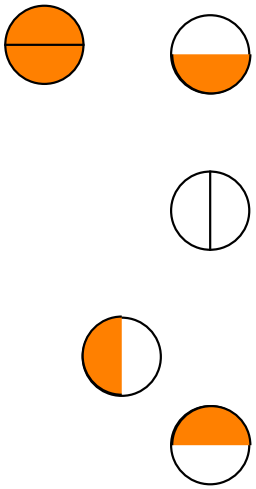
How to Join Proofs and Witnesses When Splitting Verification Task Among Verifiers?

Marie-Christine Jakobs

Why Splitting Verification Tasks?
The Performance Reason

Splitting allows parallelization

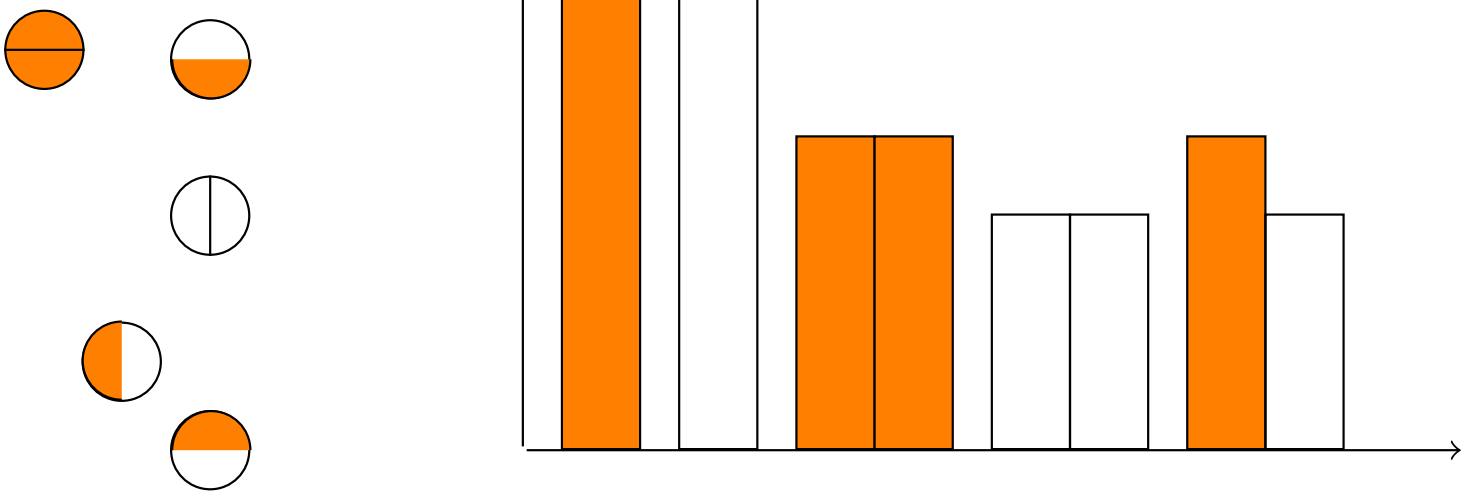
$\Longrightarrow$ aims at decreasing wall time



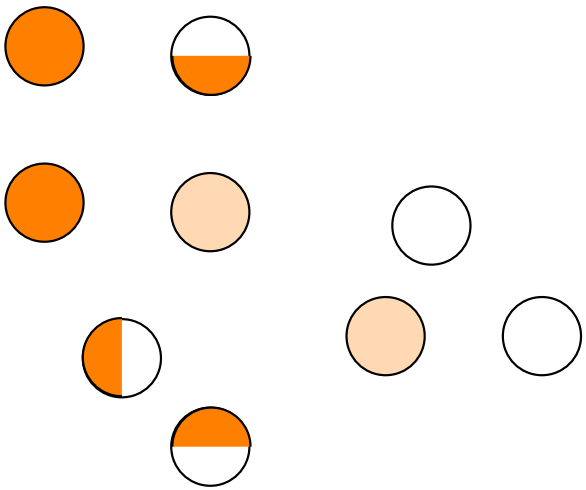
Why Splitting Verification Tasks?
The Performance Reason

Splitting allows parallelization

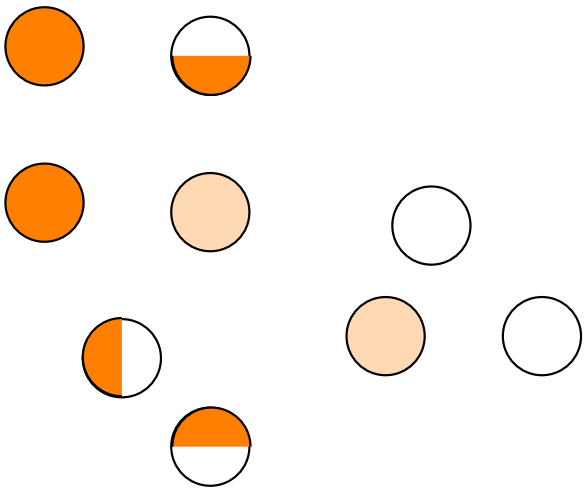
$\Longrightarrow$ aims at decreasing wall time



Why Splitting Verification Tasks? The Efficacy Reason



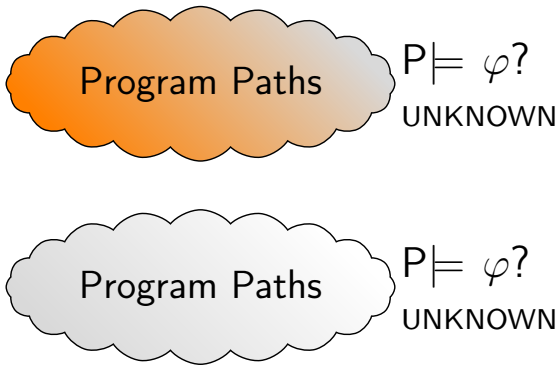
Why Splitting Verification Tasks?
The Efficacy Reason



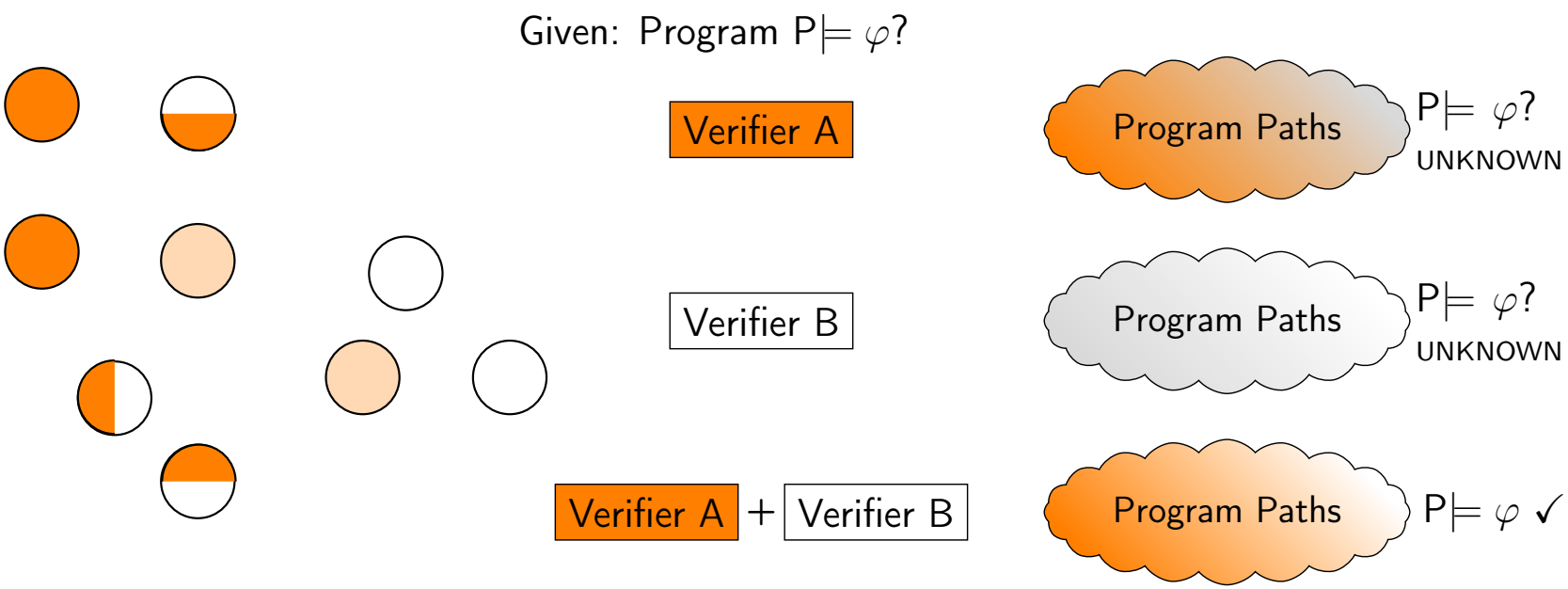
Given: Program P$\models \varphi$?

Verifier A

Verifier B

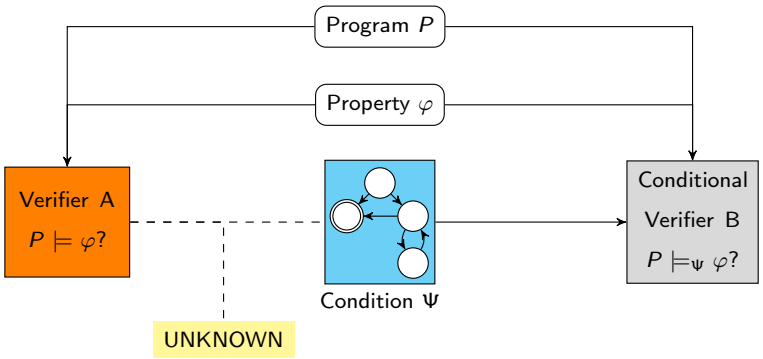


Why Splitting Verification Tasks?
The Efficacy Reason

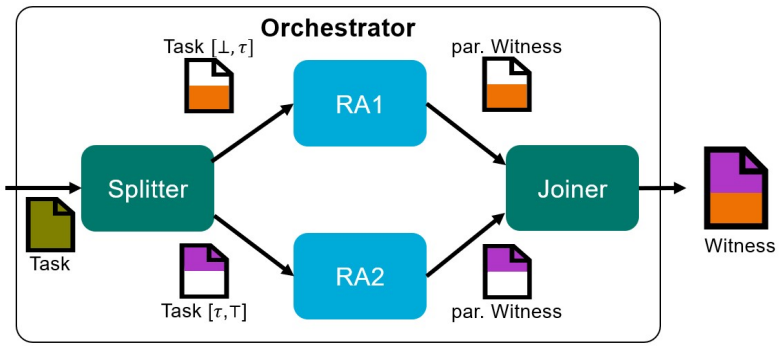


Two Examples for Splitting Approaches

Conditional Model Checking

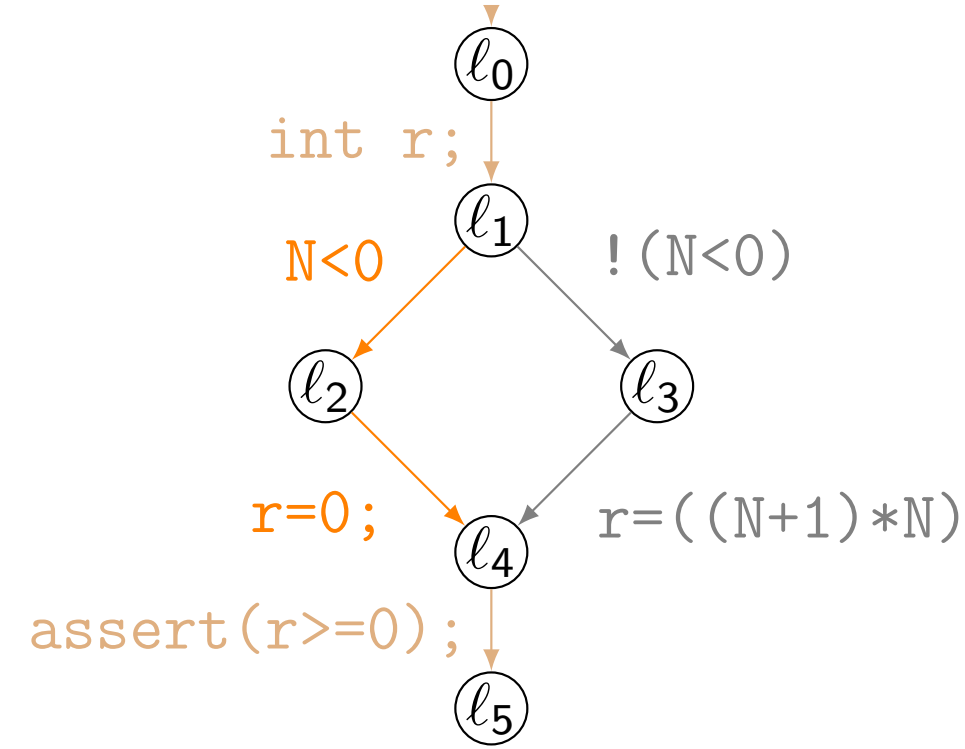


Ranged Analyses

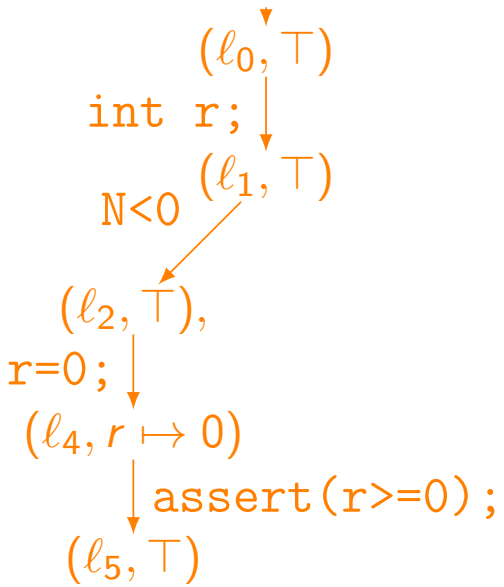


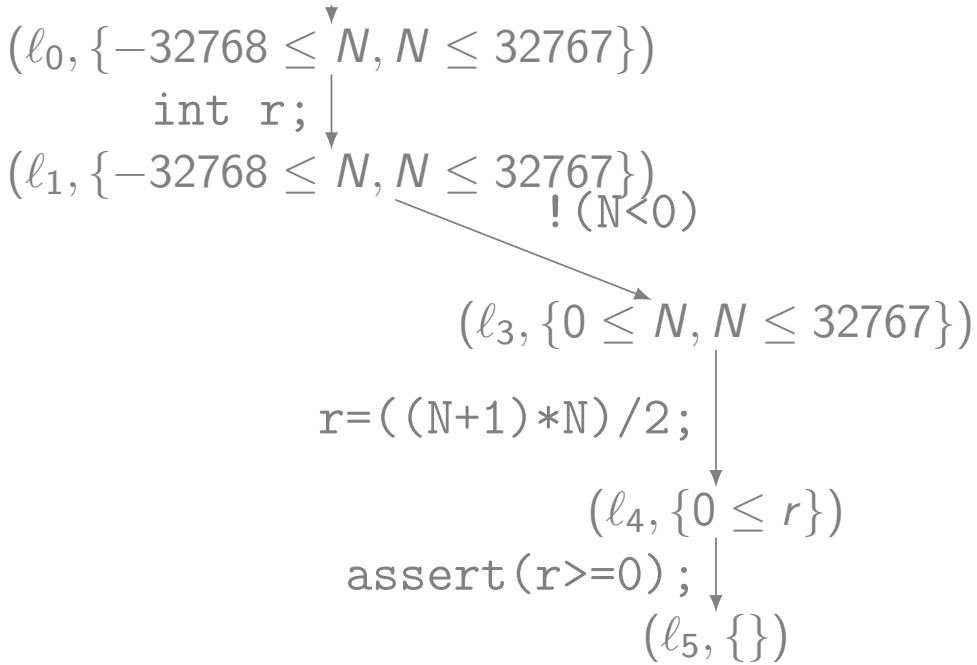
Example Program and Splitting

```
void sum(short N) {
  int r;
  if(N<0)
    r = 0;
  else
    r=((N+1)*N)/2;
  assert(r>=0);
}
```



Separately Analyzing the True and False Branch





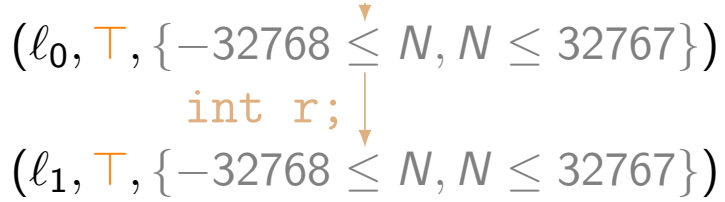
Resulting in two partial abstract reachability graphs (proofs) that cover the complete state space

(Partial) Abstract Reachability Graph

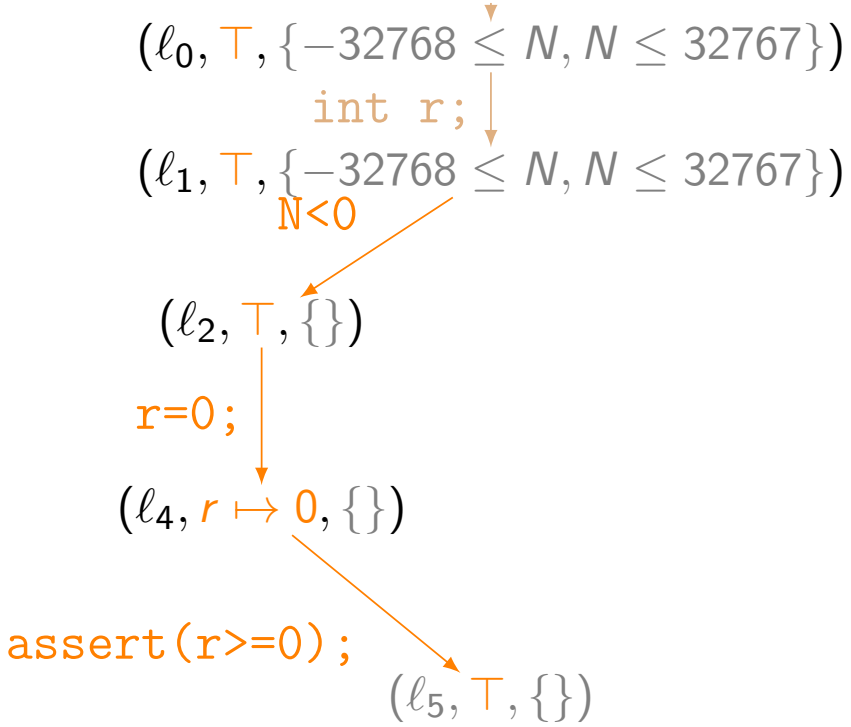
$ARG = (N, G, root)$ where

- nodes N are abstract states
- $G \subseteq N \times G_{\text{prog}}, N$
  $\forall n \in N, g \in G_{\text{prog}}$:
  $(n, g, e) \in \text{absSucc} \implies \text{cover}(e, \{n' \mid (n, g, n') \in G\})$
  $\vee \quad \neg\exists(n, g, \cdot) \in G$
- $root \in N$
  $\{c \in C \mid c(pc) = \ell_0\} \subseteq [\![root]\!]$

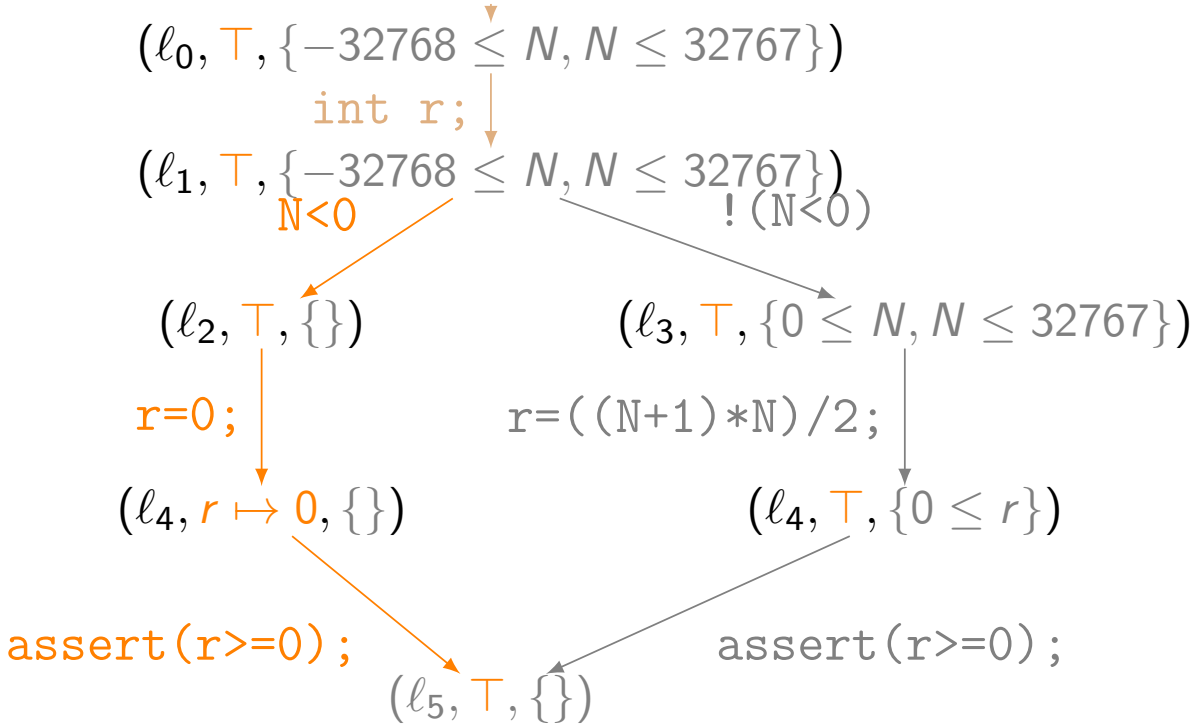
Joining Partial Abstract Reachability Graphs [PART_{PW}, SEFM 2017]



Joining Partial Abstract Reachability Graphs [PART_{PW}, SEFM 2017]



Joining Partial Abstract Reachability Graphs [PART_{PW}, SEFM 2017]



Soundness of ARG Join

Assumption

- Coverage determined on per element basis
- Coverage guaranteed if same element or $\top$ in set

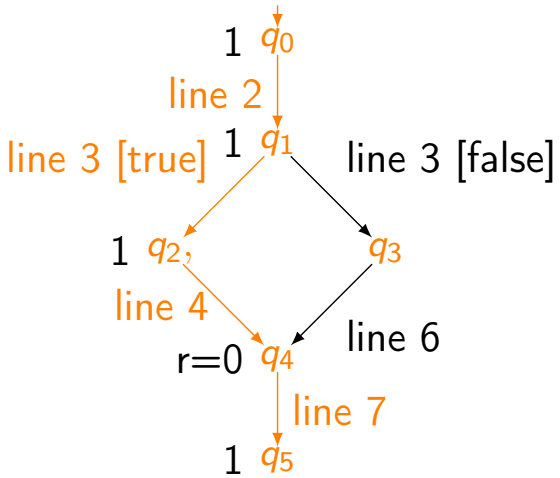
Given

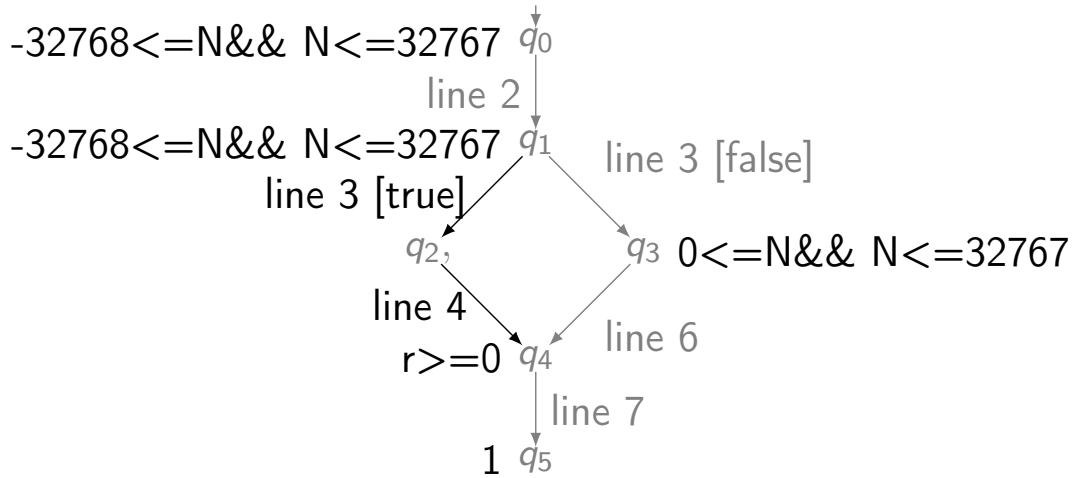
Set of partial ARGs that cover complete program behavior

Guarantee

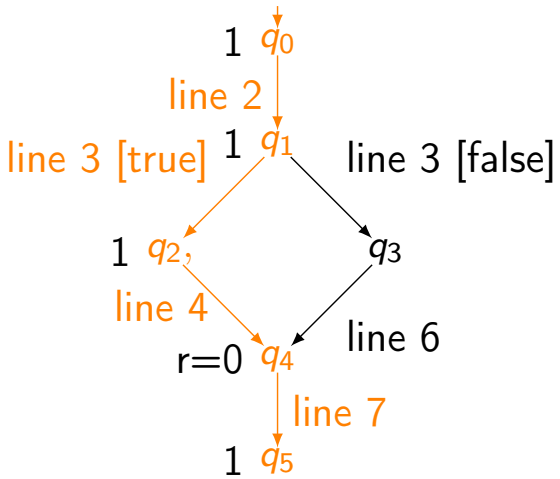
Joining set's ARGs result in complete ARG (proof)

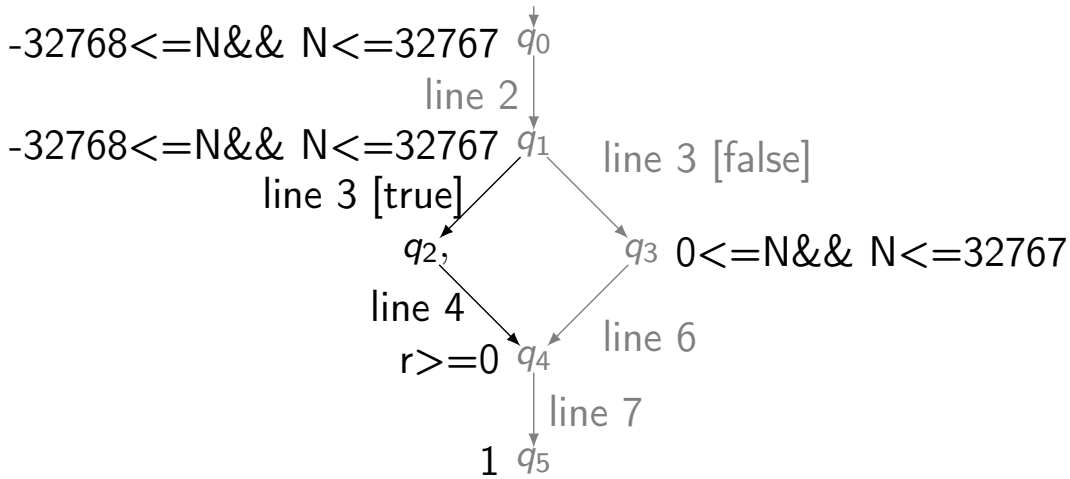
Correctness Witnesses for Separate, Partial Analysis



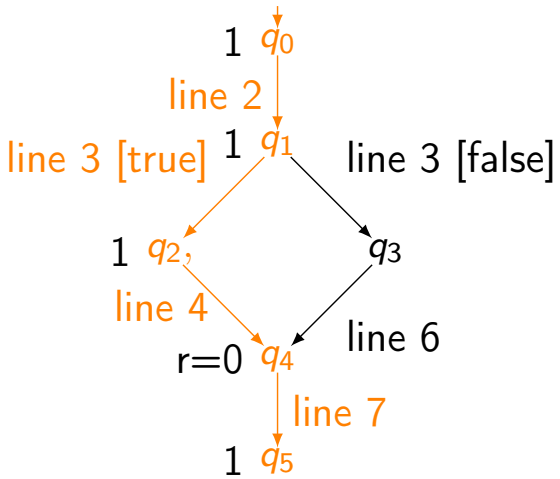


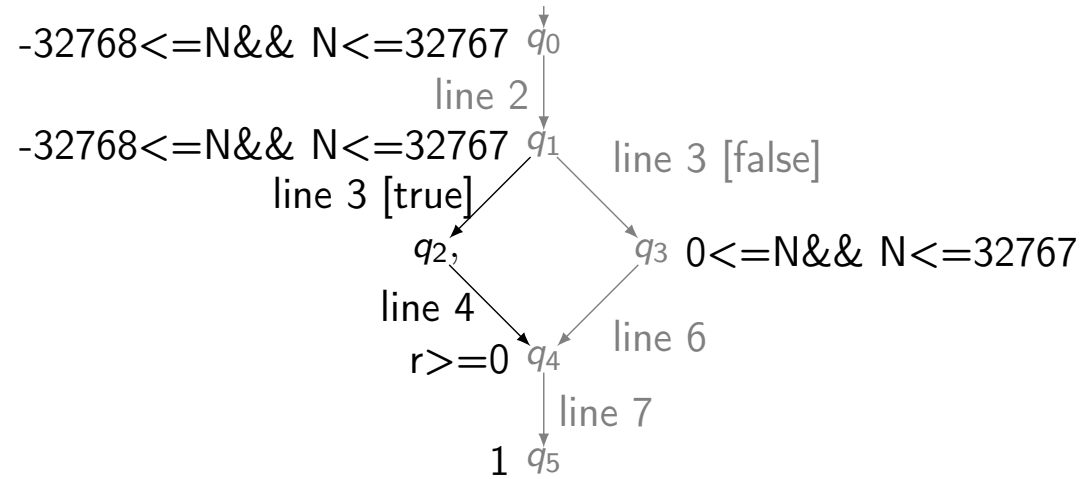
Joining Correctness Witnesses

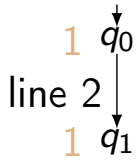




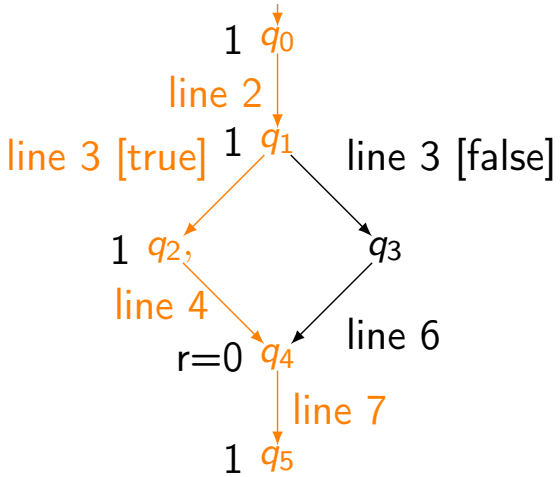
Joining Correctness Witnesses

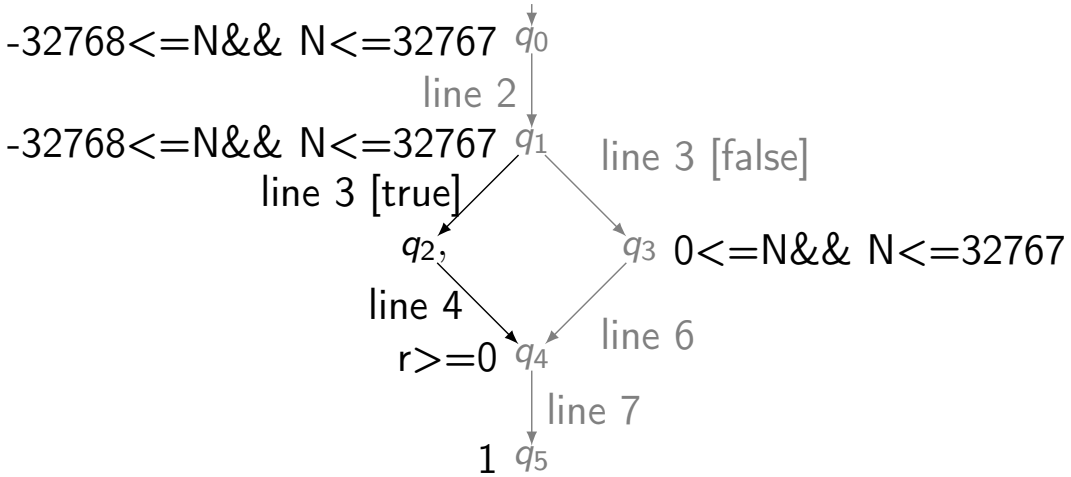


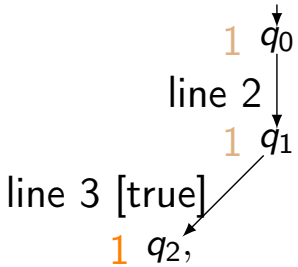




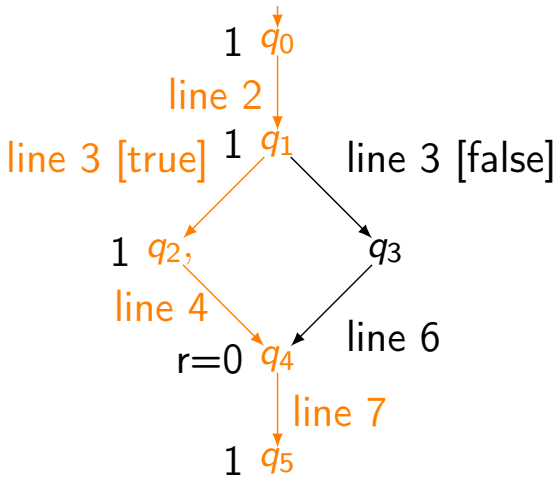
Joining Correctness Witnesses

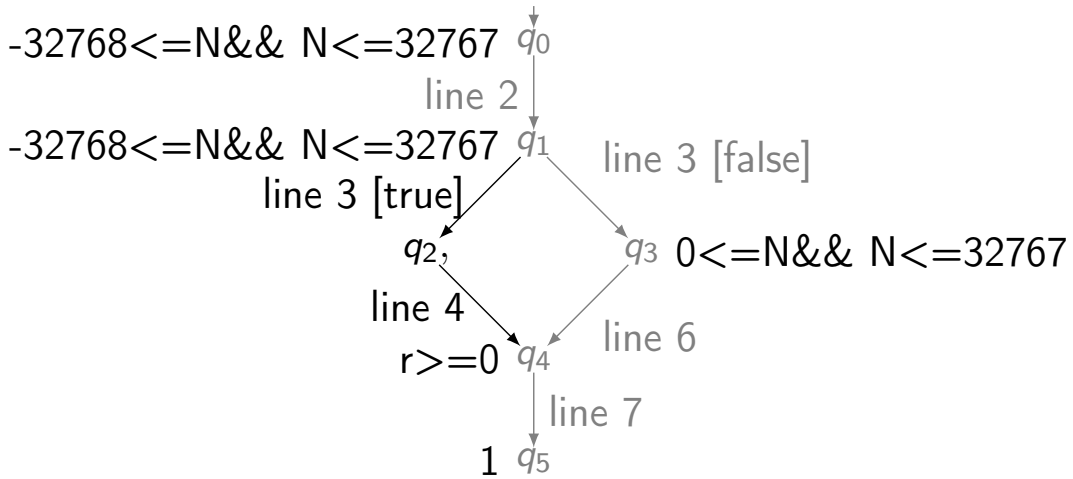


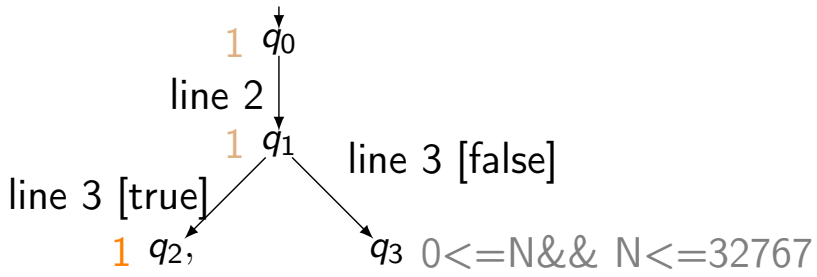




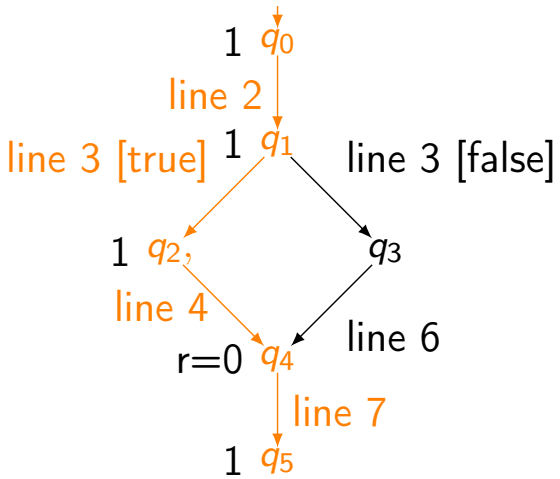
Joining Correctness Witnesses

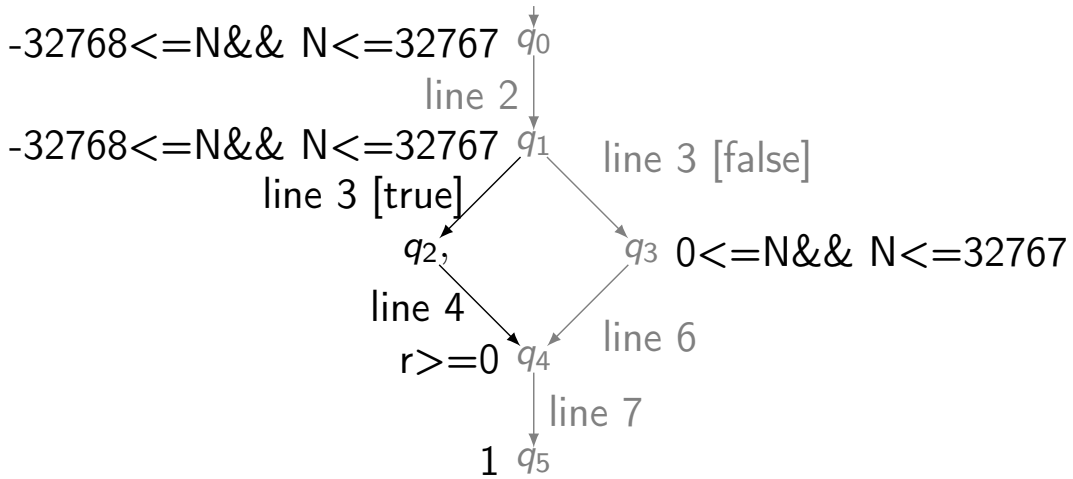


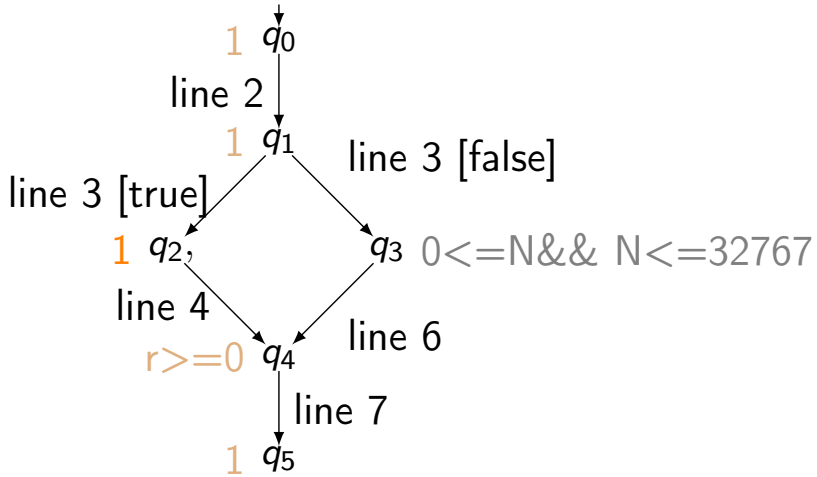




Joining Correctness Witnesses







Challenge – Nature of Partial Witnesses

(Partial) witnesses less structured than ARGs

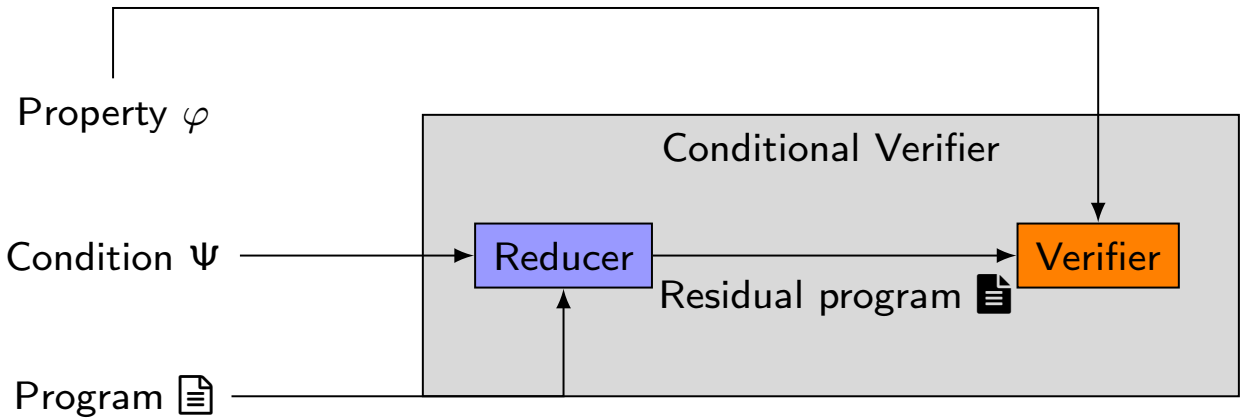
- May or may not fold to CFA, i.e., join information
- May contain non-explored paths
  (since must not restrict state-space)
- Invariant only valid for explored paths
- Not all states have an invariant, often only loop heads
- Invariant true often not given
- Provided information for matching may differ
- Matching to program can be ambiguous

Challenge – Different Tools

- Program representations do not match (incompatibility/ambiguity in combination)
- Provide different information (format, detail level, etc.)
- No consensus in what is a partial proof, what should be contained

Challenge – Program Transformation

Program transformation to make splitting transparent



Consequences for Proof

- Partial proofs for different programs
- Requires proof to be re-transformed to original program

Conclusion

- Cooperative, task splitting approaches become more important
- Use program transformation for broad applicability

Conclusion

- Cooperative, task splitting approaches become more important
- Use program transformation for broad applicability

- Validation of those approaches important (cooperation of different tools with different assumptions)
- Validation of violations straightforward

Conclusion

- Cooperative, task splitting approaches become more important
- Use program transformation for broad applicability

- Validation of those approaches important (cooperation of different tools with different assumptions)
- Validation of violations straightforward

- First approaches for generating proofs
  - try to integrate into existing validation infrastructure
  - make assumptions (i.e., restricted applicability)
  - focus on safety properties
  - too fine-grained
  - still do not match needs