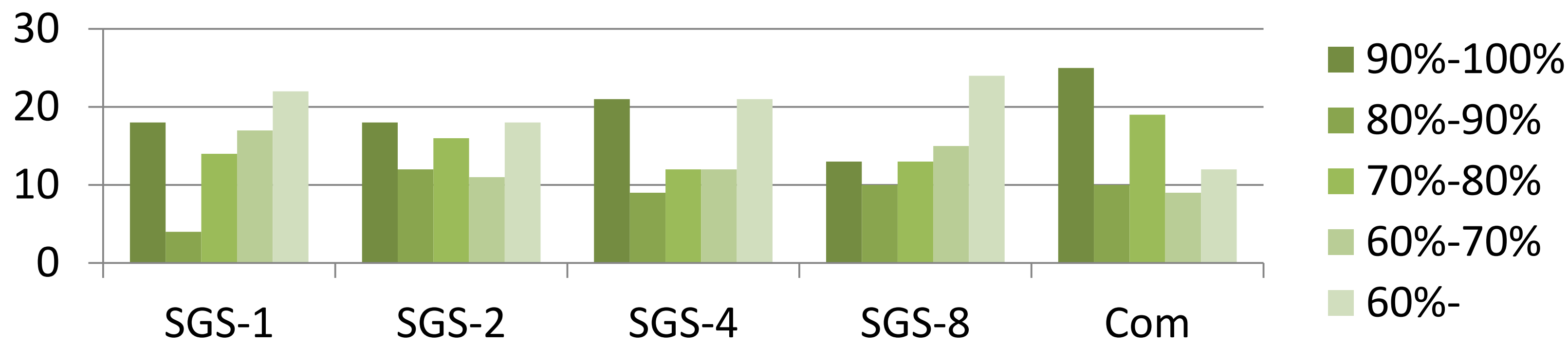
Combined SGS

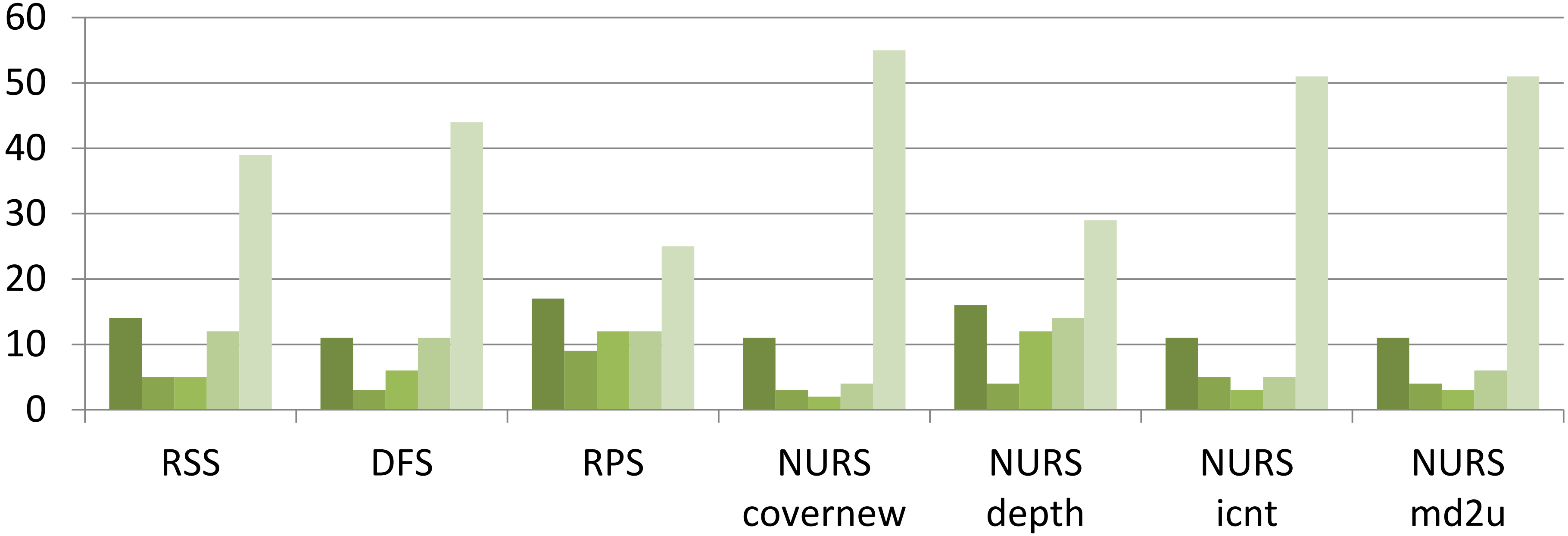
- Run SGS with length 1, 2, 4, 8 for 15 minutes each
- Combine all the generated test cases

Average Coverage (%)

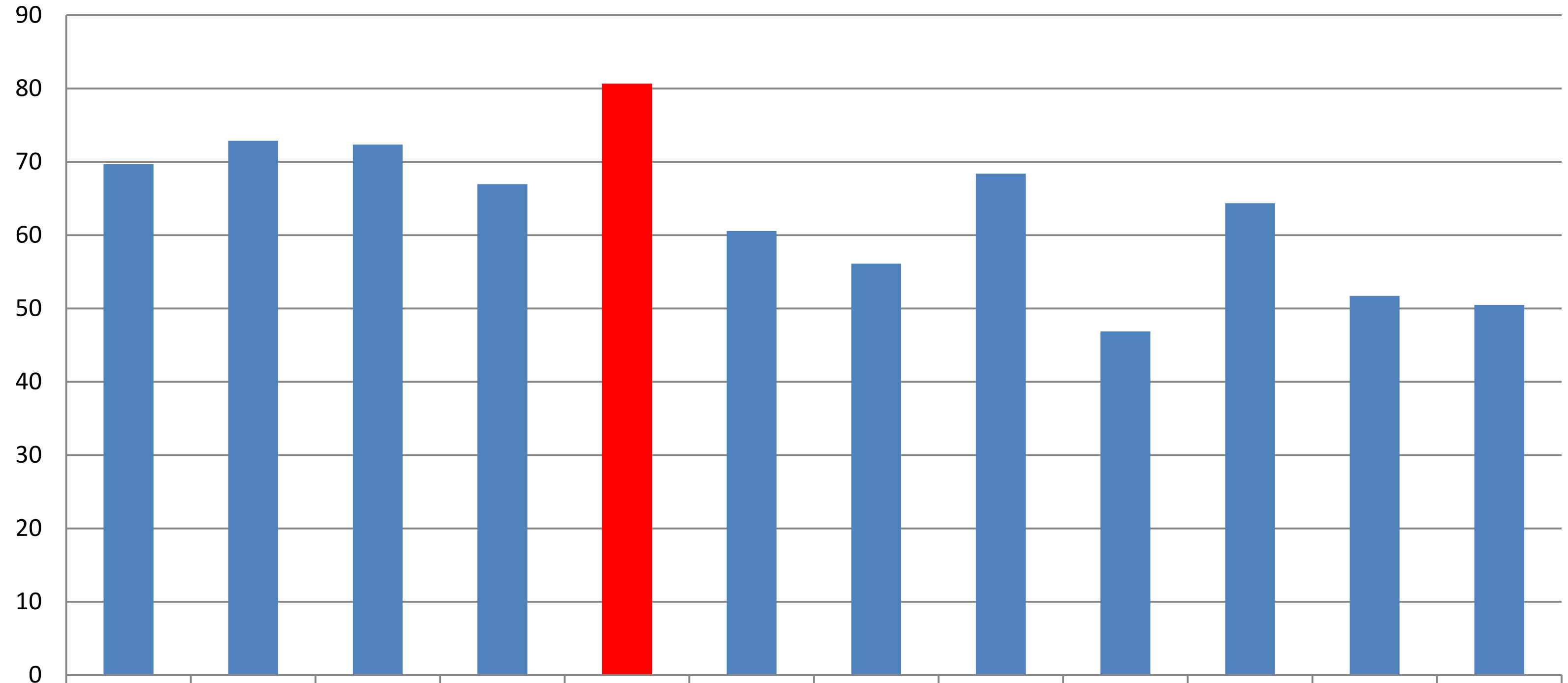


Coverage Distribution



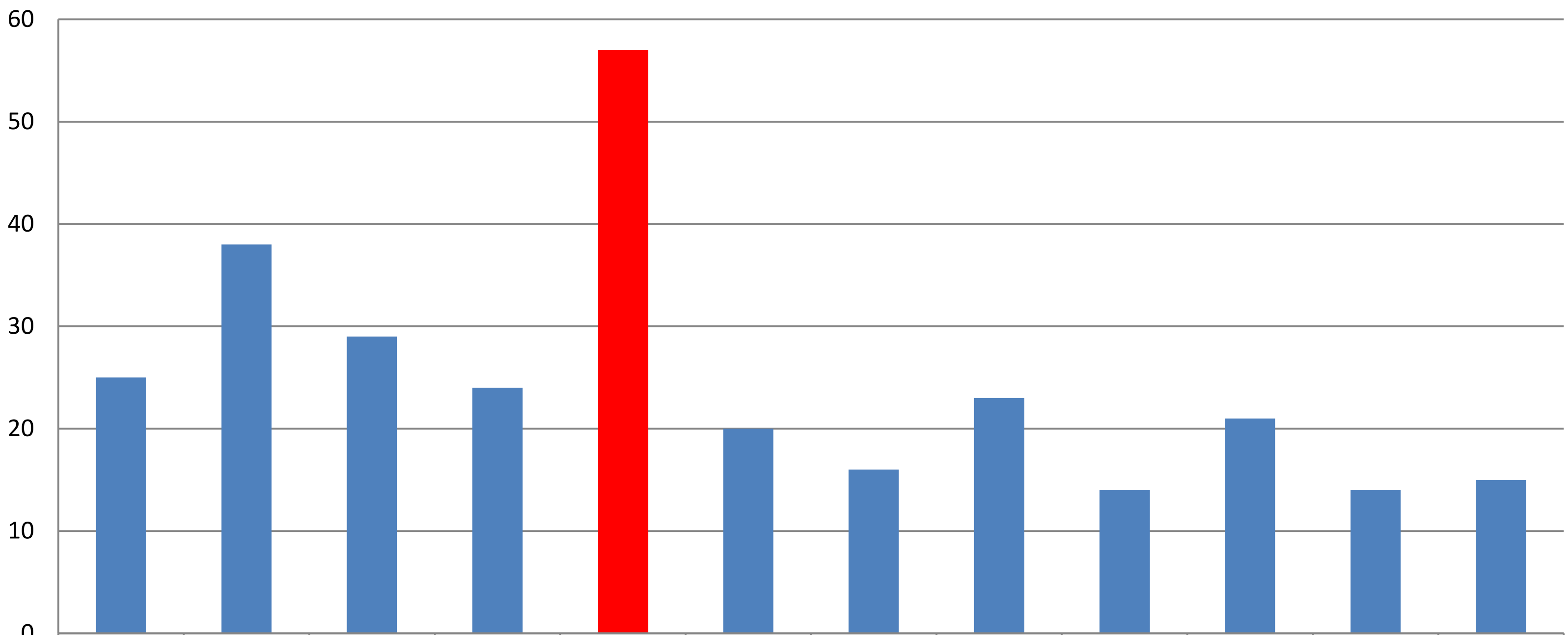


Average Coverage (%)



	SGS-1	SGS-2	SGS-4	SGS-8	Com	RSS	DFS	RPS	NURS cn	NURS depth	NURS icnt	NURS md2u
Avg.	69.67	72.87	72.35	66.95	80.67	60.56	56.12	68.38	46.87	64.35	51.72	50.5

"Best" Counts



	SGS-1	SGS-2	SGS-4	SGS-8	Com	RSS	DFS	RPS	NURS cn	NURS depth	NURS icnt	NURS md2u
Best	25	38	29	24	57	20	16	23	14	21	14	15

Results Recap

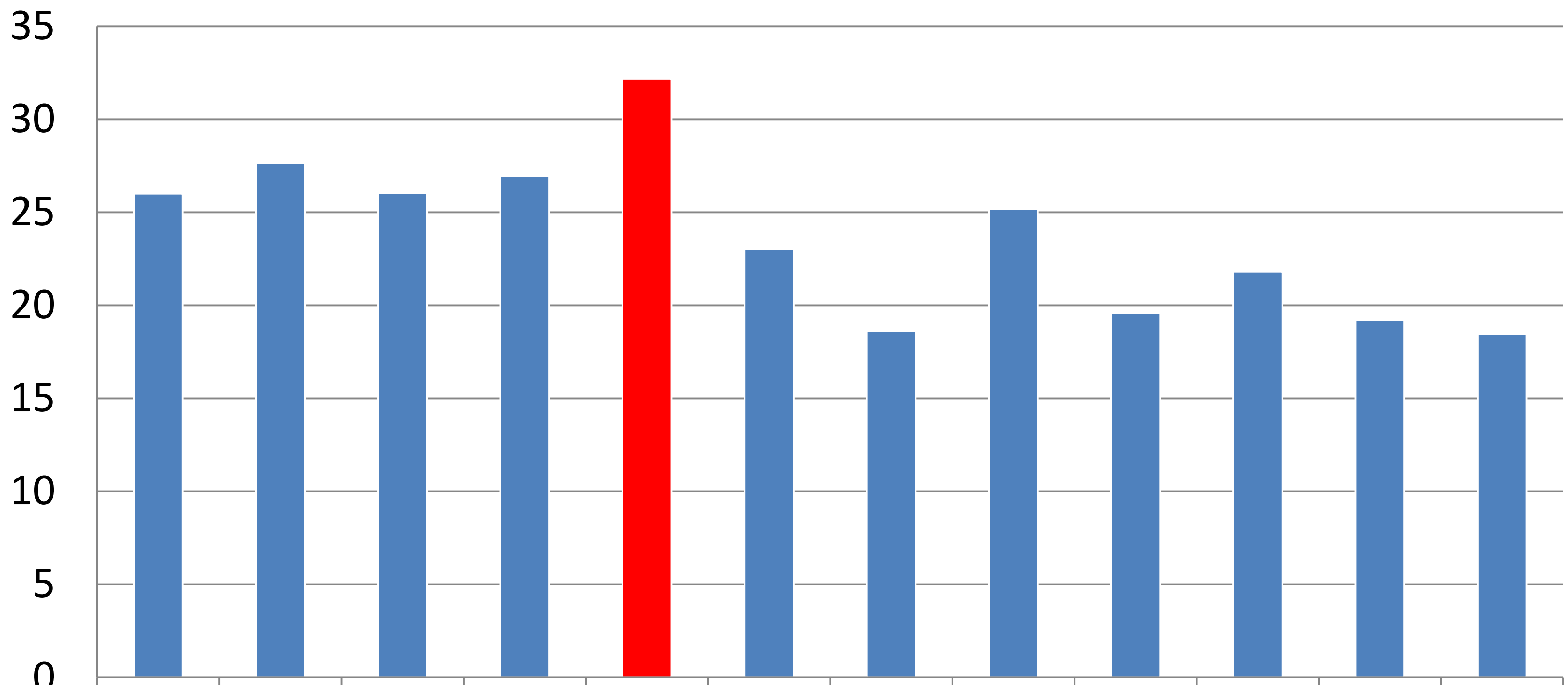
- Result 3: Combined SGS performs uniformly the best

- Result 4: SGS yields more bug reports
- Result 5: SGS has acceptable overhead

Bug Detection: Killing Mutants

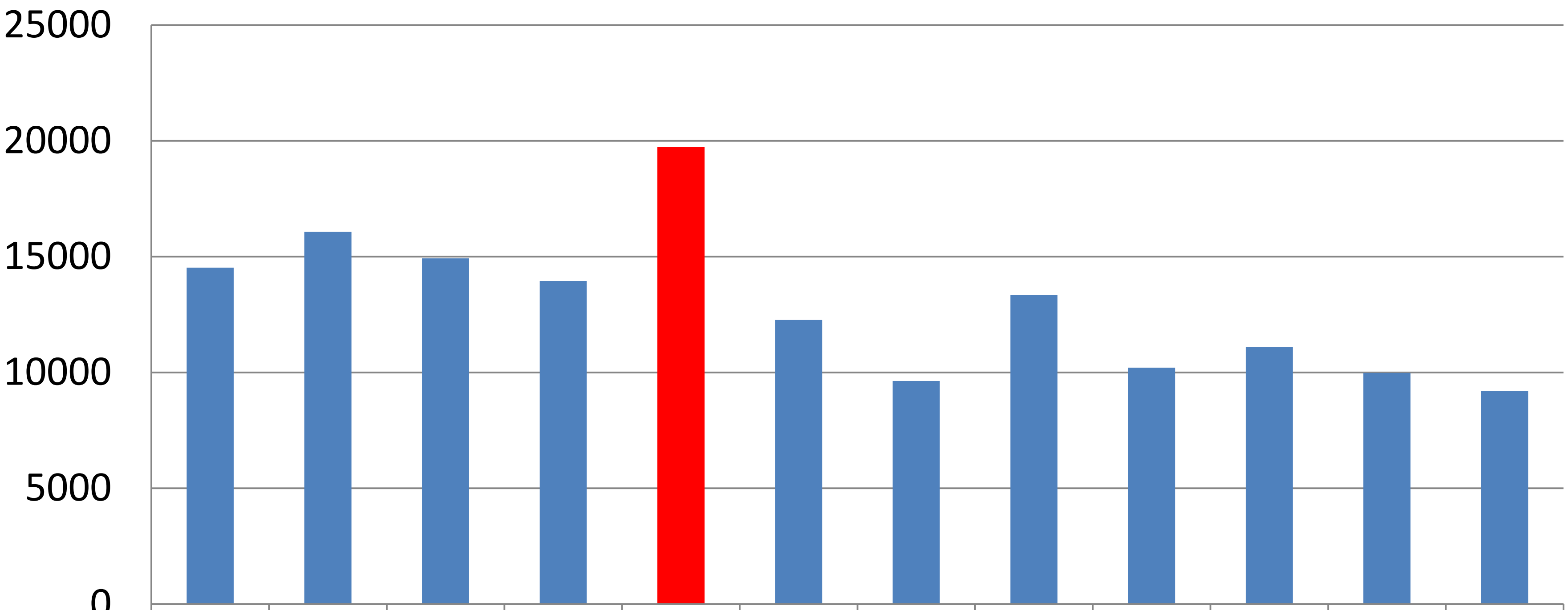
- 40 programs (which produce deterministic output)
- Run each different strategy for 1 hour
- Output all terminated test cases
- Generate mutants of the 40 programs
- Re-execute test cases on both original program and mutants
- Compare their outputs to see if mutants were killed

Average Kill Rate(%)



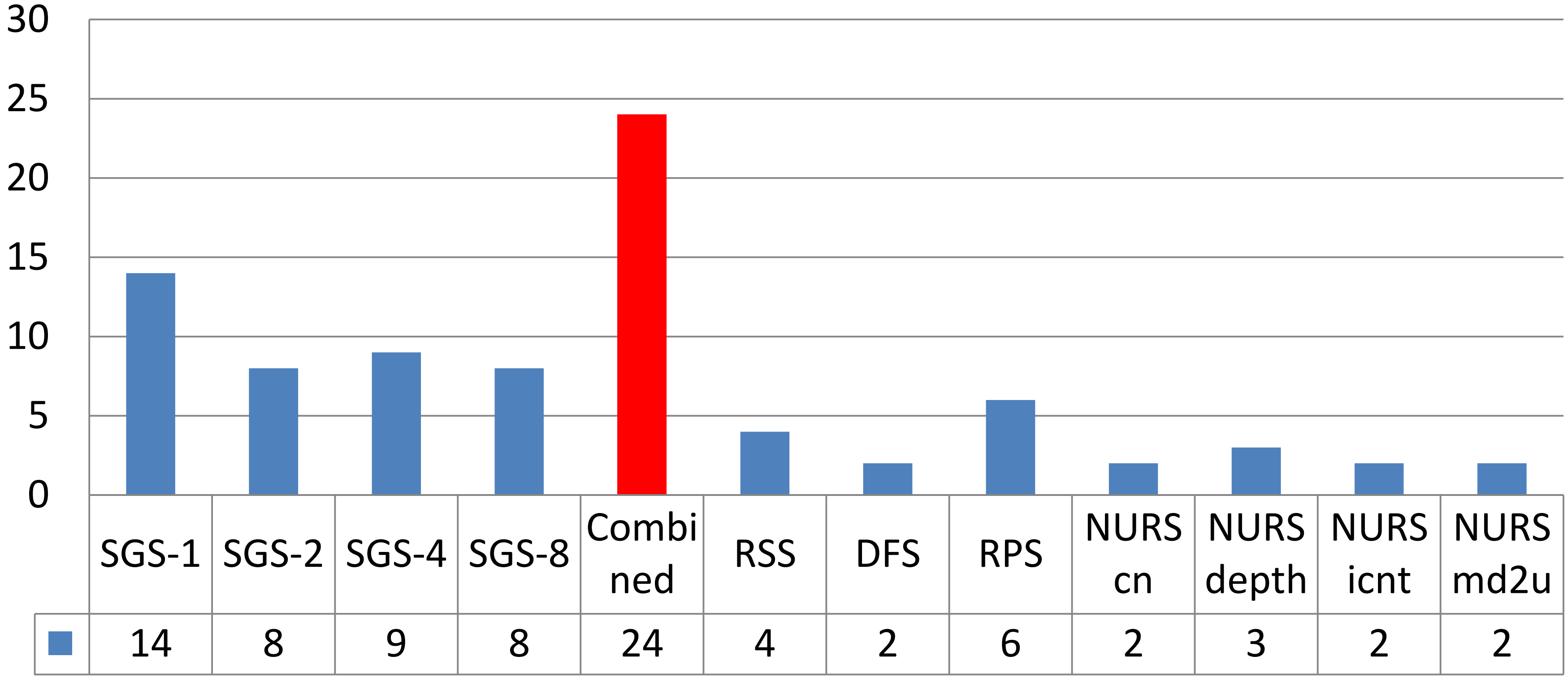
	SGS-1	SGS-2	SGS-4	SGS-8	Combined	RSS	DFS	RPS	NURS cn	NURS depth	NURS icnt	NURS md2u
■	26	27.65	26.04	26.96	32.17	23.03	18.63	25.16	19.58	21.8	19.23	18.44

Total Kill Number



	SGS-1	SGS-2	SGS-4	SGS-8	Combined	RSS	DFS	RPS	NURS cn	NURS depth	NURS icnt	NURS md2u
■	14529	16072	14930	13951	19719	12267	9632	13349	10211	11105	9987	9208

"Best" Counts



- Result 6: SGS kills more mutants

Impact of Different Length *n*

- Shorter length => less contextual information
- Longer length => more contextual information
- Combined SGS strikes a good balance
  - Efficiency
  - Effectiveness

Summary

- Introduced length-n path spectra to guide path exploration
  - Uniform, parameterized technique
  - Steering toward less traveled paths
- Implemented in KLEE and extensively evaluated
  - SGS outperforms existing search strategies
  - SGS exhibits different behavior with varying length *n*
  - Combined SGS performs the best

Thanks!