

# Ultimate (Kojak) and the Abstraction of Bitwise Operations

---

**Frank Schüssele**<sup>1</sup> Manuel Bentele<sup>1</sup> Daniel Dietsch<sup>1</sup> Matthias Heizmann<sup>2</sup> Xinyu Jiang<sup>1</sup>  
Dominik Klumpp<sup>1</sup> Andreas Podelski<sup>1</sup>

<sup>1</sup>University of Freiburg, Germany

<sup>2</sup>University of Stuttgart, Germany

- KOJAK: Model checker based on interpolant splitting<sup>1</sup>

---

<sup>1</sup>Evren Ermis, Jochen Hoenicke, and Andreas Podelski. “**Splitting via Interpolants**”. In: *VMCAI 2012*. Lecture Notes in Computer Science. 2012.

- KOJAK: Model checker based on interpolant splitting<sup>1</sup>
- control flow graph using SMT-LIB formulas

---

<sup>1</sup>Evren Ermis, Jochen Hoenicke, and Andreas Podelski. “**Splitting via Interpolants**”. In: *VMCAI 2012*. Lecture Notes in Computer Science. 2012.

- KOJAK: Model checker based on interpolant splitting<sup>1</sup>
- control flow graph using SMT-LIB formulas
- shared translation with other ULTIMATE tools

---

<sup>1</sup>Evren Ermis, Jochen Hoenicke, and Andreas Podelski. “**Splitting via Interpolants**”. In: *VMCAI 2012*. Lecture Notes in Computer Science. 2012.

- KOJAK: Model checker based on interpolant splitting<sup>1</sup>
- control flow graph using SMT-LIB formulas
- shared translation with other ULTIMATE tools
  - from C to Boogie

---

<sup>1</sup>Evren Ermis, Jochen Hoenicke, and Andreas Podelski. “**Splitting via Interpolants**”. In: *VMCAI 2012*. Lecture Notes in Computer Science. 2012.

- KOJAK: Model checker based on interpolant splitting<sup>1</sup>
- control flow graph using SMT-LIB formulas
- shared translation with other ULTIMATE tools
  - from C to Boogie
  - from Boogie to control flow graph

---

<sup>1</sup>Evren Ermis, Jochen Hoenicke, and Andreas Podelski. “**Splitting via Interpolants**”. In: *VMCAI 2012*. Lecture Notes in Computer Science. 2012.

```
int main() {  
    int x, y;  
    ...  
    int a = x + y;  
    int b = x | y;  
    ...  
}
```

```
int main() {
```

```
  int x, y;
```

```
  ...
```

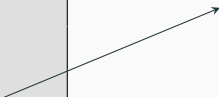
```
  int a = x + y;
```

```
  int b = x | y;
```

```
  ...
```

```
}
```

integer: (= a (+ x y))



```
int main() {  
  int x, y;  
  ...  
  int a = x + y;  
  int b = x | y;  
  ...  
}
```

integer: (= a (+ x y))

bitvector: (= a (bvadd x y))

## Modeling C in SMT-LIB

```
int main() {  
  int x, y;  
  ...  
  int a = x + y;  
  int b = x | y;  
  ...  
}
```

integer: (= a (+ x y))

bitvector: (= a (bvadd x y))

bitvector: (= b (bvor x y))

## Modeling C in SMT-LIB

```
int main() {  
  int x, y;  
  ...  
  int a = x + y;  
  int b = x | y;  
  ...  
}
```

integer: (= a (+ x y))

bitvector: (= a (bvadd x y))

integer: ???

bitvector: (= b (bvor x y))

## Modeling C in SMT-LIB

```
int main() {  
  int x, y;  
  ...  
  int a = x + y;  
  int b = x | y;  
  ...  
}
```

integer: (= a (+ x y))

bitvector: (= a (bvadd x y))

integer: true (overapproximation)

bitvector: (= b (bvor x y))

- based on earlier work<sup>2</sup>, generalized in SV-COMP paper<sup>3</sup>

---

<sup>2</sup>Yuandong Cyrus Liu et al. “**Proving LTL Properties of Bitvector Programs and Decompiled Binaries**”. In: *PLDI 2021*. 2021.

<sup>3</sup>Frank Schüssele et al. “**Ultimate Automizer and the Abstraction of Bitwise Operations**”. In: *TACAS 2024*. 2024.

# Abstraction of Bitwise Operations

- based on earlier work<sup>2</sup>, generalized in SV-COMP paper<sup>3</sup>
- two techniques to improve overapproximation for bitwise operators  
&, |, ^, ~, <<, >>

---

<sup>2</sup>Yuangdong Cyrus Liu et al. “**Proving LTL Properties of Bitvector Programs and Decompiled Binaries**”. In: *PLDI 2021*. 2021.

<sup>3</sup>Frank Schüssele et al. “**Ultimate Automizer and the Abstraction of Bitwise Operations**”. In: *TACAS 2024*. 2024.



## General rules

$$6 \mid 12 \rightsquigarrow 14$$

## General rules

$$6 \mid 12 \rightsquigarrow 14$$

$$\sim x \rightsquigarrow -1 - x$$

## General rules

$6 | 12 \rightsquigarrow 14$

$\sim x \rightsquigarrow -1 - x$

$x \ll 3 \rightsquigarrow x * 8$

$x \gg 4 \rightsquigarrow x / 16$

## General rules

$6 \mid 12 \rightsquigarrow 14$

$\sim x \rightsquigarrow -1 - x$

$x \ll 3 \rightsquigarrow x * 8$

$x \gg 4 \rightsquigarrow x / 16$

## Rules based on bitpattern

$(1\dots 1) \quad x^{-1} \rightsquigarrow -1 - x$

## General rules

$$6 \mid 12 \rightsquigarrow 14$$

$$\sim x \rightsquigarrow -1 - x$$

$$x \ll 3 \rightsquigarrow x * 8$$

$$x \gg 4 \rightsquigarrow x / 16$$

## Rules based on bitpattern

$$(1\dots 1) \quad x \hat{-} 1 \rightsquigarrow -1 - x$$

$$(0\dots 0) \quad 0 \mid x \rightsquigarrow x$$

## General rules

$6 \mid 12 \rightsquigarrow 14$

$\sim x \rightsquigarrow -1 - x$

$x \ll 3 \rightsquigarrow x * 8$

$x \gg 4 \rightsquigarrow x / 16$

## Rules based on bitpattern

$(1\dots 1) \quad x \hat{-} 1 \rightsquigarrow -1 - x$

$(0\dots 0) \quad 0 \mid x \rightsquigarrow x$

$(0\dots 01\dots 1) \quad x \& 255 \rightsquigarrow x \% 256$

## Constrained Overapproximation

```
procedure and(a: int, b: int) {  
    if (a == 0 || b == 0) return 0;  
    if (a == b) return a;  
  
    var r: int;  
    assume (a >= 0 || b < 0) ==> r <= a;  
    assume (a < 0 || b >= 0) ==> r <= b;  
    assume (a >= 0 || b >= 0) ==> r >= 0;  
    assume (a < 0 || b < 0) ==> r > a + b;  
    return r;  
}
```

**Table 1:** Comparison for U**AUTOMIZER** on ReachSafety

	Bitvector			Integer (optimized)			Integer (old)		
		time	mem		time	mem		time	mem
	#	(h)	(GB)	#	(h)	(GB)	#	(h)	(GB)
total (10 205)	1 958	65	1 862	2 076	37	2 600	2 051	36	2 550
safe (7 557)	1 183	41	1 030	1 350	22	1 510	1 324	21	1 440
unsafe (2 648)	775	24	832	726	15	1 090	727	15	1 110

**Table 2:** Comparison for UAUTOMIZER on Termination-BitVectors

	Integer (optimized)			Integer (old)		
	#	time (s)	mem (GB)	#	time (s)	mem (GB)
total (37)	31	410	12.1	12	122	4.2
safe (23)	23	325	9.2	7	73	2.5
unsafe (14)	8	85	2.9	5	49	1.7