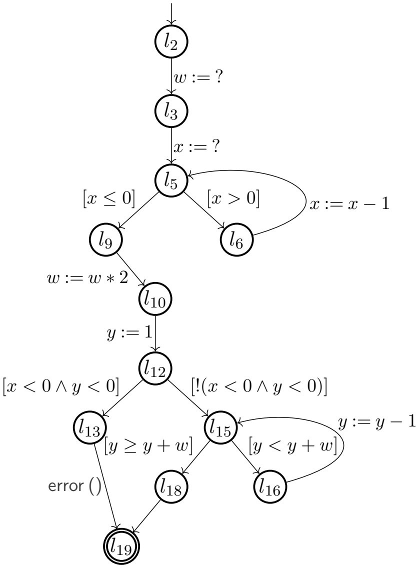
Incremental Slicing
CEGAR + Program Slicing

Thomas Lemberger

LMU Munich, Germany

The Evil Loop Program

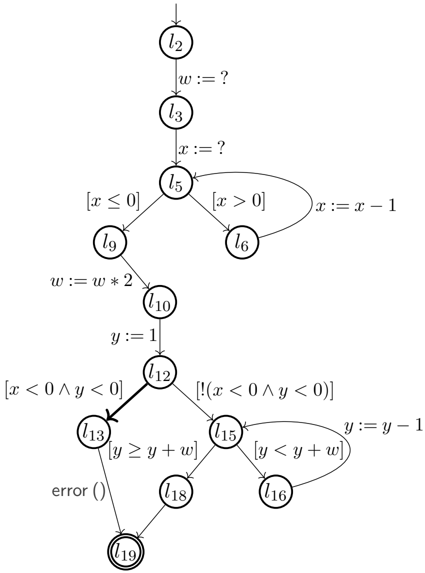
```
int main() {
    int w = ?;
    int x = ?;

    while (x > 0) {
        x--;
    }

    int w = w * 2;
    int y = 1;

    if (x < 0 && y < 0) {
        error();
    } else {
        while (y < y + w) {
            y--;
        }
    }
}
```



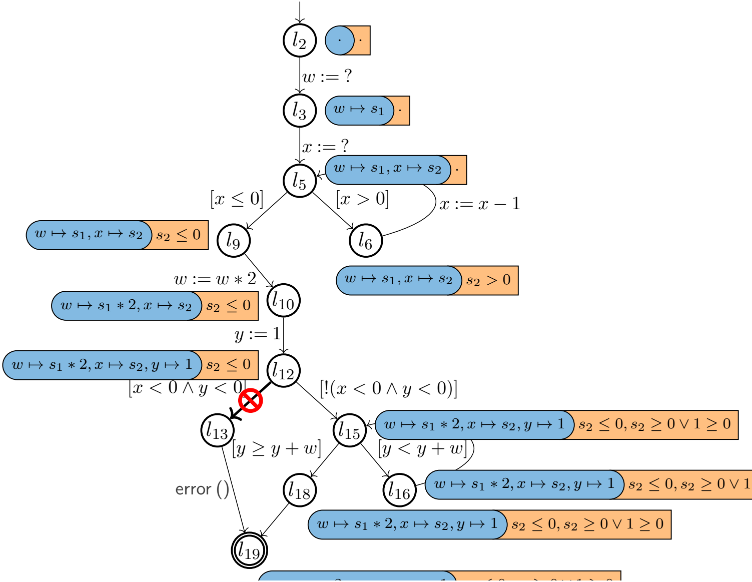
Symbolic Execution

1. Symbolic memory: Stores (symbolic) variable assignments.
   $v : X \mapsto \mathcal{Z} \cup \mathcal{S}$
2. Path constraints: Constrain symbolic values.
   $pc \in 2^{\mathcal{P}}$

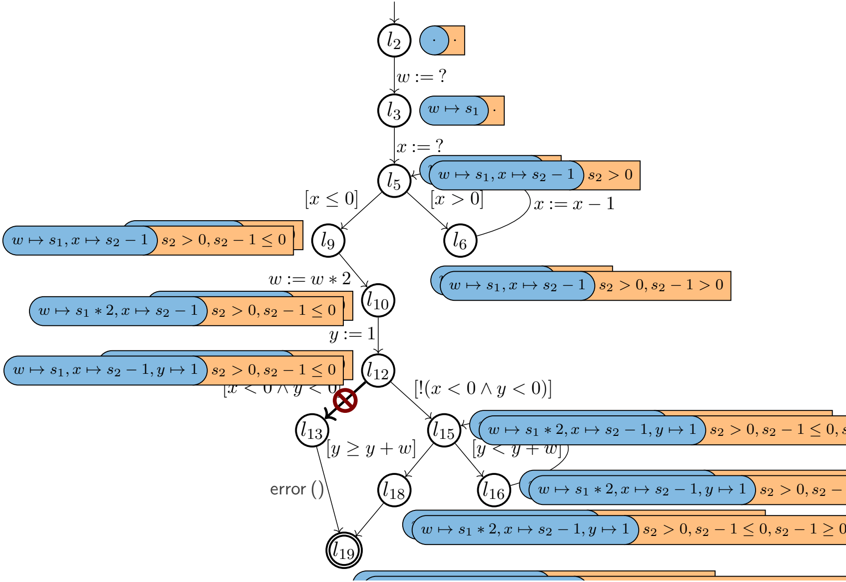
Path Explosion ruins the day



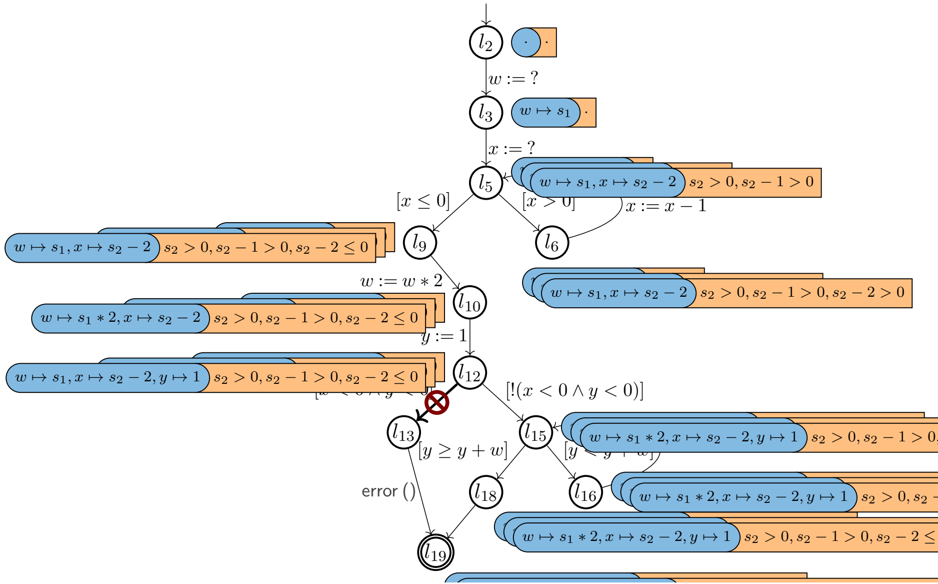
Path Explosion ruins the day



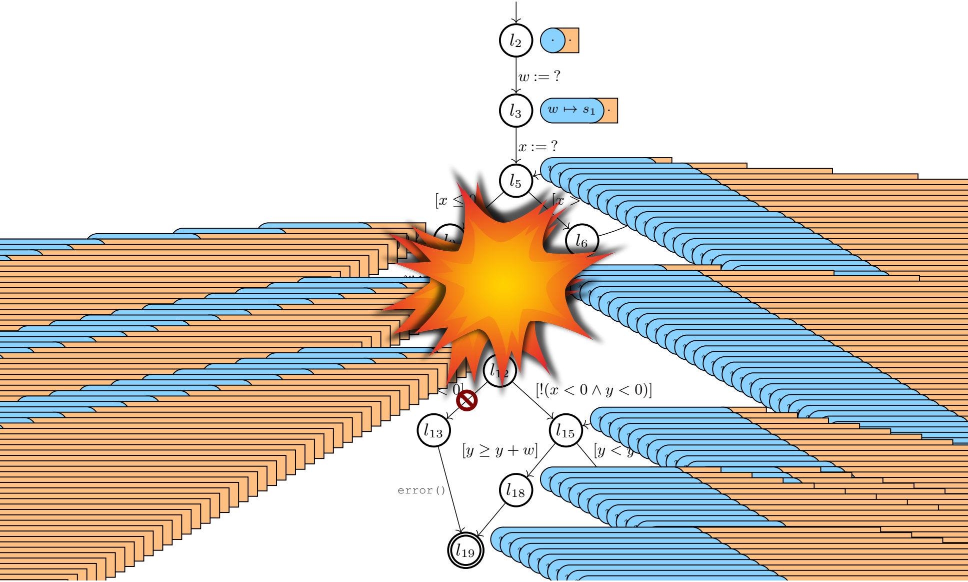
Path Explosion ruins the day



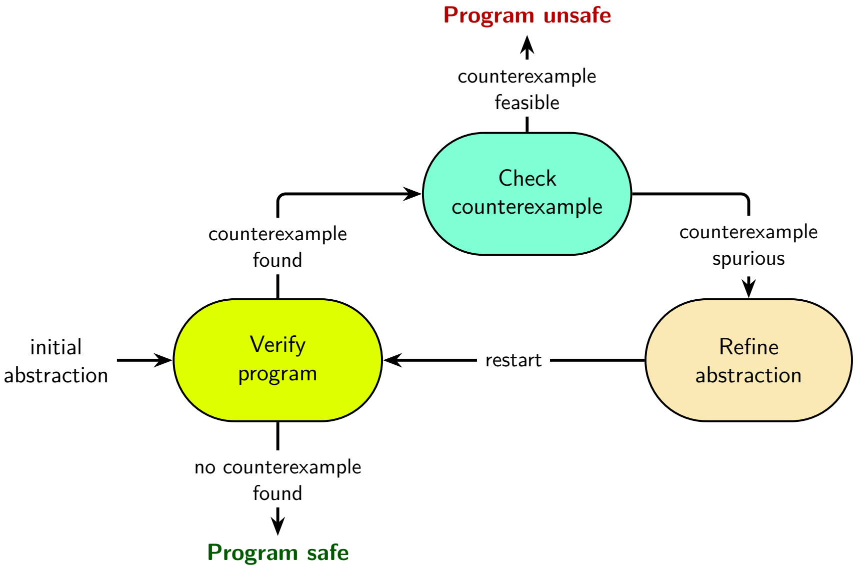
Path Explosion ruins the day



Path Explosion ruins the day



Abstraction with CEGAR

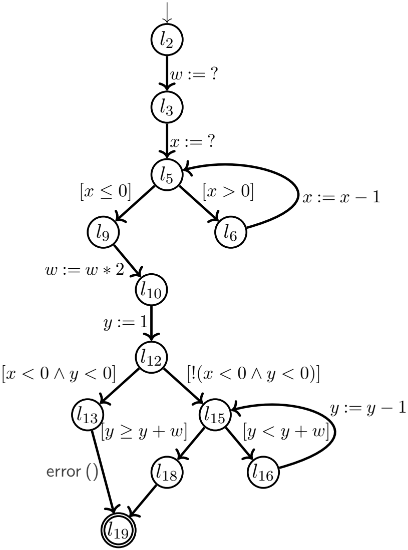


Issues with CEGAR

- CEGAR *only* keeps information that is necessary
- Often many refinements needed to get similar information at different locations
- Solution for some abstract domains: *Scoped precisions*
  - Example explicit-state model checking:
  - Scoped precision needs 7x less refinements than location-based precision
- But: Scoped precision may be too coarse

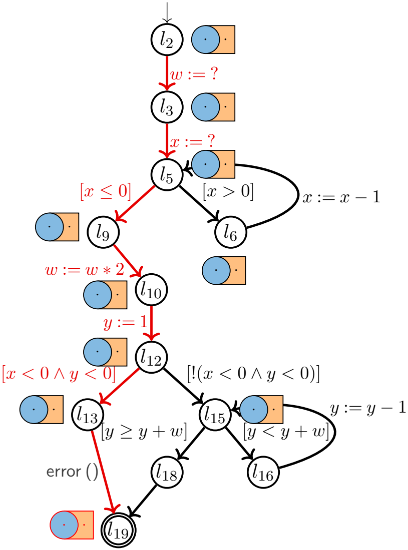
Scoped CEGAR fails

First iteration: $\pi = \emptyset$



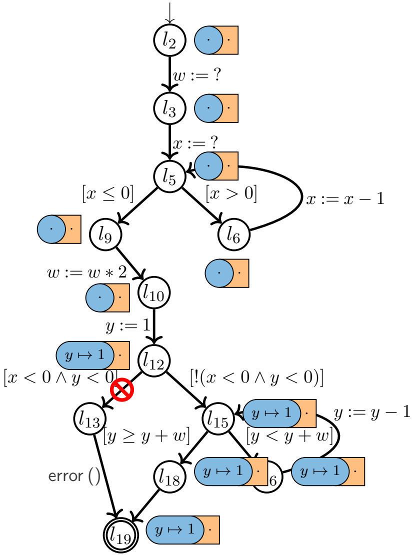
Scoped CEGAR fails

First iteration: $\pi = \emptyset$



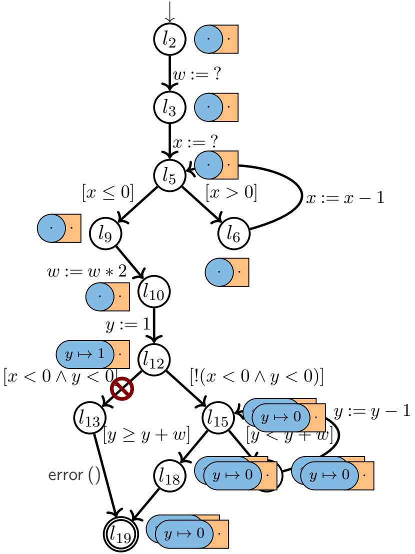
Scoped CEGAR fails

Second iteration: $\pi = \{y\}$



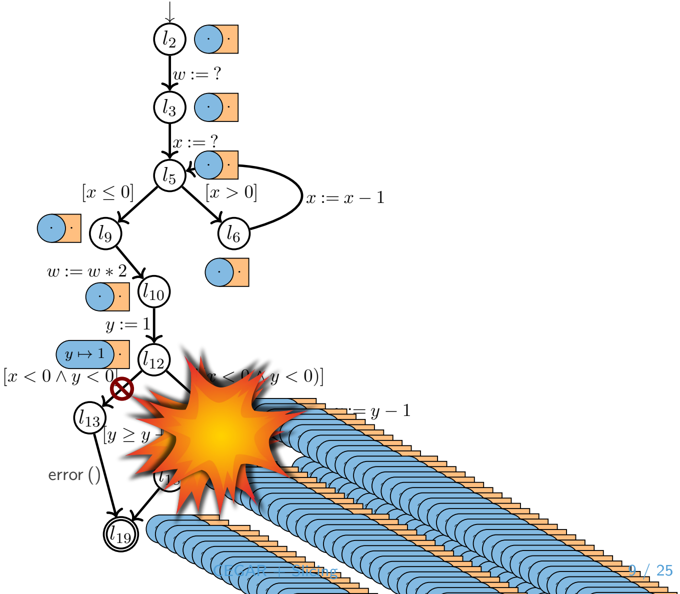
Scoped CEGAR fails

Second iteration: $\pi = \{y\}$



Scoped CEGAR fails

Second iteration: $\pi = \{y\}$



Next Try: Static Slicing

- Goal: Slice $P/g \subseteq P$ behaviorally equivalent to P regarding CFA edge $g = (l, op, l')$
  - Whenever P exits on an input I, P/g also exits on I, and the values of all program variables at l' are equal for P and P/g.
- Procedure: Replace irrelevant CFA edges (l, op, l') with $(l, \texttt{noop}, l')$
- Relevant edges computed through *dependence graph*
  - Data and control dependences

Dependence Graph: Data Dependence

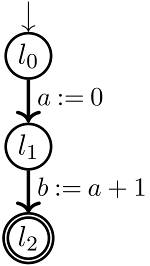
CFA edge $g_2 = (l_2, op_2, l'_2)$ data dependent on edge $g_1 = (l_1, op_1, l'_1)$ for variable $x \in X$, if:

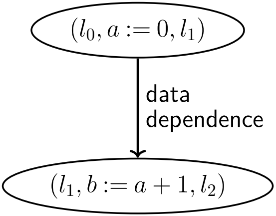
1. g_1 defines x
2. g_2 uses x
3. a path between the two exists without a new definition of x

Dependence Graph: Data Dependence

CFA edge $g_2 = (l_2, op_2, l'_2)$ data dependent on edge $g_1 = (l_1, op_1, l'_1)$ for variable $x \in X$, if:

1. g_1 defines x
2. g_2 uses x
3. a path between the two exists without a new definition of x



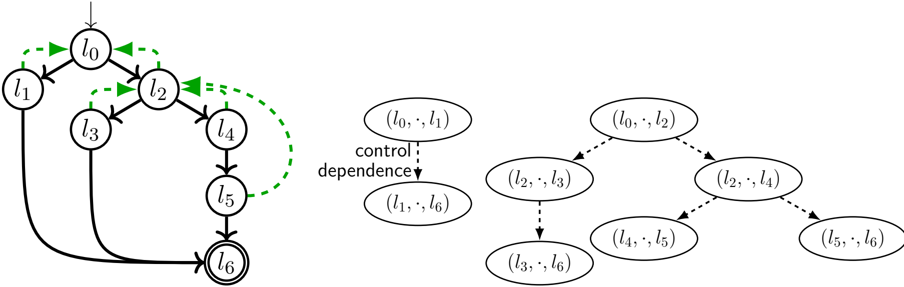


Dependence Graph: Control Dependence

CFA edge g_2 control dependent on edge g_1, if
g_1 is a program branch that must be entered at the time of its encounter to get to g_2.

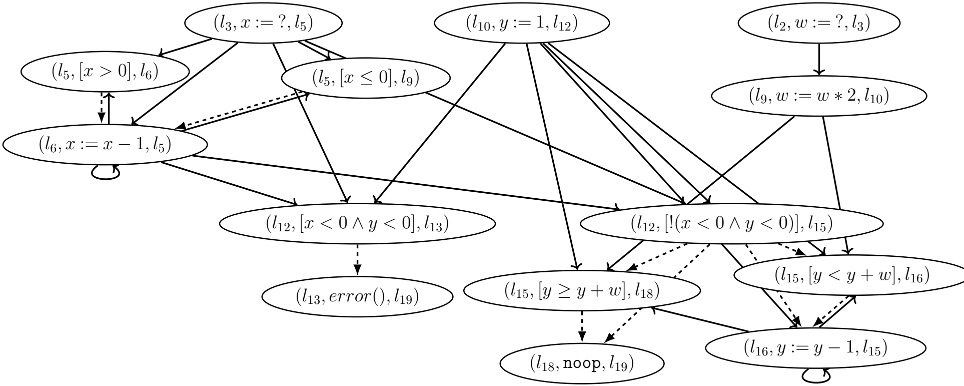
Dependence Graph: Control Dependence

CFA edge g_2 control dependent on edge g_1, if g_1 is a program branch that must be entered at the time of its encounter to get to g_2.



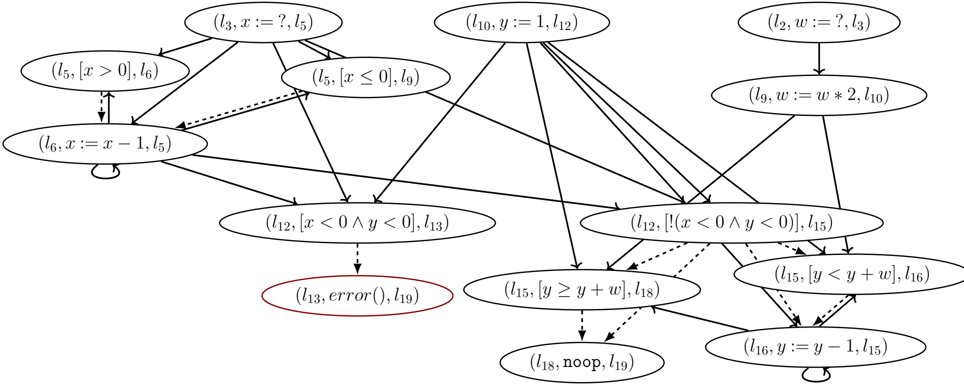
Slice Computation

- Backwards-search through dependence graph, starting at slicing criterion g
- Slice consists of all reachable CFA edges



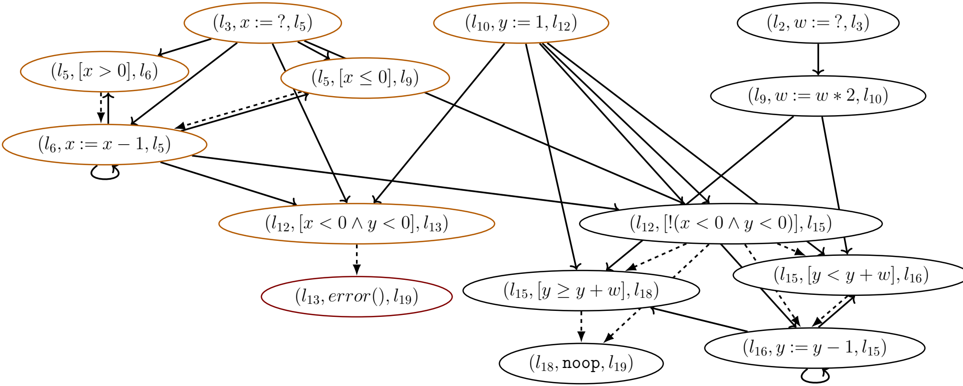
Slice Computation

- Backwards-search through dependence graph, starting at slicing criterion g
- Slice consists of all reachable CFA edges

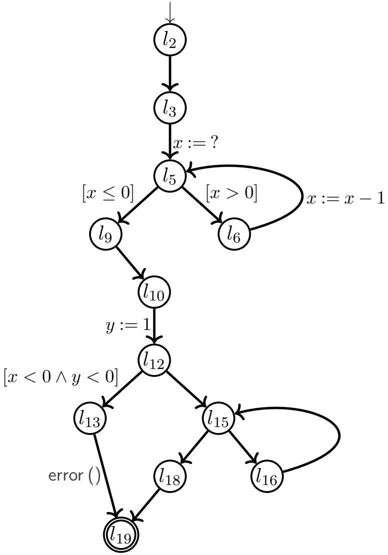


Slice Computation

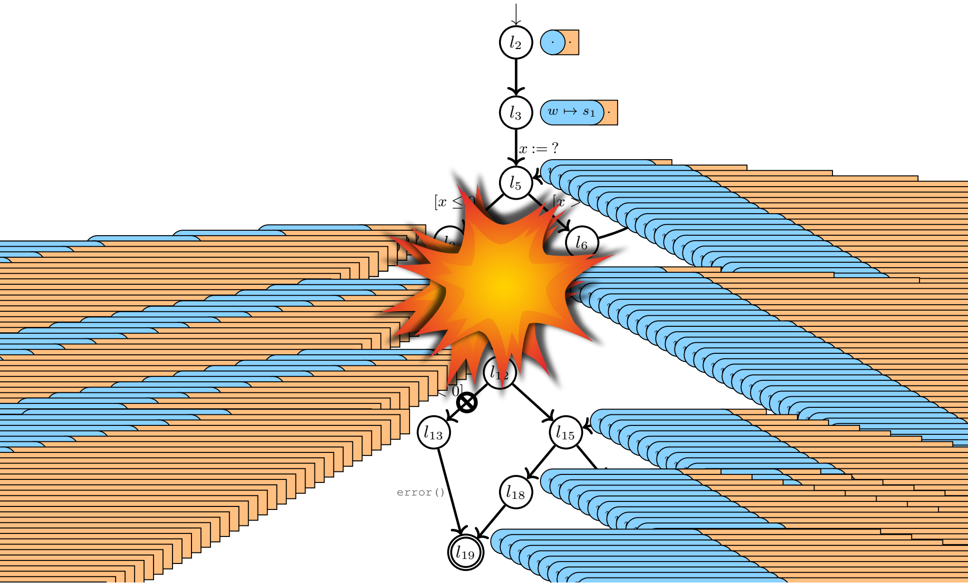
- Backwards-search through dependence graph, starting at slicing criterion g
- Slice consists of all reachable CFA edges



Slicing also fails!



Slicing also fails!

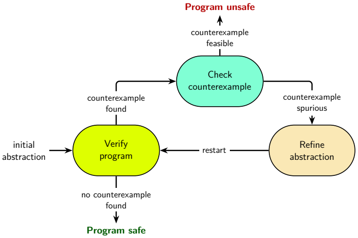


Problem of CEGAR and Slicing

- CEGAR:
  - Location-based precisions too fine-grained
  - Scoped precision too coarse (information could be tracked at unfortunate locations)
- Slicing:
  - Doesn't consider semantics (too coarse)
  - Only done for fixed slicing criteria

Incremental Slicing

- Combine CEGAR and static slicing
- Create dynamic approach to static program slicing
- Consider program operations as they become necessary



Incremental Slicing in CPAchecker

1. Dependence Graph
2. Slicing CPA
3. Slice Refinement

Incremental Slicing in CPAchecker

1. Dependence Graph
2. Slicing CPA
3. Slice Refinement

Slicing CPA

- Reminder: CPA $A = (D, \Pi, \rightsquigarrow, \text{merge}, \text{stop}, \text{prec})$
- Slicing CPA $\mathbb{SC}$ wraps other CPA
- Always runs on original CFA
- Precision represents slice: $\Pi_{\mathbb{SC}} = 2^G$
- $\mathbb{SC}$ modifies CFA edge g before passing it to wrapped CPA

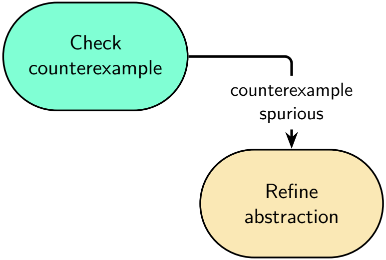
For $g = (l, op, l')$,

$$g' = \begin{cases} g & \text{if } g \in \pi_{\mathbb{SC}} \\ (l, \texttt{noop}, l') & \text{otherwise} \end{cases}$$

and

$$e \overset{g}{\rightsquigarrow}_{\mathbb{SC}} e' \text{ if } e \overset{g'}{\rightsquigarrow} e'$$

Slice Refinement



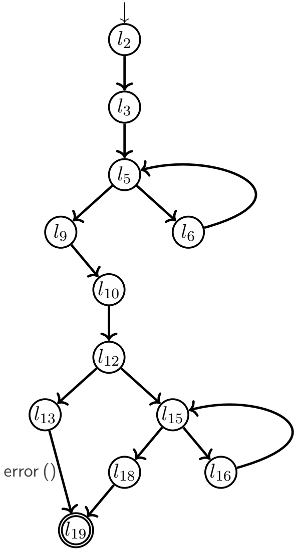
- Counterexample check: Run wrapped CPA on counterexample without slicing
- Abstraction Refinement: $\pi'_{\mathbb{SC}} = \pi_{\mathbb{SC}} \cup (P/g_e)$
- Initial abstraction: $\pi_{\mathbb{SC}_0} = \emptyset$

Combination: Slicing and wrapped CEGAR

- Combination possible: Slicing CPA with CEGAR + wrapped CPA with CEGAR
- Goal: Use scoped precision of wrapped CPA, but avoid easy mistakes through slicing

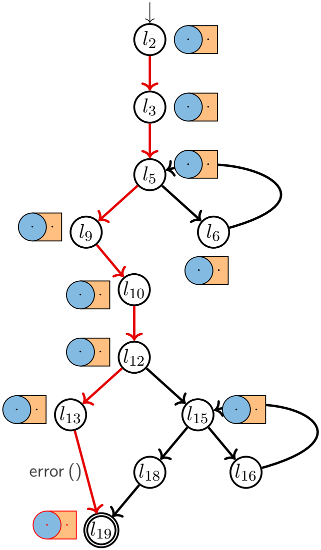
Combination: Slicing and wrapped CEGAR

First iteration: $\pi = \emptyset$



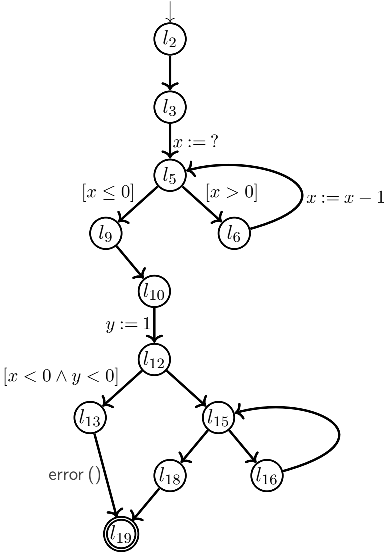
Combination: Slicing and wrapped CEGAR

First iteration: $\pi = \emptyset$



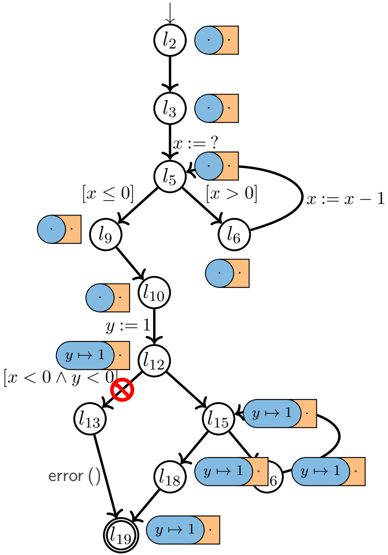
Combination: Slicing and wrapped CEGAR

Second iteration: $\pi = \{y\}$



Combination: Slicing and wrapped CEGAR

Second iteration: $\pi = \{y\}$



Evaluation

Table: Results over ReachSafety + LDV (5 590 tasks)

	SymEx	SymEx_{S}	SymEx_{C}	$\text{SymEx}_{\text{S+C}}$
Correct	726	1398	**2398**	2000
Correct proof	190	1147	**2108**	1794
Correct alarm	**536**	251	290	206

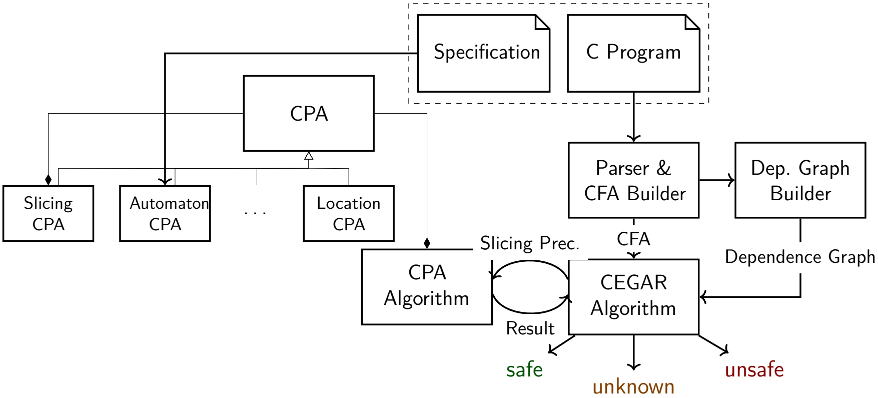
At the current state +17 correct proofs and +19 correct alarms

Conclusion

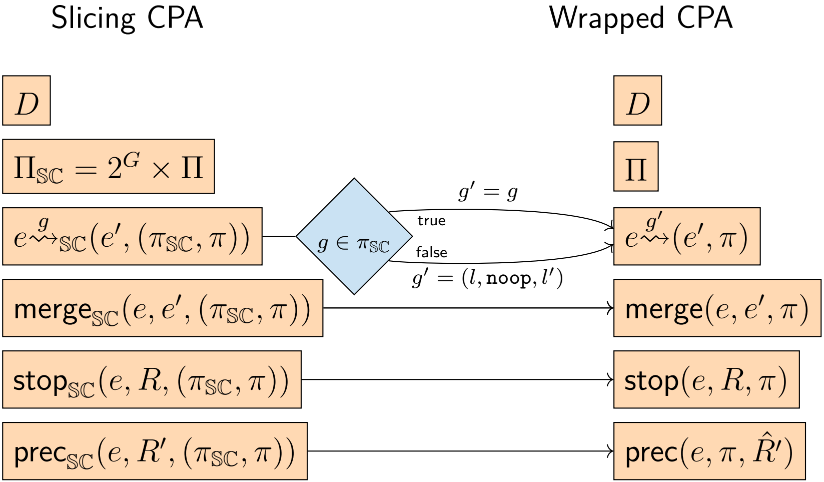
- Incremental slicing not competitive to CEGAR (yet?)
- Program slicing can complement CEGAR-approaches
- Flexible framework for dependence graph computation/slicing

Appendix

Incremental Slicing in CPAchecker



Slicing CPA Definition



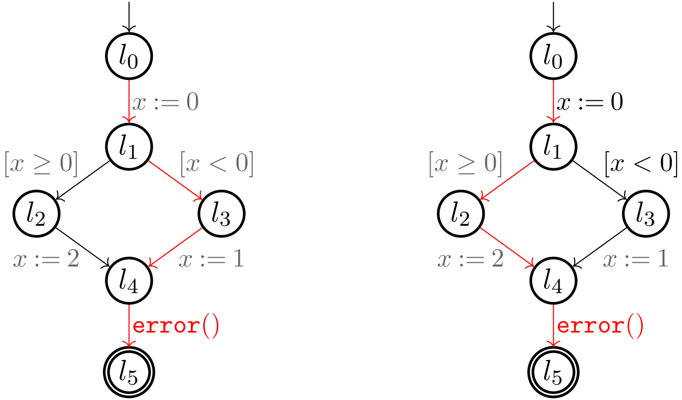
Details: Dependence Graph Construction

1. Data Dependence Computation:
   - Reaching Definitions + Variable Uses
   - Abstract State: Data Dependences + Reaching Definitions per State

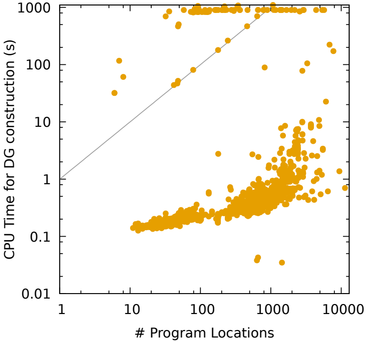
     (S, d) with $S \subseteq X \times G$, set G of CFA edges, and $d \subseteq G \times X \times G$
   - Example: $(g_1, x, g_2) \Rightarrow g_1$ is, through use of x, data dependent on g_2.
2. Control Dependence Computation:
   - Post-Dominator Computation: Backwards-run of dominator computation (pseudo-CPA)
   - Abstract State: Set of candidate dominator locations for corresponding program location

     $D \subseteq L$ with set L of program locations
   - Pseudo-merge when control flow meets: $\delta \sqcup_D \delta' = \delta \cap \delta'$
   - Transfer relation: $\delta \overset{g}{\rightsquigarrow} \delta \cup \{l\}$ for $g = (l, \cdot, l')$

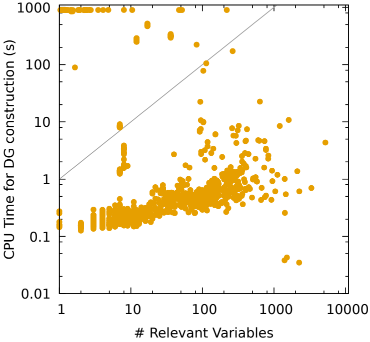
Why we need target-path slicing



Dependence Graph Construction Time Correlation



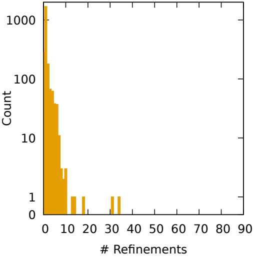
(a) Number of program locations to construction time



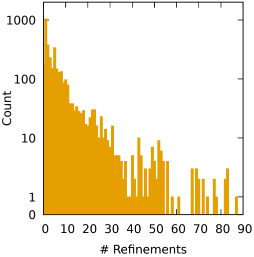
(b) Number of relevant variables to construction time

Figure: Indicators for the CPU time required for dependence graph construction

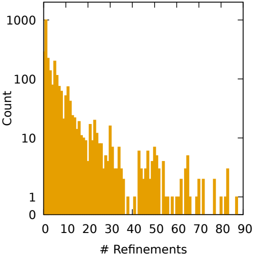
Number of Refinements Performed



(a) SYMEX$_{S}$



(b) SYMEX$_{C}$



(c) SYMEX$_{S+C}$