Automatic Detection of Vulnerable Variables for CTL Properties of Programs

MOUSSAOUI REMIL Naïm, URBAN Caterina, MINÉ Antoine



November 18, 2024

LPAR 2024

Problem statement

```
1  void main(int x, int y, int z){
2      while(y > z){
3          y = y - x;
4      }
5      ERROR()
6  }
```

Problem statement

```
1        void main(int x, int y, int z){
2            while(y > z){
3                y = y - x;
4            }
5            ERROR()
6        }
```

How many **variable(s)** an attacker needs to control to reach the error ?

Problem statement

```
1        void main(int x, int y, int z){
2            while(y > z){
3                y = y - x;
4            }
5            ERROR()
6        }
```

How many **variable(s)** an attacker needs to control to reach the error ?

Trivial coarse response: all variables

Problem statement

```
1  void main(int x, int y, int z){
2      while(y > z){
3          y = y - x;
4      }
5      ERROR()
6  }
```

How many **variable(s)** an attacker needs to control to reach the error ?

Trivial coarse response: all variables

Our analysis:

- reduce as possible the sets of variable(s) to control
- numeric constraints on these variables to ensure the error

```
1        void main(int x, int y, int z){
2            while(y > z){
3                y = y - x;
4            }
5            ERROR()
6        }
```

Two possible minimal sets:

- A: $\{y, z\}$ + condition $\{z \geq y\}$

```
1  void main(int x, int y, int z){
2      while(y > z){
3          y = y - x;
4      }
5      ERROR()
6  }
```

Two possible minimal sets:

- A: $\{y, z\}$ + condition $\{z \geq y\} \Longleftarrow$ trivial condition (loop condition)

```
1  void main(int x, int y, int z){
2      while(y > z){
3          y = y - x;
4      }
5      ERROR()
6  }
```

Two possible minimal sets:

- A: $\{y, z\}$ + condition $\{z \geq y\} \Longleftarrow$ trivial condition (loop condition)
- B: $\{x\}$ + condition $\{x \geq 1\}$

```
1  void main(int x, int y, int z){
2      while(y > z){
3          y = y - x;
4      }
5      ERROR()
6  }
```

Two possible minimal sets:

- A: $\{y, z\}$ + condition $\{z \geq y\} \Longleftarrow$ trivial condition (loop condition)
- B: $\{x\}$ + condition $\{x \geq 1\} \Longleftarrow$ **non-trivial** condition (semantics)

Error as a CTL property

```
1  void main(int x, int y, int z){
2      while(y > z){
3          y = y - x;
4      }
5      ERROR()
6  }
```

$$\mathrm{AF}\{l_5 : \mathsf{true}\} \in CTL$$

Error as a CTL property

```
1  void main(int x, int y, int z){
2      while(y > z){
3          y = y - x;
4      }
5      ERROR()
6  }
```

$\mathrm{AF}\{l_5 : \mathsf{true}\} \in CTL$

CTL properties

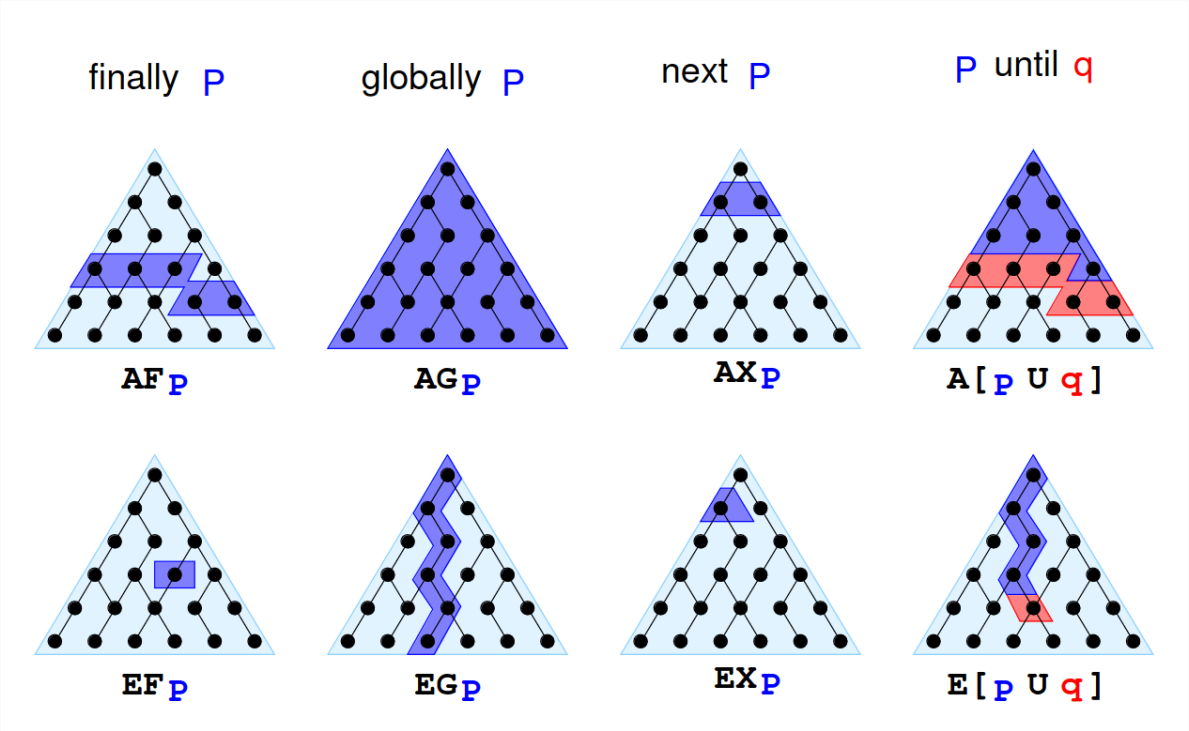
CTL: A branching time logic to express program (trace) properties

$$
\begin{aligned}
\phi ::= & \, l \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \\
& \mid \mathrm{AX}\phi \mid \mathrm{AG}\phi \mid \mathrm{AF}\phi \mid \mathrm{A}\{\phi \,\mathrm{U}\, \phi\} \\
& \mid \mathrm{EX}\phi \mid \mathrm{EG}\phi \mid \mathrm{EF}\phi \mid \mathrm{E}\{\phi \,\mathrm{U}\, \phi\}
\end{aligned}
$$

CTL properties

CTL: A branching time logic to express program (trace) properties

$$\phi ::= l \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi$$
$$\mid \mathrm{AX}\phi \mid \mathrm{AG}\phi \mid \mathrm{AF}\phi \mid \mathrm{A}\{\phi\,\mathrm{U}\,\phi\}$$
$$\mid \mathrm{EX}\phi \mid \mathrm{EG}\phi \mid \mathrm{EF}\phi \mid \mathrm{E}\{\phi\,\mathrm{U}\,\phi\}$$



Abstract interpretation for CTL properties

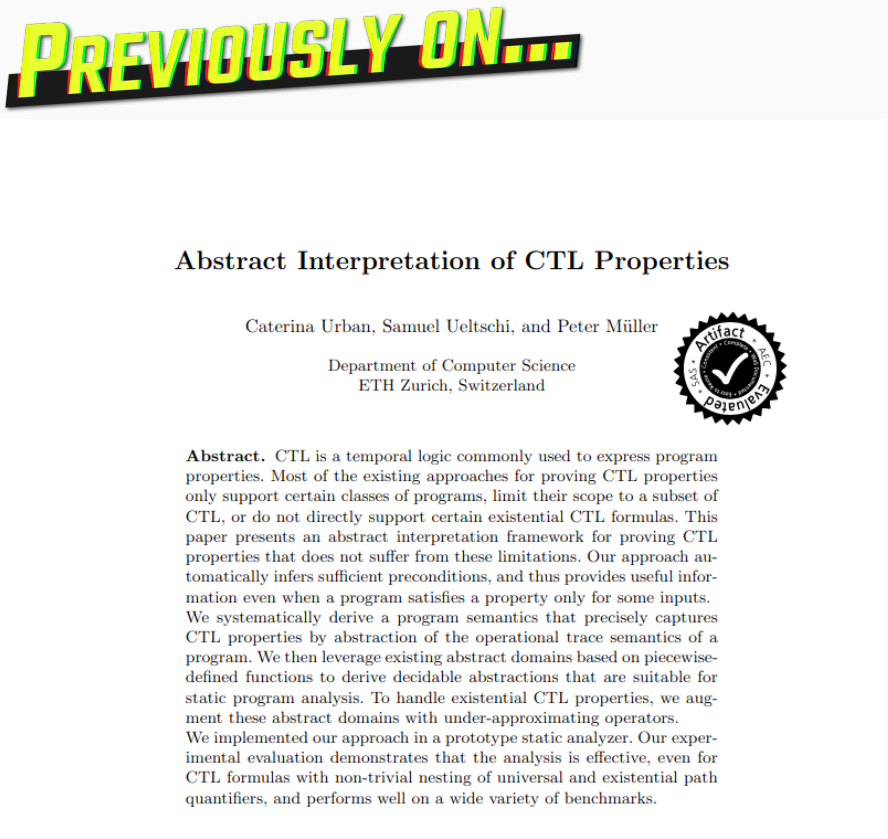
Abstract Interpretation of CTL Properties

Caterina Urban, Samuel Ueltschi, and Peter Müller

Department of Computer Science
ETH Zurich, Switzerland

Abstract. CTL is a temporal logic commonly used to express program properties. Most of the existing approaches for proving CTL properties only support certain classes of programs, limit their scope to a subset of CTL, or do not directly support certain existential CTL formulas. This paper presents an abstract interpretation framework for proving CTL properties that does not suffer from these limitations. Our approach automatically infers sufficient preconditions, and thus provides useful information even when a program satisfies a property only for some inputs.
We systematically derive a program semantics that precisely captures CTL properties by abstraction of the operational trace semantics of a program. We then leverage existing abstract domains based on piecewise-defined functions to derive decidable abstractions that are suitable for static program analysis. To handle existential CTL properties, we augment these abstract domains with under-approximating operators.
We implemented our approach in a prototype static analyzer. Our experimental evaluation demonstrates that the analysis is effective, even for CTL formulas with non-trivial nesting of universal and existential path quantifiers, and performs well on a wide variety of benchmarks.

Abstract interpretation for CTL properties


PREVIOUSLY ON...

Abstract Interpretation of CTL Properties

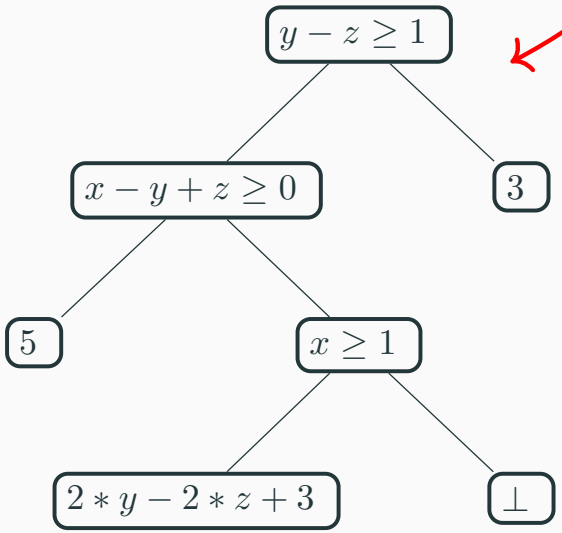
Caterina Urban, Samuel Ueltschi, and Peter Müller

Department of Computer Science
ETH Zurich, Switzerland

Abstract. CTL is a temporal logic commonly used to express program properties. Most of the existing approaches for proving CTL properties only support certain classes of programs, limit their scope to a subset of CTL, or do not directly support certain existential CTL formulas. This paper presents an abstract interpretation framework for proving CTL properties that does not suffer from these limitations. Our approach automatically infers sufficient preconditions, and thus provides useful information even when a program satisfies a property only for some inputs.
We systematically derive a program semantics that precisely captures CTL properties by abstraction of the operational trace semantics of a program. We then leverage existing abstract domains based on piecewise-defined functions to derive decidable abstractions that are suitable for static program analysis. To handle existential CTL properties, we augment these abstract domains with under-approximating operators.
We implemented our approach in a prototype static analyzer. Our experimental evaluation demonstrates that the analysis is effective, even for CTL formulas with non-trivial nesting of universal and existential path quantifiers, and performs well on a wide variety of benchmarks.

- Abstract **sound** semantics: decision tree representing piecewise defined function
- Provide efficient algorithms for a static analysis of C programs

```
1  void main(int x, int y, int z){
2     while(y > z){
3        y = y - x;
4     }
5     ERROR()
6  }
```

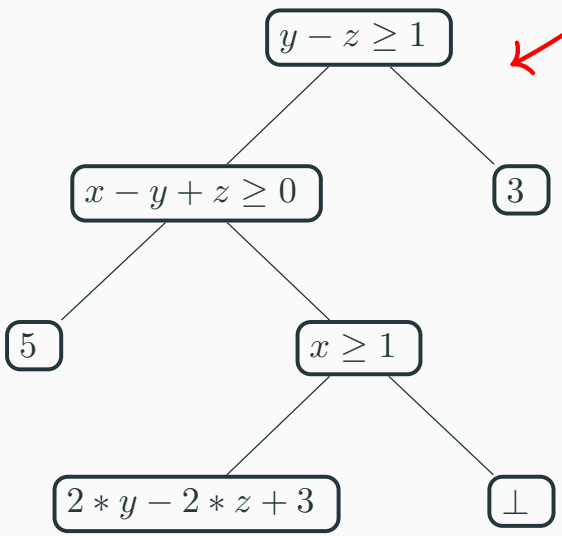


abstracted in

represents

$$f(x,y,z) = \begin{cases} 3 & y < z \\ 5 & x - y + z \geq 0 \\ 2 * y - 2 * z + 3 & x \geq 1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

```
1  void main(int x, int y, int z){
2     while(y > z){
3        y = y - x;
4     }
5     ERROR()
6  }
```



abstracted in

represents

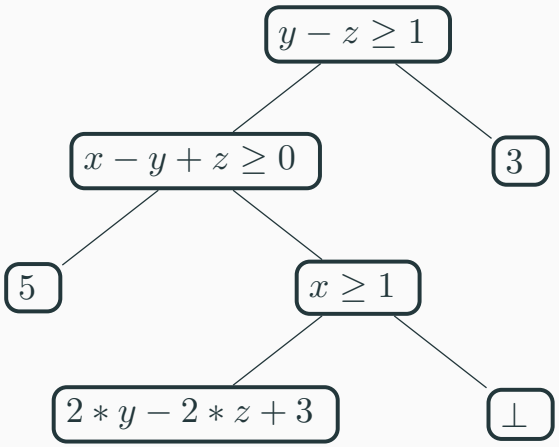
$$f(x,y,z) = \begin{cases} 3 & y < z \\ 5 & x - y + z \geq 0 \\ 2 * y - 2 * z + 3 & x \geq 1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

Theorem

Given a decision tree $\mathcal{T}_\phi$ for a CTL property ϕ, if $s \in dom(\mathcal{T}_\phi)$ then $s \models \phi$.

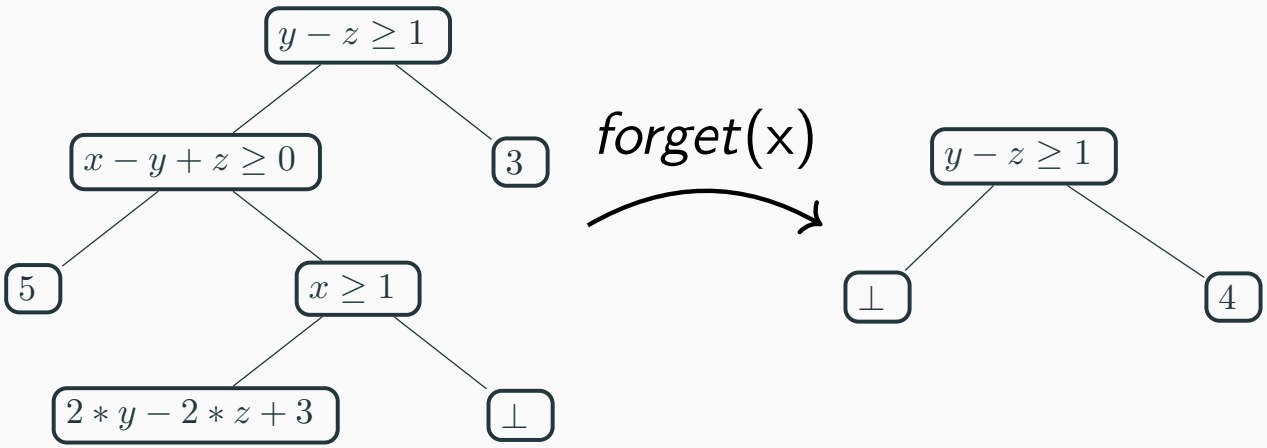
Abstract semantics

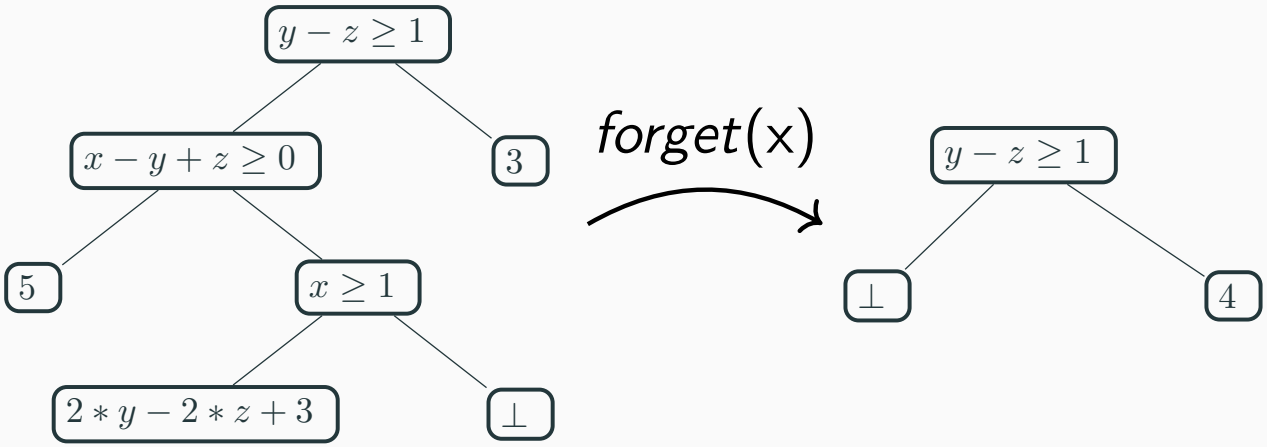
forget : given a decision tree and a variable X try to build a tree by removing all the constraints on X.

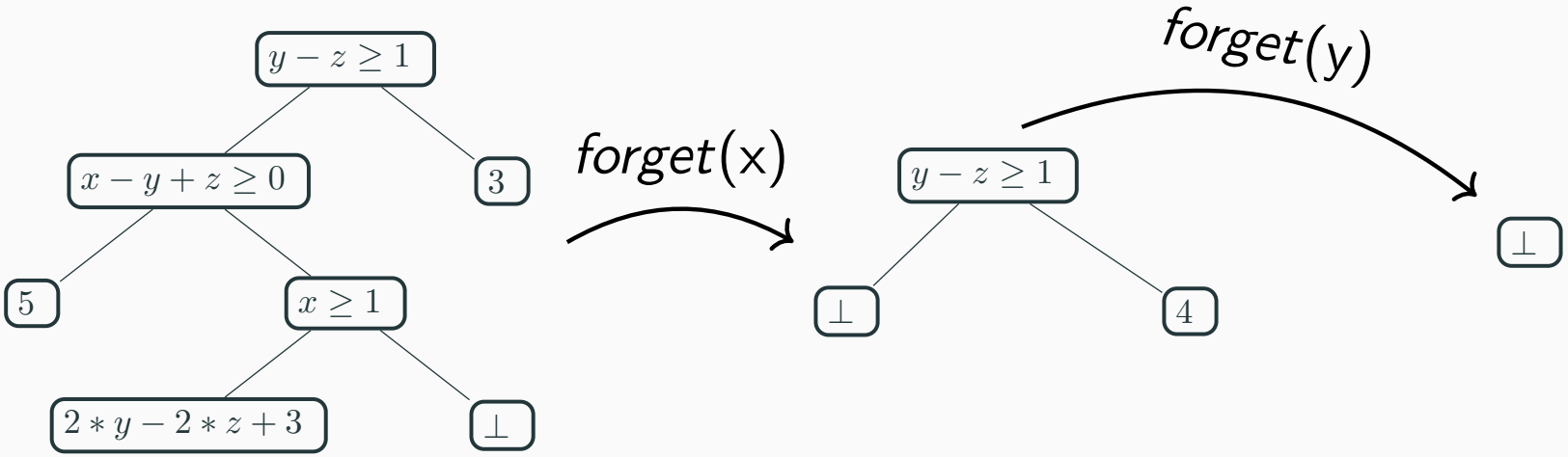


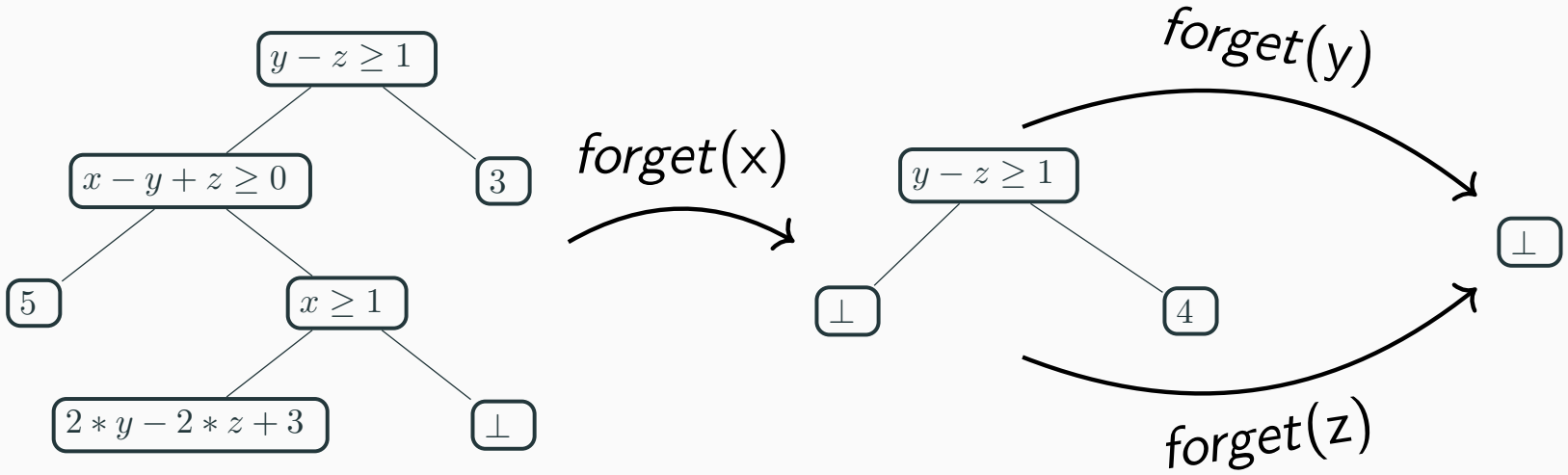
Abstract semantics

forget : given a decision tree and a variable X try to build a tree by removing all the constraints on X.




$y - z \geq 1$
$x - y + z \geq 0$
3
5
$x \geq 1$
$2 * y - 2 * z + 3$
$\perp$
forget(x)
$y - z \geq 1$
$\perp$
4

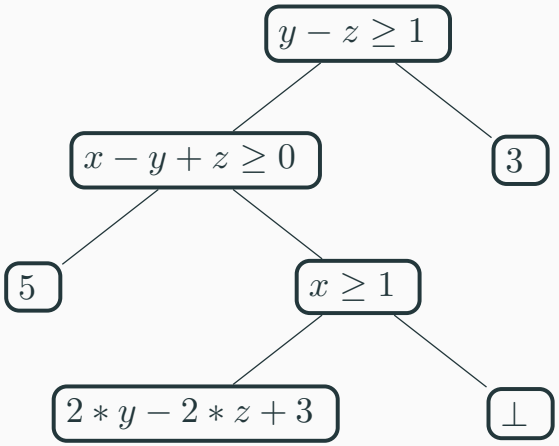

$y - z \geq 1$
$x - y + z \geq 0$
3
5
$x \geq 1$
$2 * y - 2 * z + 3$
$\bot$
forget(x)
$y - z \geq 1$
$\bot$
4
forget(y)
$\bot$

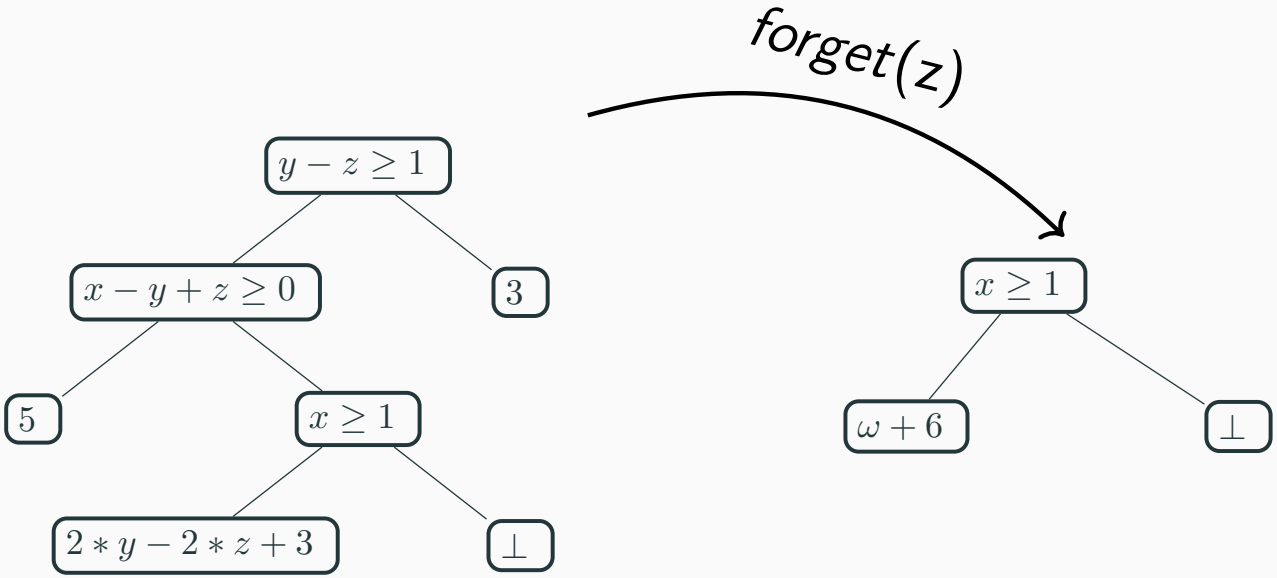


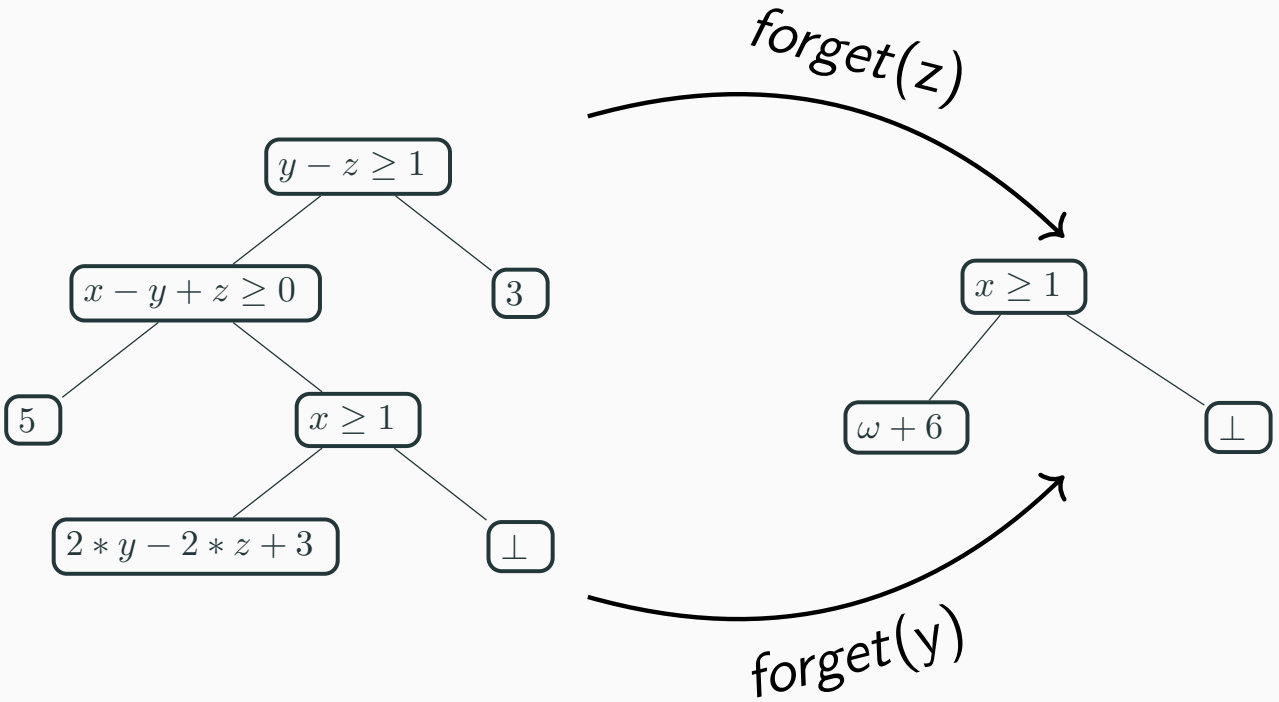
$\{x\}$ is safe: maximal set of variable that can be removed

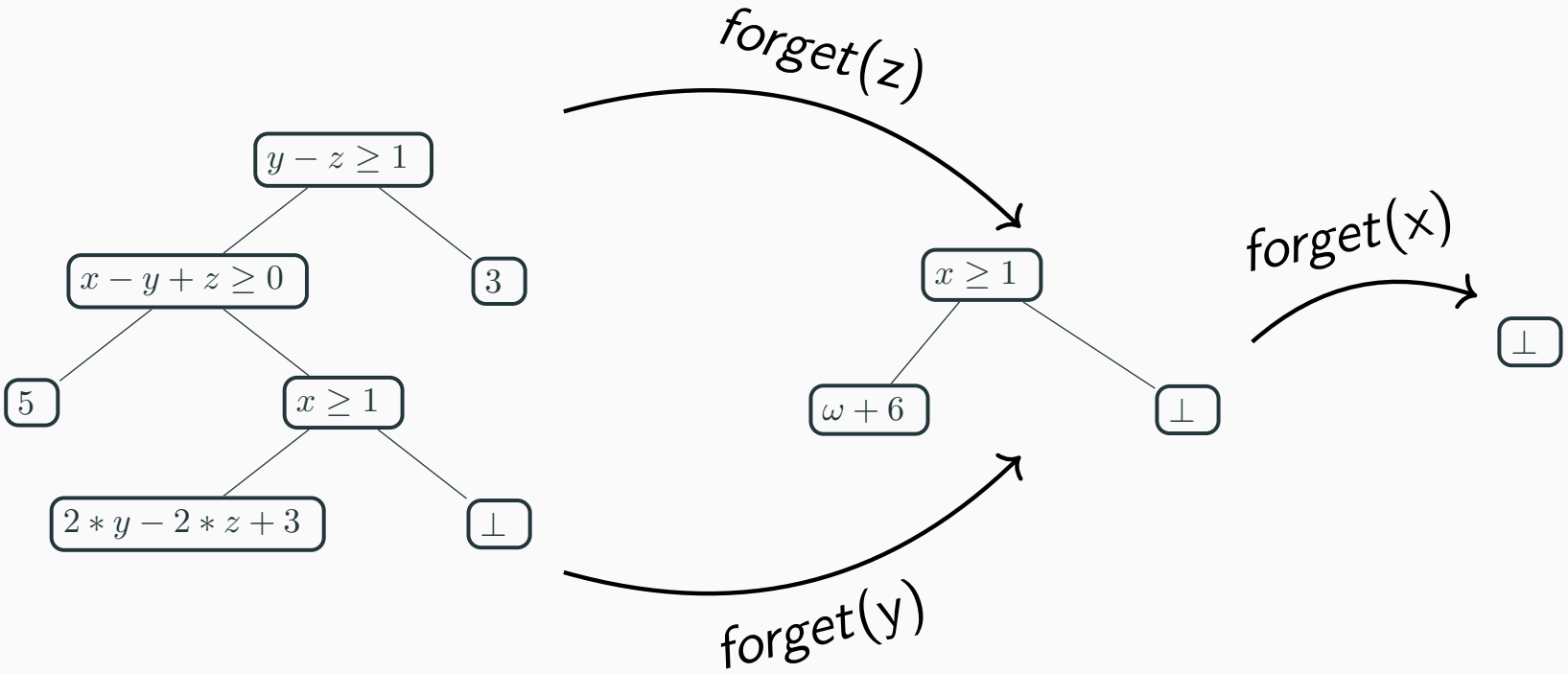
$\{y, z\}$ is vulnerable: Complement of maximal safe set

$\{y, z\}$ is a first **minimal** set to control to ensure the reachability of the error

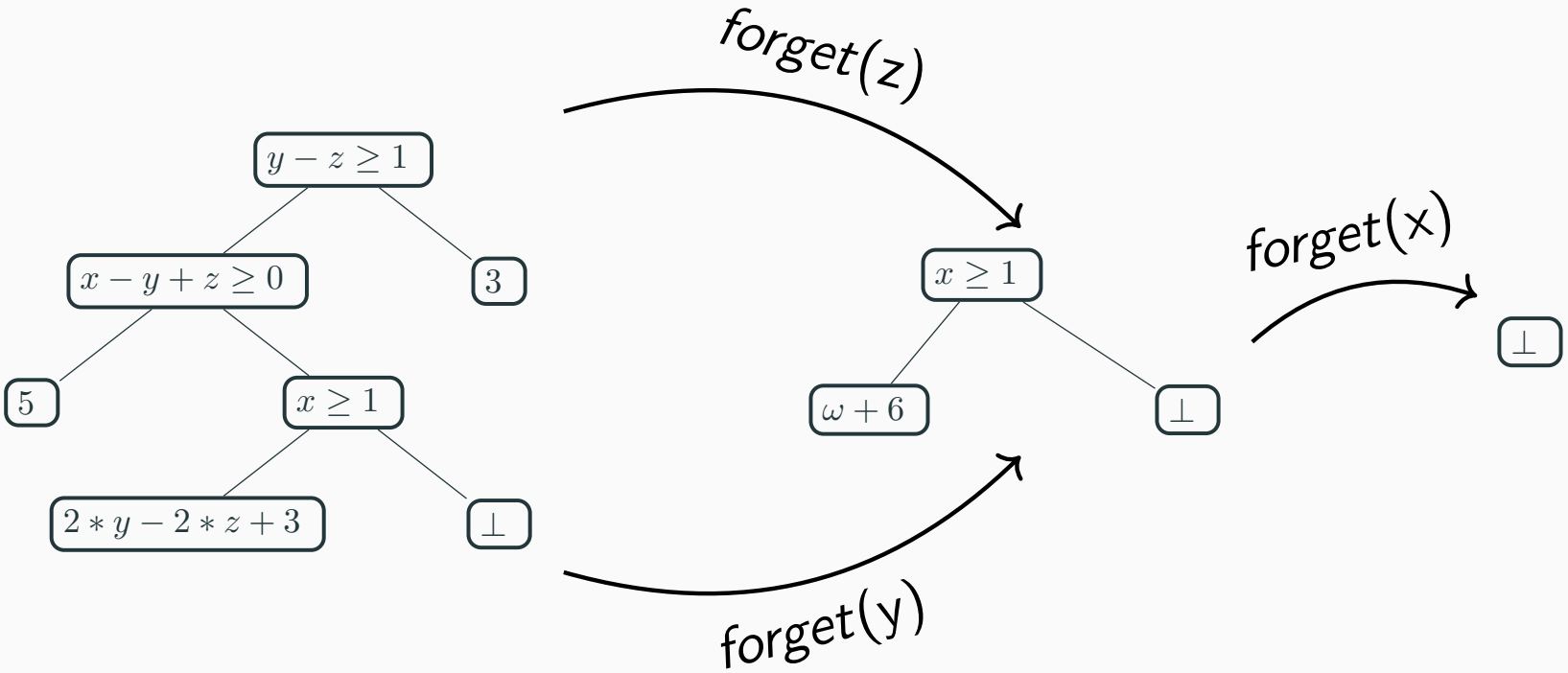

$y - z \geq 1$
$x - y + z \geq 0$
3
5
$x \geq 1$
$2 * y - 2 * z + 3$
$\bot$


forget(z)
$y - z \geq 1$
$x - y + z \geq 0$
3
5
$x \geq 1$
$2 * y - 2 * z + 3$
$\bot$
$x \geq 1$
$\omega + 6$
$\bot$


forget(z)
$y - z \geq 1$
$x - y + z \geq 0$
3
5
$x \geq 1$
$2 * y - 2 * z + 3$
$\bot$
$x \geq 1$
$\omega + 6$
$\bot$
forget(y)



Note: removing contraints on y removes the contraints on z and vice-versa

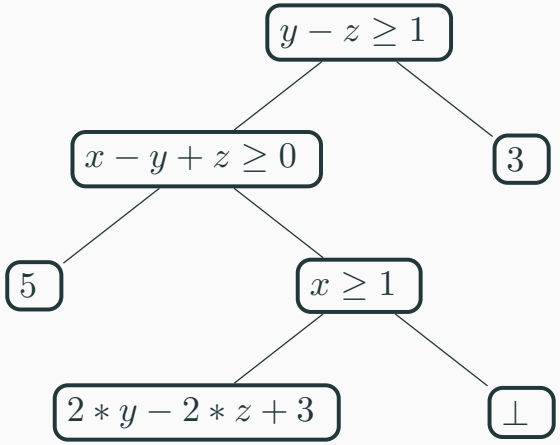


Note: removing contraints on y removes the contraints on z and vice-versa

$\{y, z\}$ is safe

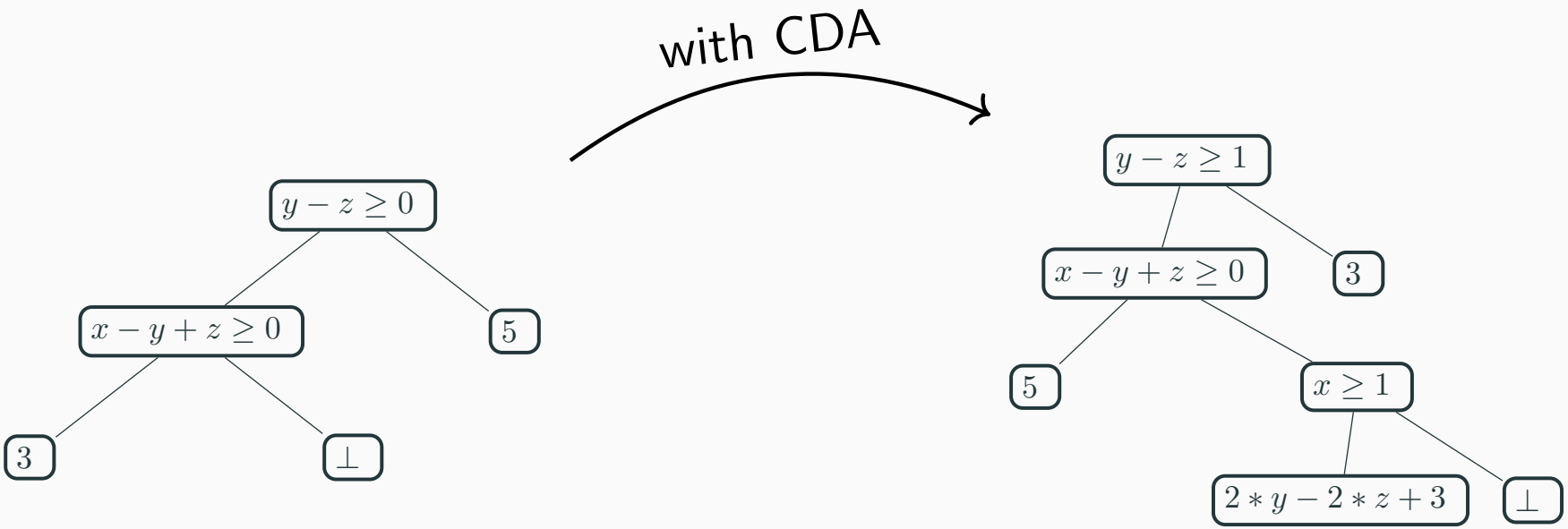
$\{x\}$ is vulnerable

$\{x\}$ is a **second** minimal set to control to ensure reachability of the error



Result of the algorithm is a set of sets to control: $\{\{x\}, \{y, z\}\}$

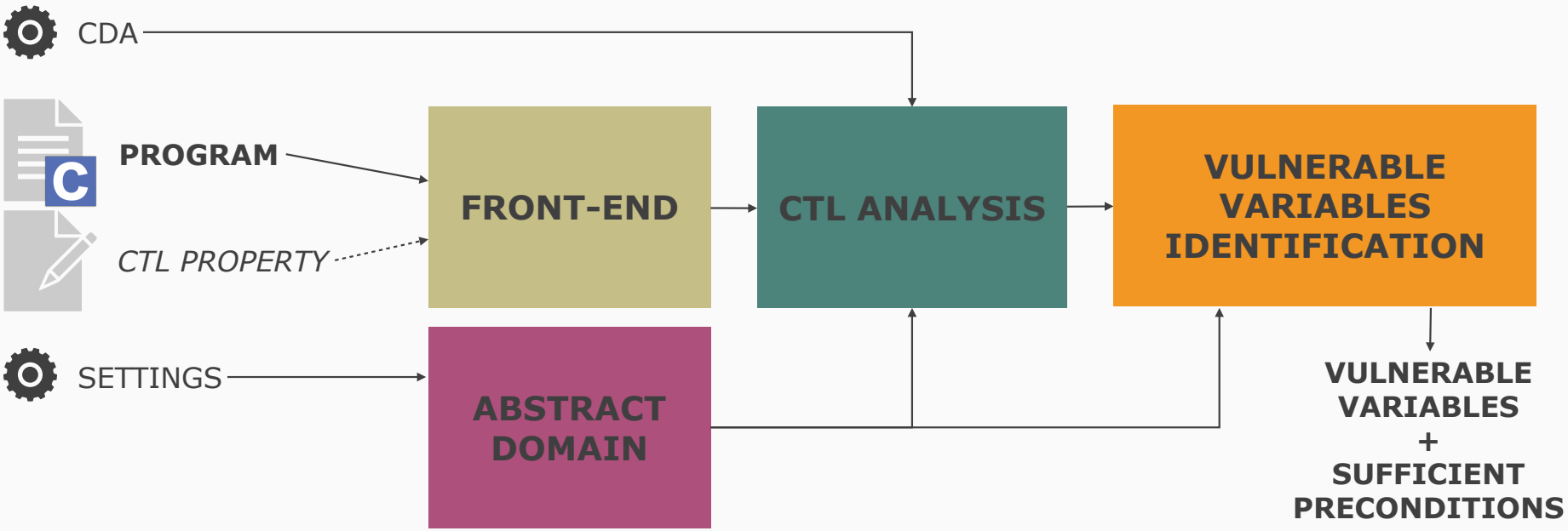
Abstract refinement process



FuncTion-V

Static Analyzer coded in Ocaml

Support a numerical subset of C programs.



Experimentation

Evaluated on 347 examples from: SV-Comp, Literature, LTL-Automizer..

Experimentation

Evaluated on 347 examples from: SV-Comp, Literature, LTL-Automizer..

Number of vulnerable variable sets:

	Vulnerable Sets	Number of Programs	Total Time	Average Time	Average L
NO CDA	0	146	15 s	0.1 s	79
	1	165 (+15)	21 s	0.1 s	22
	2	30	10 s	0.33 s	24
	3	6	5 s	0.7 s	25
CDA	0	154	23 s	0.15 s	77
	1	160 (+14)	934 s	5 s	22
	2	79	432 s	15 s	25
	3	5	124 s	25 s	26

Experimentation

Evaluated on 347 examples from: SV-Comp, Literature, LTL-Automizer..

Number of vulnerable variable sets:

	Vulnerable Sets	Number of Programs	Total Time	Average Time	Average L
NO CDA	0	146	15 s	0.1 s	79
	1	165 (+15)	21 s	0.1 s	22
	2	30	10 s	0.33 s	24
	3	6	5 s	0.7 s	25
CDA	0	154	23 s	0.15 s	77
	1	160 (+14)	934 s	5 s	22
	2	79	432 s	15 s	25
	3	5	124 s	25 s	26

Average minimum percentage of vulnerable variables:

	Termination	Robust Reachability	CTL
NO CDA	17%	26%	32%
CDA	16%	22%	28%

Related Work

Robust Reachability: error reachability independently of the values of uncontrolled variables

Implemented in: BINSEC-RSE

Related Work

Non-exploitability: absence of runtime errors independently of the values of controlled variables

Implemented in: MOPSA

Comparison

FuncTion-V:

(+) : **infer** set of variable that imply Robust reachability or Non-exploitability

(+) : CTL properties: combination of safety and liveness properties

(+) : Termination sensitive

(−) : Restricted to a subset of numerical C-programs while Mopsa and Binsec-RSE are not!

Comparison

FuncTion-V:

(+) : **infer** set of variable that imply Robust reachability or Non-exploitability

(+) : CTL properties: combination of safety and liveness properties

(+) : Termination sensitive

(−) : Restricted to a subset of numerical C-programs while Mopsa and Binsec-RSE are not!

Conclusion

- Detection of variables that an attacker can control to ensure an undesirable properties
- Experimentation on various programs
- **More** detailed in the paper: semantics, dynamic programming algorithm, Conflict driven analysis