Symbolic Execution for Memory Consumption Analysis

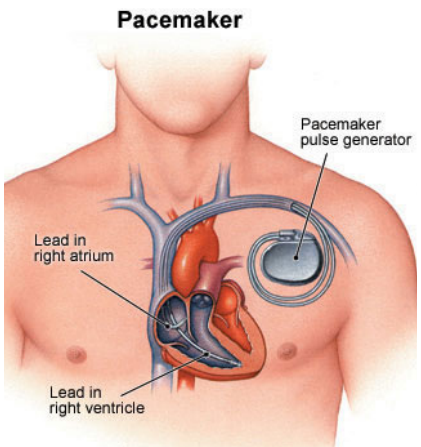
Duc-Hiep Chu, Joxan Jaffar, Rasool Maghareh

National University of Singapore (NUS)

LCTES 2016

Safety Critical Embedded Systems

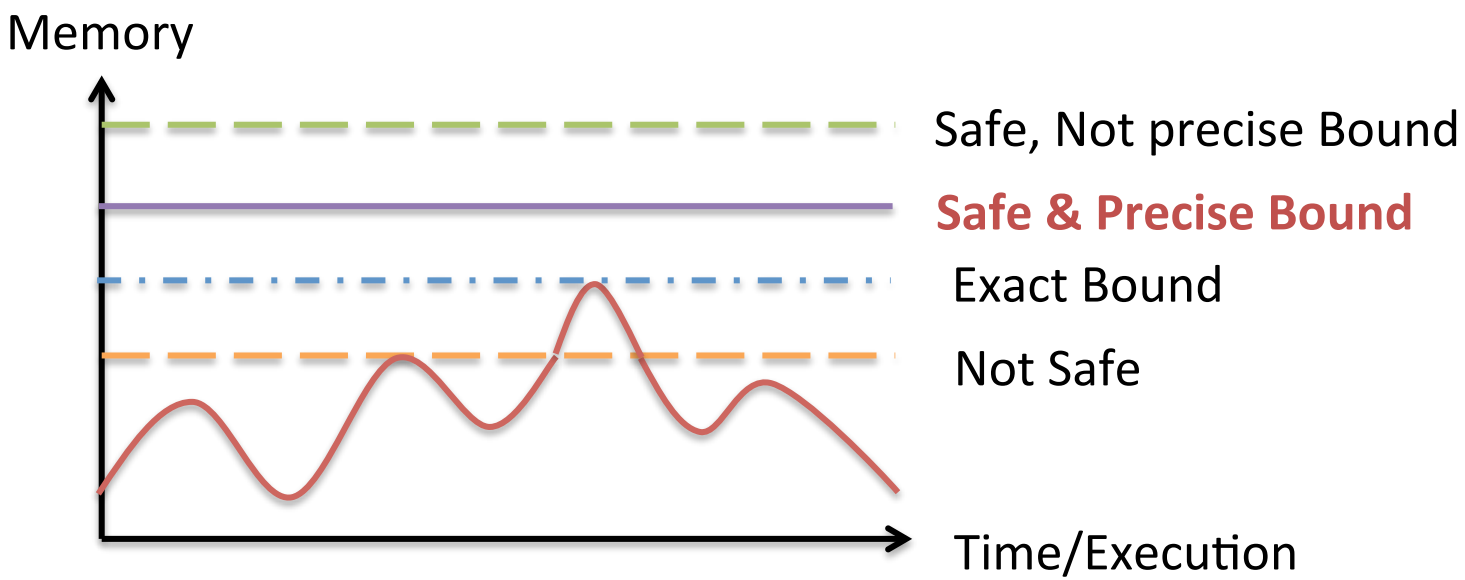
- Medical devices
- Aircraft flight control systems
- Automobiles



Is the use of Dynamic Memory Allocation safe?

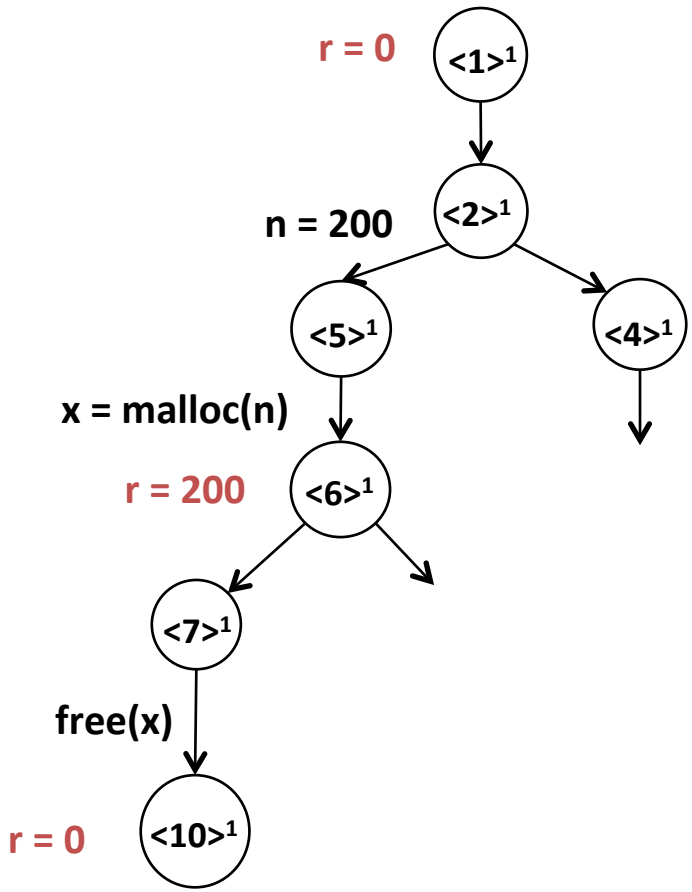
Worst-Case Memory Consumption Analysis

- Compute an upper bound of the memory consumption
- The bound should be sound and precise



Worst-Case Memory Consumption Analysis

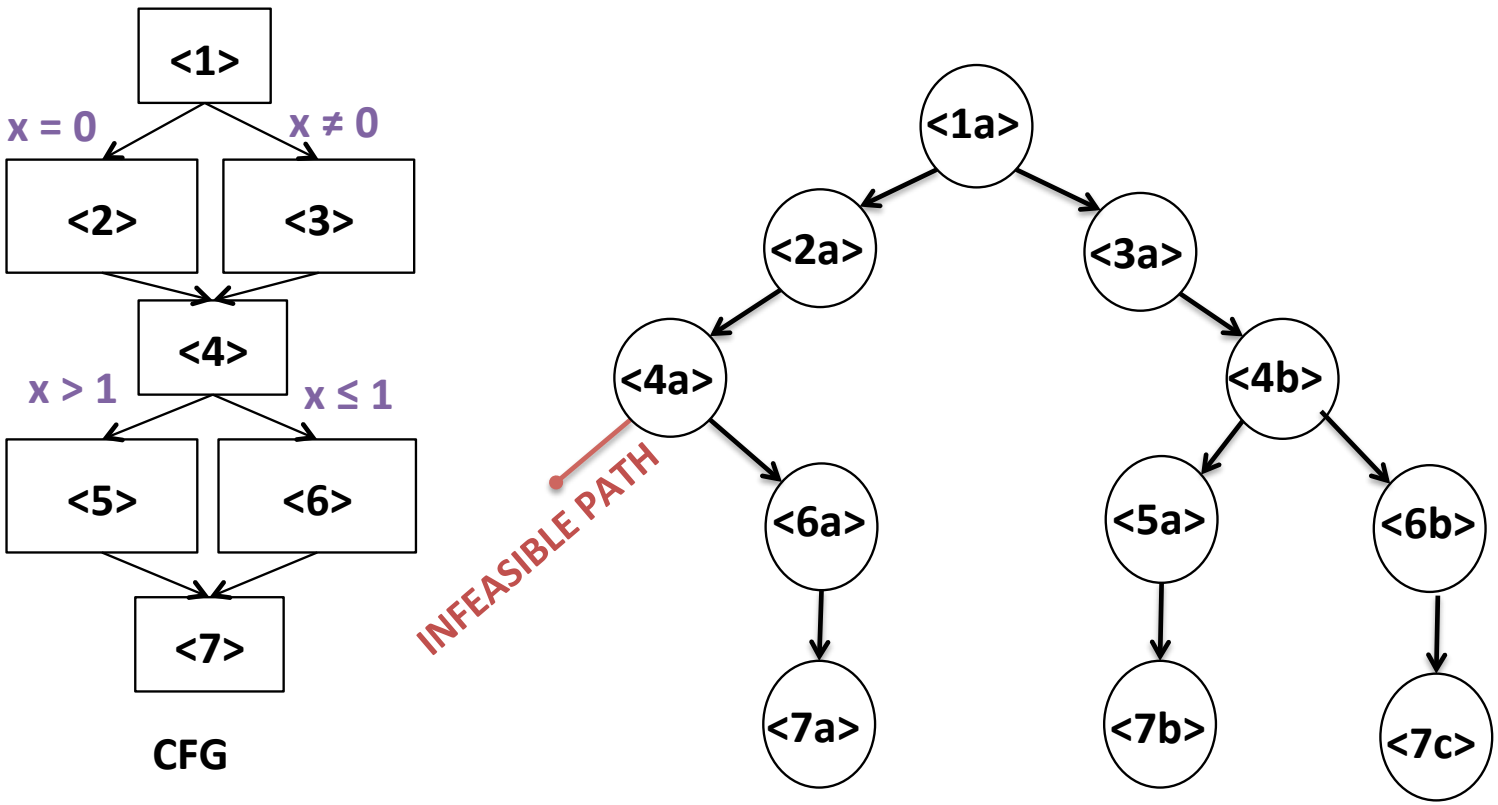
- Memory is a non-cumulative resource, i.e. it can be acquired then released. Example:
  - high-water mark usage = 200
  - net usage = 0



Traditional ILP-based (Integer Linear Programming) techniques for cumulative resource, e.g. timing, is no longer applicable

Symbolic Execution

- Represents the values of program variables as symbolic expressions of the input symbolic values
- Path condition: a formula over the symbolic inputs, which those inputs must satisfy in order for execution to follow that path
- A path is *infeasible* if its path condition is unsatisfiable



OUTLINE

Analysis Framework

- Worst-case heap usage analysis
- Loops and Recursions are bounded
  - Loops can just be fully unrolled
  - (valid assumption in in safety critical embedded systems)

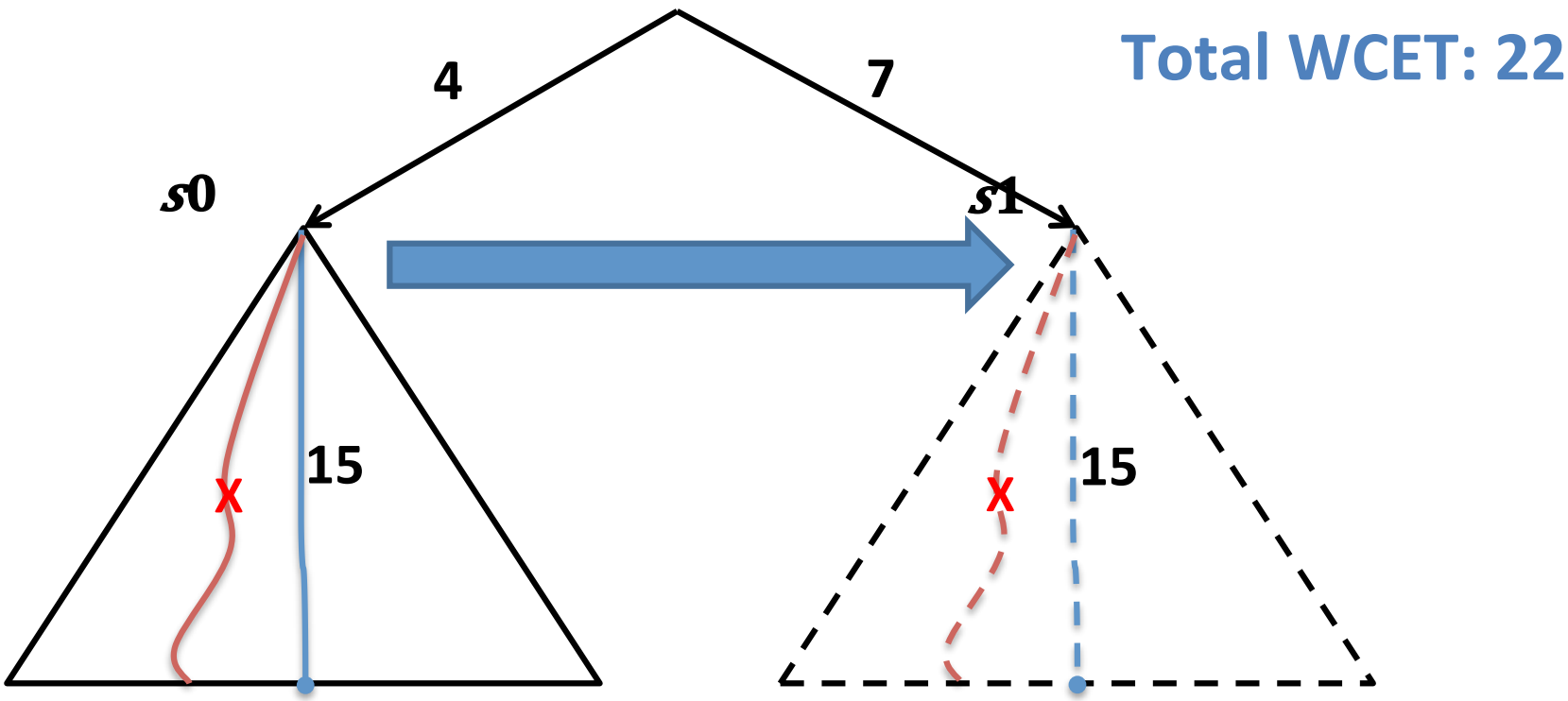
Analysis Framework

1. Build a symbolic execution tree from the input program
2. Track precisely the heap usage along all symbolic paths
3. Generate an aggregated upper bound

Scalability?

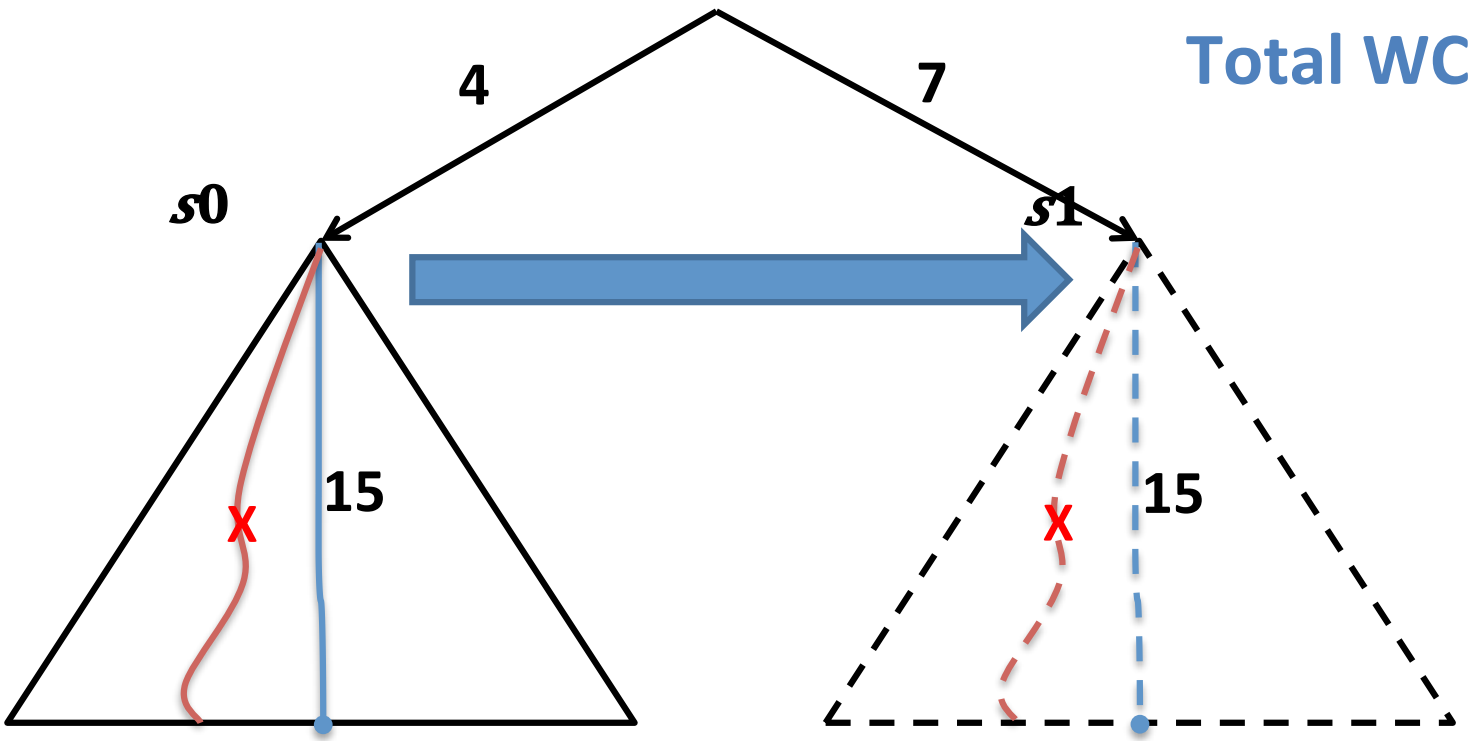
Scalable Symbolic Execution

- Our foundational work on WCET analysis [Chu and Jaffar 2011, Chu and Jaffar 2013]
  - Generalized form of dynamic programming
  - Analyzed subtrees are summarized and reused



Soundness of Reuse

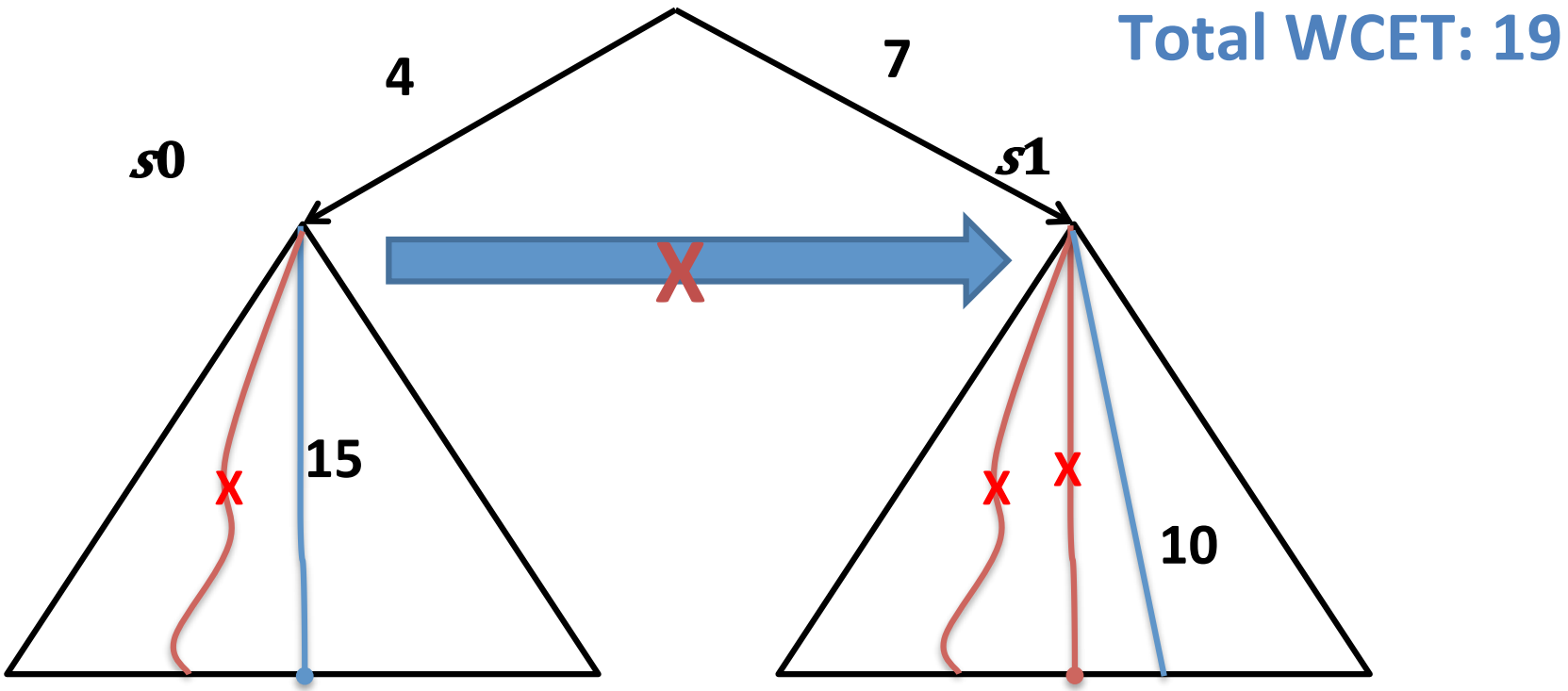
- Interpolating infeasible paths
  - ψ is an interpolant of $s0$, capturing the reason for infeasible paths in the left subtree
  - subsumption check: $s1 \models \psi$



Total WCET: 22

Precision of Reuse

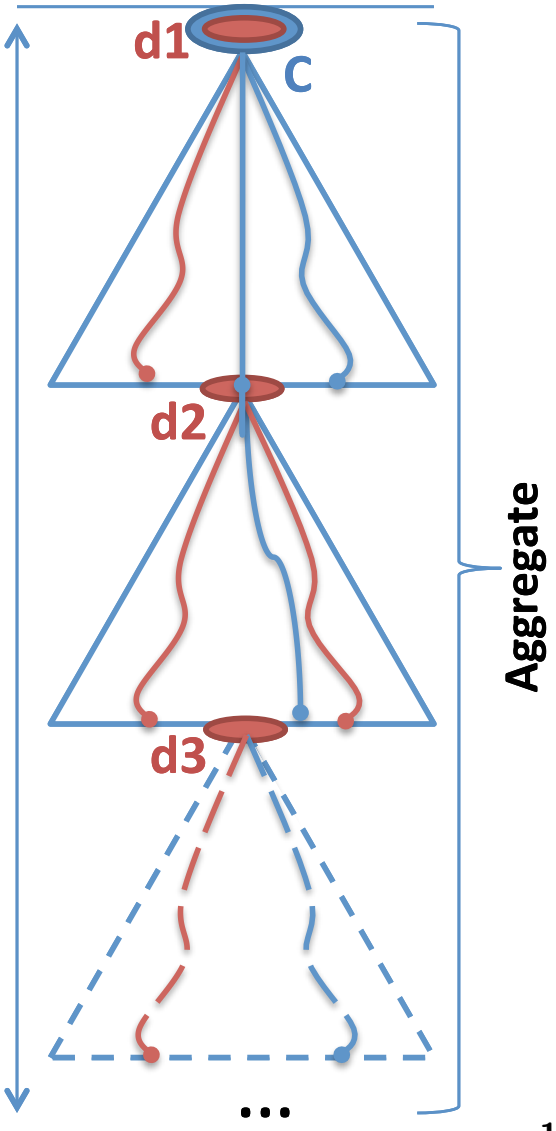
- Witness path: The most resource consuming path
- Witness check: Witness path must be feasible in the new context



Loop Unrolling with Summarization

- Critical for computing precise bounds when loops are complicated
  - Discover infeasible paths across iterations

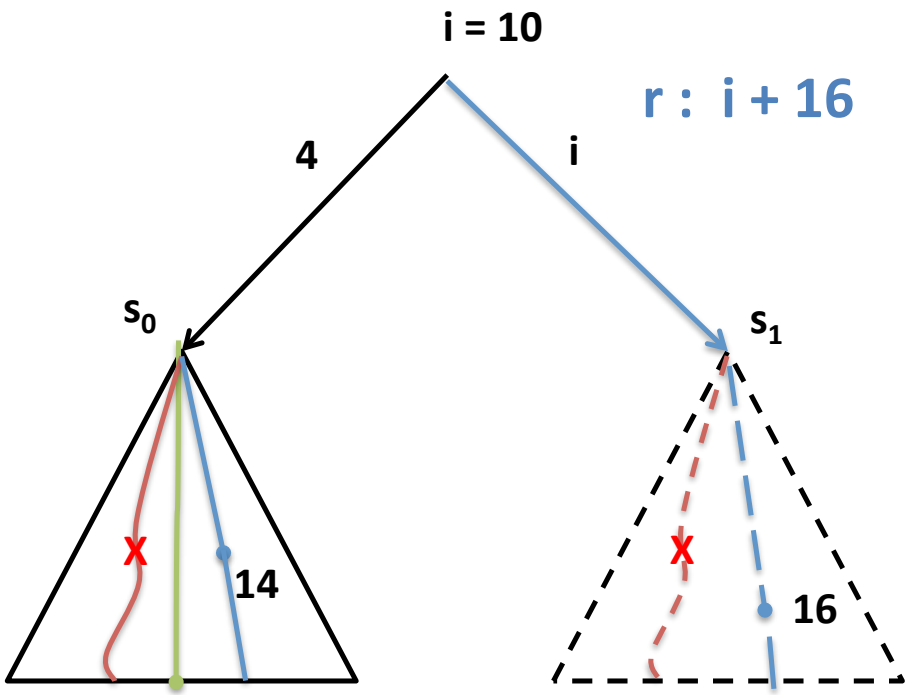
- Scalability has been shown in our prior works
  - Summarize each loop iteration
  - Reuse summarizations in later iterations
  - Summarize a sequence of loop iterations



Remaining Challenges

- Memory is a non-cumulative resource
  - For each summarization, we need two witnesses: a high-water mark witness and a net-usage witness
  - Needed when we combine summarizations of loop iterations

- The amount of memory allocated/deallocated in different iterations can be different, i.e. dynamic cost modelling
  - The memory consumption in each subtree is summarized using a symbolic expression



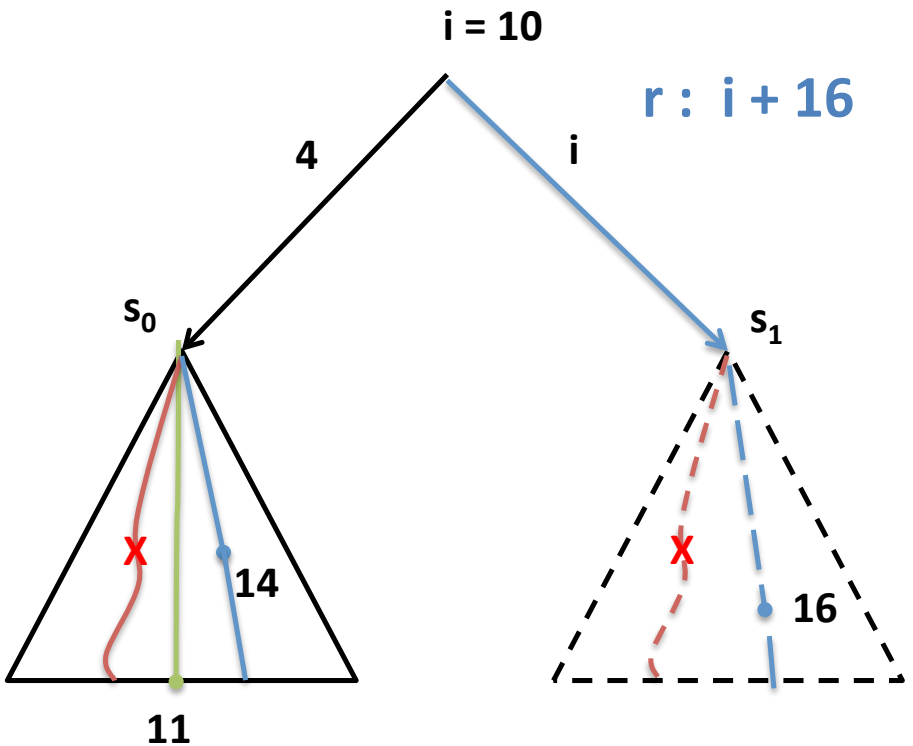
Summarization in HWM Analysis

Comprises of:

- an interpolant (as before)
- witness conditions (as before, but now for two types of witness)
- dominating conditions (for each witness):
  - a set of constraints over the program variables
  - ensuring the witness path is the most resource consuming, i.e. still dominating the other paths

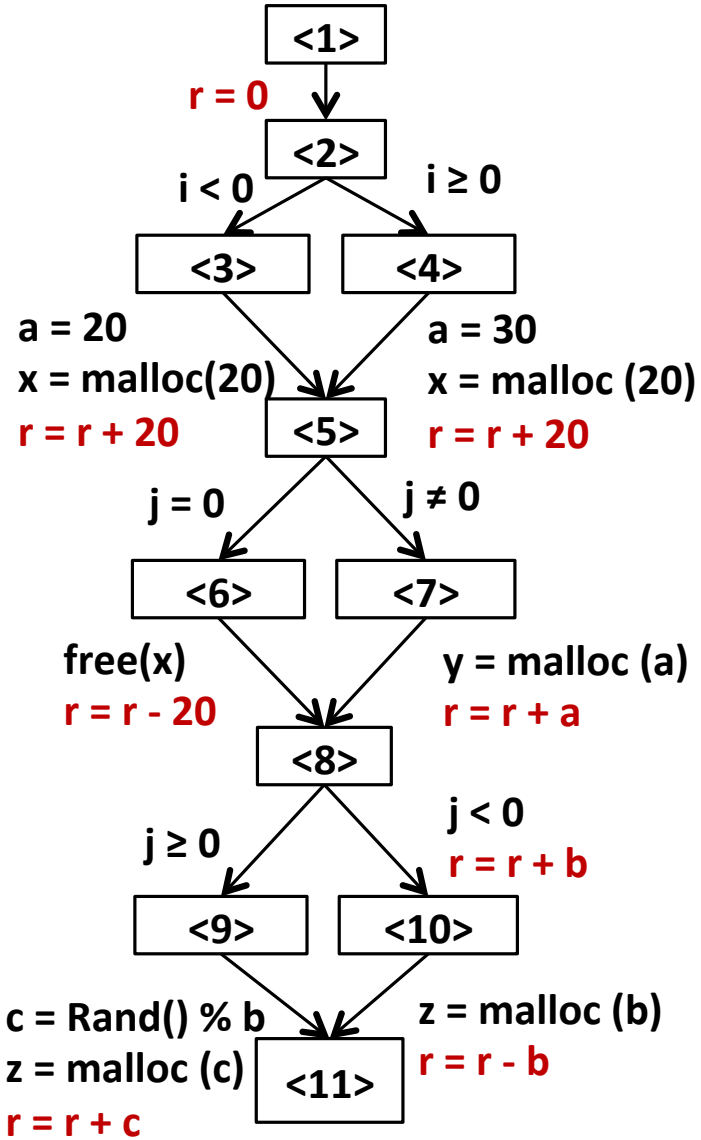
Summarization in HWM Analysis

- When $i = 10$, we have $i + 16 > 4 + 14$
  - $i = 10$ is a trivial dominating condition
  - a weaker condition is $i \geq 2$



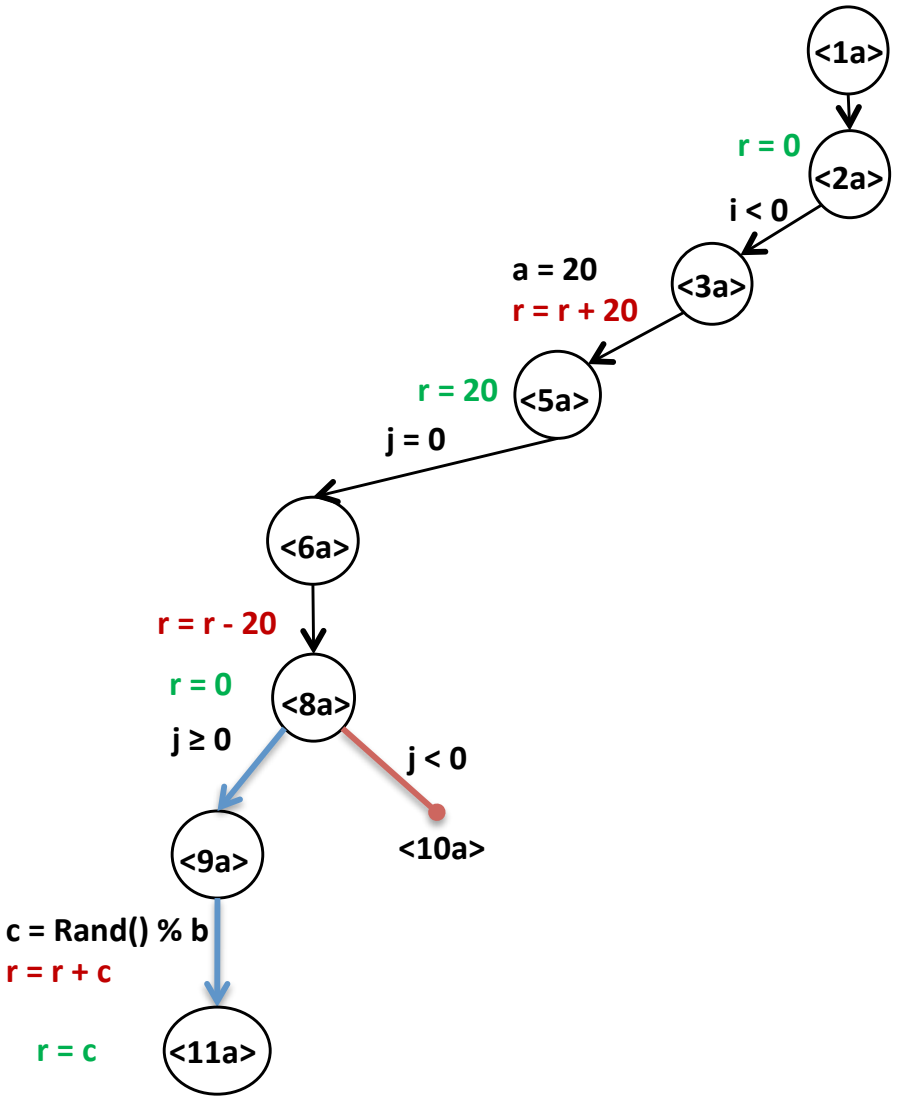
OUTLINE

Example


<1>
r = 0
<2>
i < 0
i ≥ 0
<3>
<4>
a = 20
x = malloc(20)
r = r + 20
a = 30
x = malloc (20)
r = r + 20
<5>
j = 0
j ≠ 0
<6>
<7>
free(x)
r = r - 20
y = malloc (a)
r = r + a
<8>
j ≥ 0
j < 0
r = r + b
<9>
<10>
c = Rand() % b
z = malloc (c)
r = r + c
z = malloc (b)
r = r - b
<11>

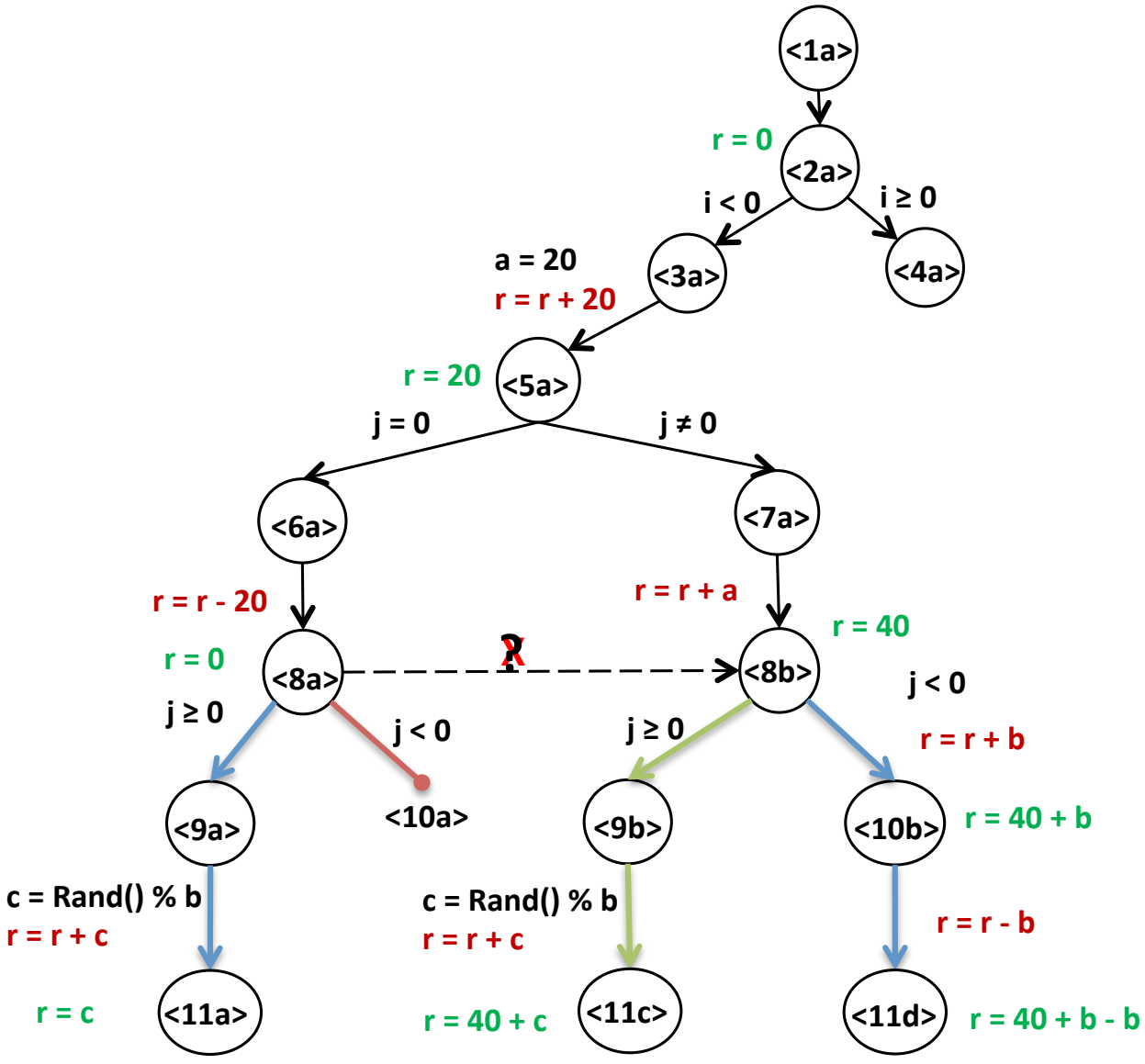
Example

At ⟨8a⟩, high-water mark witness: ([+,c])



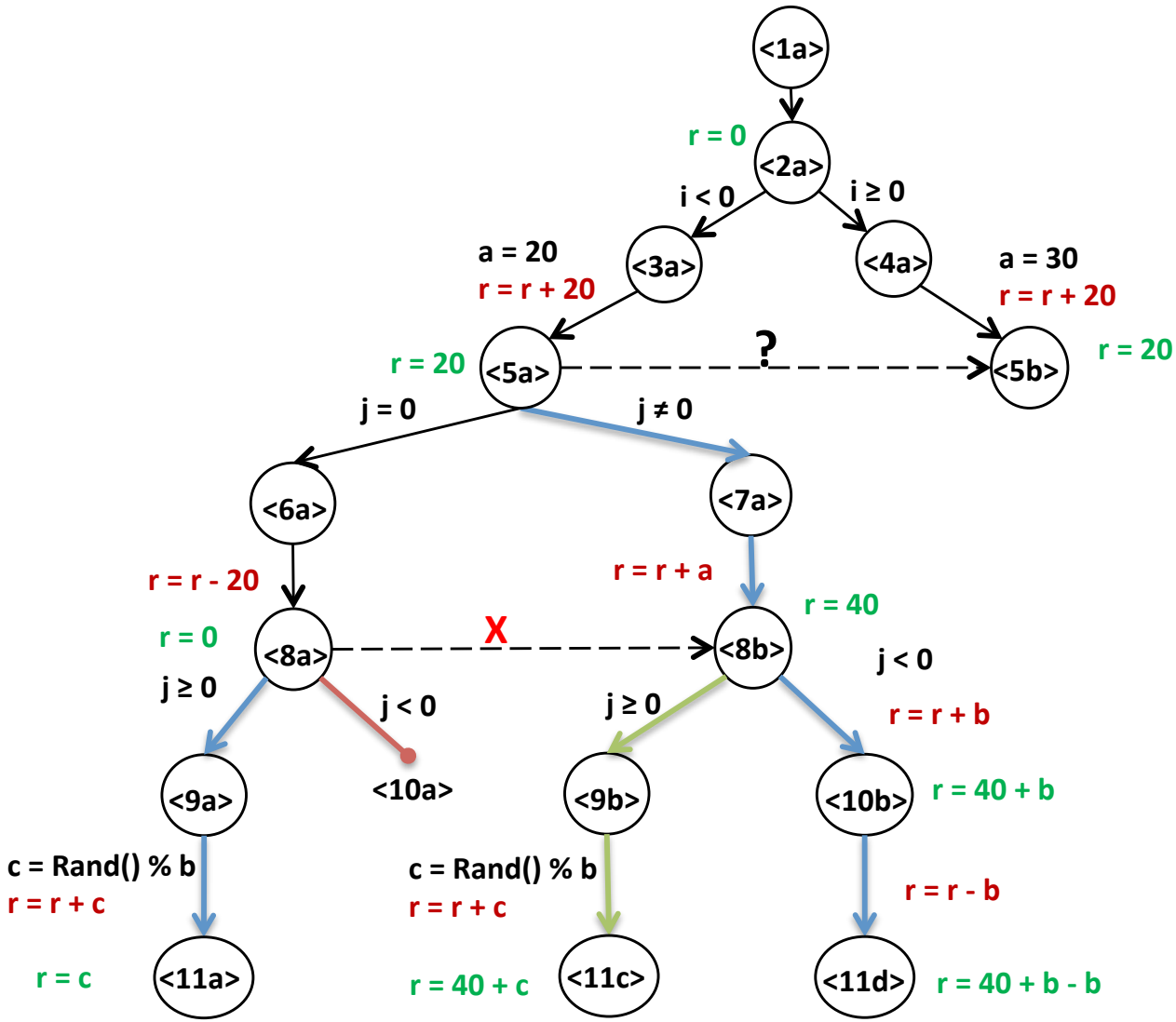
Example

At ⟨8b⟩, high-water mark witness: ([+,b])



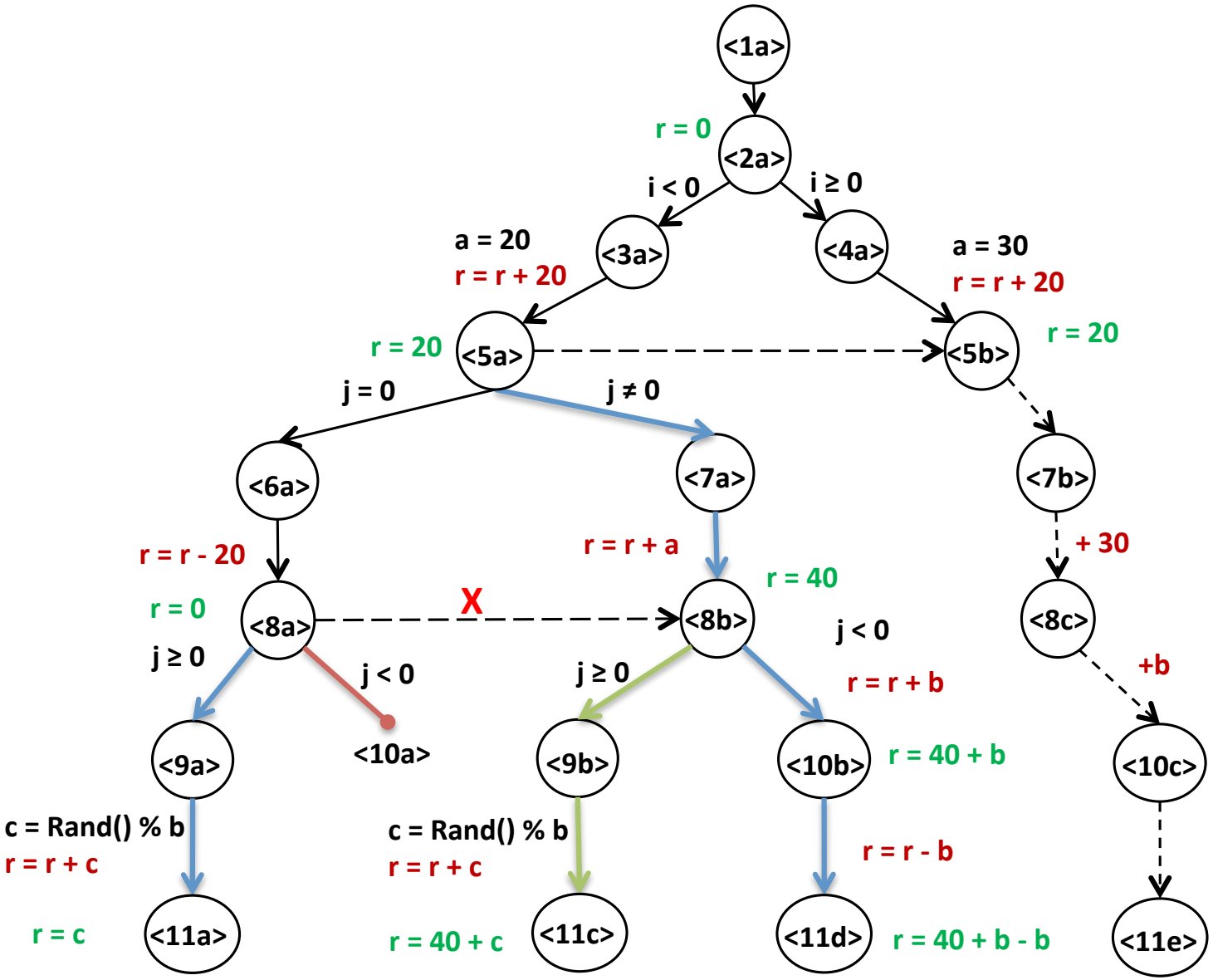
Example

At ⟨5a⟩, high-water mark witness: ([+,a],[+,b])



Example

At $\langle 1a \rangle$, the worst-case analysis is $50 + b$



OUTLINE

Table 1

Benchmark	LOC	Input Parameters	Time	States	Reuses	Bound
larson	614	threads = 1, num_chunks = 100	55.55	613	198	240050000
ndes	219	N.A.	21.21	643	201	11214
puzzle	197	N.A.	164.43	1094	354	204
fasta	121	N.A.	0.23	91	17	755
chomp	401	N.A.	1.59	153	36	6800
statemate	1090	N.A.	233.63	3553	1296	64
nsicheneu	3144	N.A.	791.75	3639	1376	112
shbench	121	threads = 1, Nalloc = 100	554.39	4461	1408	43600
himenobmtxpa	272	N.A.	175.45	1472	406	57344
dry	491	N.A.	0.3	142	39	112
fft1 (main)	234	MAXWAVES = 8	5.23	88	23	16 * MAXSIZE + 64
nsieve-bits	33	N.A.	4.74	552	192	sz / 8 + 4
ffbench	287	Asize = 10	138.56	1550	508	262160

Table 2

Benchmark	LOC	Input Parameters	Time	States	Reuses	Bound
cache-thrash	120	threads = 1, iterations = 100	7.96	344	108	1
cache-thrash	120	threads = 2, iterations = 100	8.00	329	103	1
cache-thrash	120	threads = 1, iterations = 200	58.96	644	208	1
cache-thrash	120	threads = 2, iterations = 200	57.60	629	203	1
cache-scratch	126	threads = 1, iterations = 100	9.32	350	108	9
cache-scratch	126	threads = 2, iterations = 100	9.26	338	104	18
cache-scratch	126	threads = 1, iterations = 200	68.86	650	208	9
cache-scratch	126	threads = 2, iterations = 200	69.40	638	204	18

objSize = 1, repetitions = 10

Summary

- Worst-case memory consumption analysis is important for safety critical embedded system
- Memory is a non-cumulative resource
  - Acquired then later released
  - Traditional ILP-based methods are no longer applicable
- Symbolic Execution can be scalable
  - Reuse is key

Thank You!!!

Q & A