Ultimate (Kojak) and the Abstraction of Bitwise Operations

Frank Schüssele[1] Manuel Bentele[1] Daniel Dietsch[1] Matthias Heizmann[2] Xinyu Jiang[1] Dominik Klumpp[1] Andreas Podelski[1]

[1]University of Freiburg, Germany

[2]University of Stuttgart, Germany

Ultimate Kojak

- KOJAK: Model checker based on interpolant splitting[1]

[1] Evren Ermis, Jochen Hoenicke, and Andreas Podelski. **"Splitting via Interpolants".** In: *VMCAI 2012*. Lecture Notes in Computer Science. 2012.

Ultimate Kojak

- KOJAK: Model checker based on interpolant splitting[1]
- control flow graph using SMT-LIB formulas

[1] Evren Ermis, Jochen Hoenicke, and Andreas Podelski. **"Splitting via Interpolants".** In: *VMCAI 2012*. Lecture Notes in Computer Science. 2012.

Ultimate Kojak

- KOJAK: Model checker based on interpolant splitting[1]
- control flow graph using SMT-LIB formulas
- shared translation with other ULTIMATE tools

[1] Evren Ermis, Jochen Hoenicke, and Andreas Podelski. **"Splitting via Interpolants".** In: *VMCAI 2012*. Lecture Notes in Computer Science. 2012.

Ultimate Kojak

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

[1] Evren Ermis, Jochen Hoenicke, and Andreas Podelski. **"Splitting via Interpolants".** In: *VMCAI 2012*. Lecture Notes in Computer Science. 2012.

Ultimate Kojak

- Kojak: Model checker based on interpolant splitting[1]
- control flow graph using SMT-LIB formulas
- shared translation with other Ultimate tools
  - from C to Boogie
  - from Boogie to control flow graph

[1] Evren Ermis, Jochen Hoenicke, and Andreas Podelski. **"Splitting via Interpolants".** In: *VMCAI 2012*. Lecture Notes in Computer Science. 2012.

Modeling C in SMT-LIB

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

Modeling C in SMT-LIB

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

integer: `(= a (+ x y))`

Modeling C in SMT-LIB

```c
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

```c
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

```c
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

```c
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))`

Abstraction of Bitwise Operations

- based on earlier work[2], generalized in SV-COMP paper[3]

[2] Yuandong Cyrus Liu et al. **"Proving LTL Properties of Bitvector Programs and Decompiled Binaries".** In: *PLDI 2021*. 2021.

[3] Frank Schüssele et al. **"Ultimate Automizer and the Abstraction of Bitwise Operations".** In: *TACAS 2024*. 2024.

Abstraction of Bitwise Operations

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

[2]Yuandong Cyrus Liu et al. **"Proving LTL Properties of Bitvector Programs and Decompiled Binaries".** In: *PLDI 2021*. 2021.
[3]Frank Schüssele et al. **"Ultimate Automizer and the Abstraction of Bitwise Operations".** In: *TACAS 2024*. 2024.

Rewrite rules

Rewrite rules

General rules

$6 \mid 12 \quad \rightsquigarrow \qquad 14$

Rewrite rules

General rules

$$\begin{aligned} \texttt{6 | 12} &\rightsquigarrow \texttt{14} \\ \texttt{\~{}x} &\rightsquigarrow \texttt{-1 - x} \end{aligned}$$

Rewrite rules

General rules

`6 | 12`	$\rightsquigarrow$	`14`
`~x`	$\rightsquigarrow$	`-1 - x`
`x << 3`	$\rightsquigarrow$	`x * 8`
`x >> 4`	$\rightsquigarrow$	`x / 16`

Rewrite rules

General rules

`6 | 12`	$\rightsquigarrow$	`14`
`~x`	$\rightsquigarrow$	`-1 - x`
`x << 3`	$\rightsquigarrow$	`x * 8`
`x >> 4`	$\rightsquigarrow$	`x / 16`

Rules based on bitpattern

$(1...1)$	`x ^ -1`	$\rightsquigarrow$	`-1 - x`

Rewrite rules

General rules

`6 | 12`	$\rightsquigarrow$	`14`
`~x`	$\rightsquigarrow$	`-1 - x`
`x << 3`	$\rightsquigarrow$	`x * 8`
`x >> 4`	$\rightsquigarrow$	`x / 16`

Rules based on bitpattern

(1...1)	`x ^ -1`	$\rightsquigarrow$	`-1 - x`
(0...0)	`0 | x`	$\rightsquigarrow$	`x`

Rewrite rules

General rules

`6 | 12`	$\rightsquigarrow$	`14`
`~x`	$\rightsquigarrow$	`-1 - x`
`x << 3`	$\rightsquigarrow$	`x * 8`
`x >> 4`	$\rightsquigarrow$	`x / 16`

Rules based on bitpattern

(1...1)	`x ^ -1`	$\rightsquigarrow$	`-1 - x`
(0...0)	`0 | x`	$\rightsquigarrow$	`x`
(0...01...1)	`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;
}
```

Evaluation

Table 1: Comparison for UAutomizer on ReachSafety

	Bitvector			Integer (optimized)			Integer (old)		
	#	time (h)	mem (GB)	#	time (h)	mem (GB)	#	time (h)	mem (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

Evaluation

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