Loop Unwinding in the 2LS Framework for Analysis of Dynamically Allocated Memory

František Nečas

September 13, 2022

The 2LS Framework

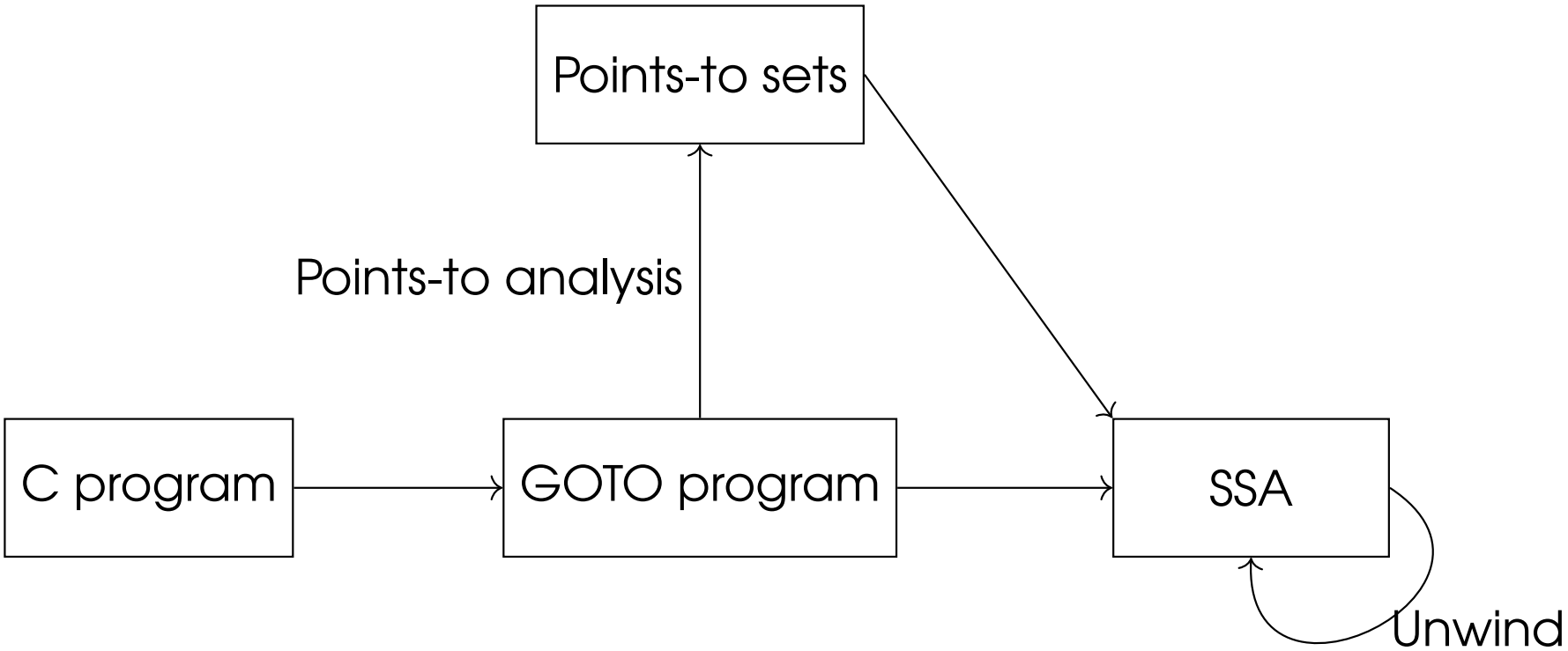
- Static analysis tool for C programs built upon the CProver infrastructure
- Computes loop invariants
- Algorithm *k*I*k*I = *k*-induction, bounded model checking and abstract interpretation
  - Loop unwinding plays a key role
- SSA internal representation facilitates usage of an incremental SMT solver

Motivation Example

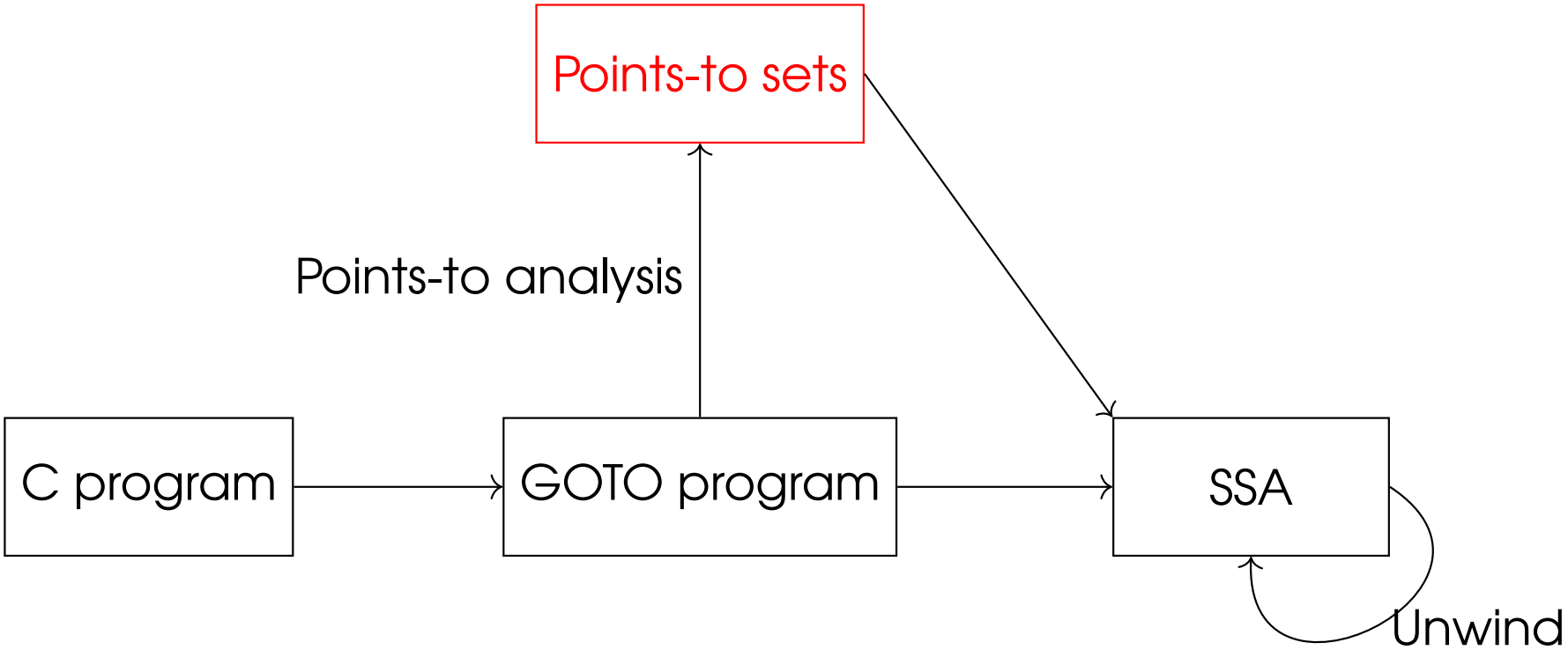
- Unbounded linked list containing mostly the value 1
- 2LS previously failed to find counterexample
- Reason: original unwinder does not support dynamic memory

```
Node *n;
Node *p = 0;
int i = 0;
while (nondet_int()) {
  n = malloc(sizeof(Node));
  n->val = i == 2 ? 2: 1;
  n->next = p;
  p = n;
  i++;
}
while (p != 0) {
  assert(p->val == 1);
  p = p->next;
}
```

Root Cause of the Problem



Root Cause of the Problem



- Changes to SSA should be minimal to utilize incremental solving $\rightarrow$ pre-compute results of dereferencing
- Malloc calls are replaced by a new abstract dynamic object:

$$\texttt{malloc(sizeof(x))} \longrightarrow \&do_i.$$

- Assume pointer p may point to one of objects $\{o_1, o_2, o_3\}$
- R-value expression $p \rightarrow f$ is represented in SSA as:

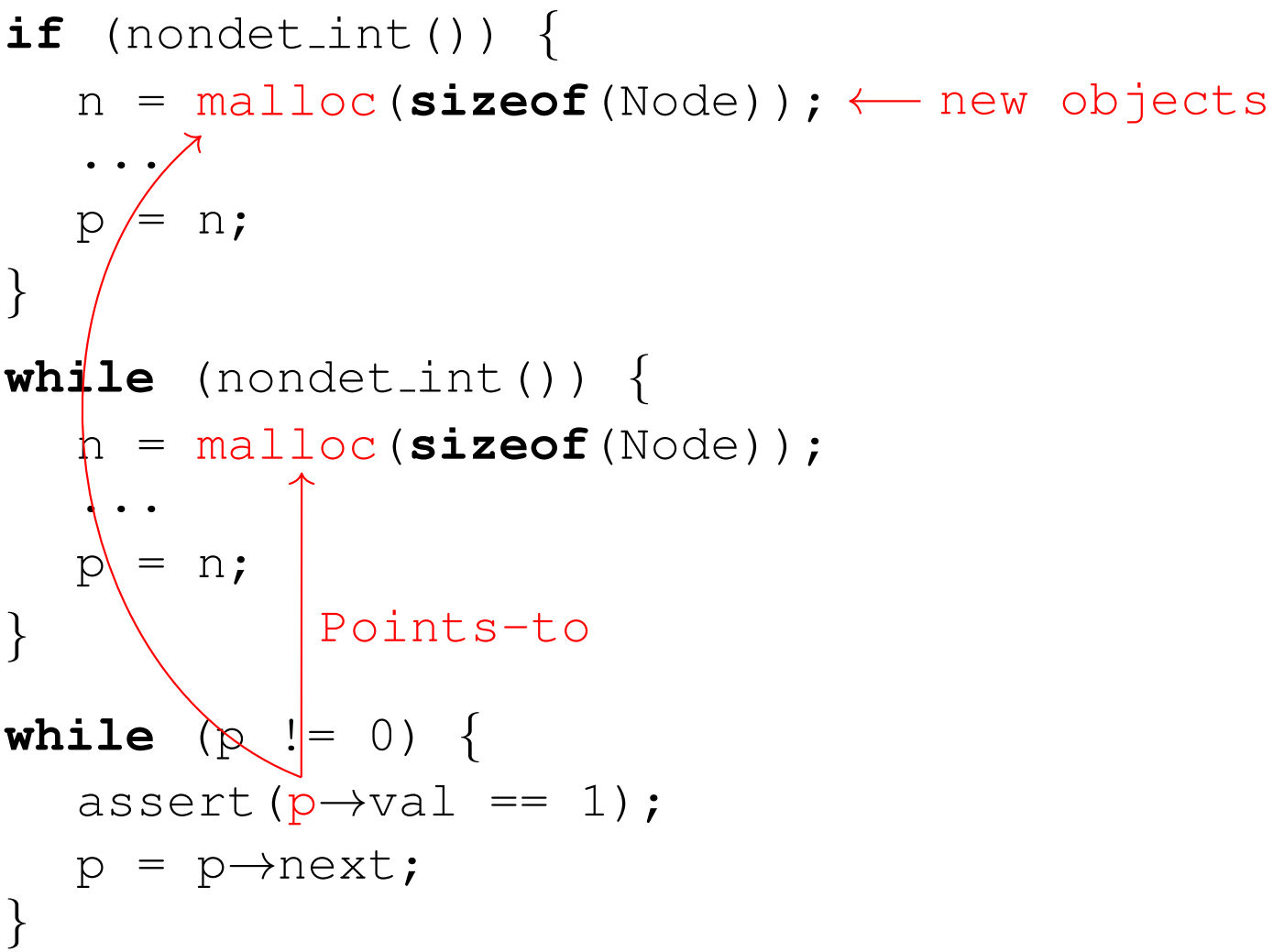
$$p = \&o_1 \Rightarrow o_1.f \wedge$$
$$p = \&o_2 \Rightarrow o_2.f \wedge$$
$$p = \&o_3 \Rightarrow o_3.f$$

Points-to Analysis and Unwinding

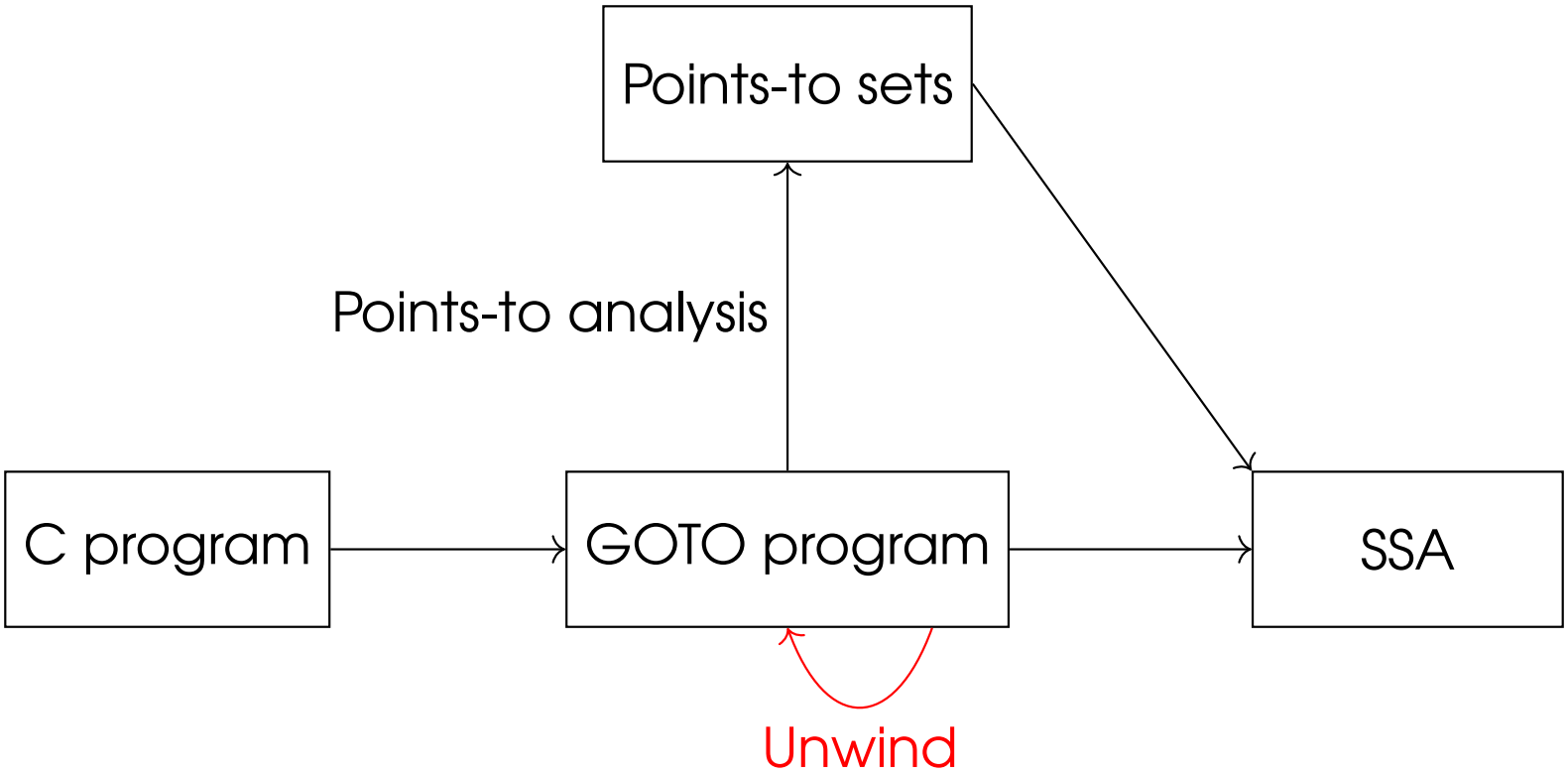
```
while (nondet_int()) {
   n = malloc(sizeof(Node));
   ...
   p = n;
}

while (p != 0) {
   assert(p→val == 1);
   p = p→next;
}
```

Points-to

Points-to Analysis and Unwinding



Proposed Solution



- Benchmarks from SV-COMP, 5-minute timeout

	Reach Safety 1574 tasks		**Memory Safety** 408 tasks	
	Old	New	Old	New
Correct results	776	814	92	143
Correct true	603	622	55	81
Correct false	173	192	37	62
Incorrect results	2	1	3	6
Incorrect true	1	0	1	2
Incorrect false	1	1	2	4
Score	**1331**	**1420**	**83**	**96**
CPU time per task (s)	104.2	107.4	47.5	47.5

- Benchmarks from SV-COMP, 5-minute timeout

	Reach Safety 1574 tasks		**Memory Safety** 408 tasks	
	Old	New	Old	New
Correct results	776	814	92	143
Correct true	**603**	**622**	55	81
Correct false	**173**	**192**	37	62
Incorrect results	2	1	3	6
Incorrect true	1	0	1	2
Incorrect false	1	1	2	4
Score	1331	1420	83	96
CPU time per task (s)	104.2	107.4	47.5	47.5

- Benchmarks from SV-COMP, 5-minute timeout

	Reach Safety 1574 tasks		**Memory Safety** 408 tasks	
	Old	New	Old	New
Correct results	776	814	92	143
Correct true	603	622	**55**	**81**
Correct false	173	192	**37**	**62**
Incorrect results	2	1	3	6
Incorrect true	1	0	1	2
Incorrect false	1	1	2	4
Score	1331	1420	83	96
CPU time per task (s)	104.2	107.4	47.5	47.5

- Benchmarks from SV-COMP, 5-minute timeout

	Reach Safety 1574 tasks		**Memory Safety** 408 tasks	
	Old	New	Old	New
Correct results	776	814	92	143
Correct true	603	622	55	81
Correct false	173	192	37	62
Incorrect results	2	1	3	6
Incorrect true	1	0	1	2
Incorrect false	1	1	2	4
Score	1331	1420	83	96
CPU time per task (s)	**104.2**	**107.4**	**47.5**	**47.5**

- Benchmarks from SV-COMP, 5-minute timeout

	Reach Safety 1574 tasks		**Memory Safety** 408 tasks	
	Old	New	Old	New
Correct results	776	814	92	143
Correct true	603	622	55	81
Correct false	173	192	37	62
Incorrect results	2	1	3	6
Incorrect true	1	0	**1**	**2**
Incorrect false	1	1	**2**	**4**
Score	1331	1420	83	96
CPU time per task (s)	104.2	107.4	47.5	47.5

- Benchmarks from SV-COMP, 5-minute timeout

	Reach Safety 1574 tasks		**Memory Safety** 408 tasks	
	Old	New	Old	New
Correct results	776	814	92	143
Correct true	603	622	55	81
Correct false	173	192	37	62
Incorrect results	2	1	3	6
Incorrect true	1	0	1	2
Incorrect false	1	1	2	4
Score	**1331**	**1420**	**83**	**96**
CPU time per task (s)	104.2	107.4	47.5	47.5

Future Focus

- Support for recursive function calls
  - Combine unwinding with inlining when inconclusive
- Utilize incremental SAT solving