Proving LTL Properties of Bitvector Programs

Yuandong Cyrus Liu
Advisor, Eric Koskinen
Stevens Institute of Technology

***Abstract*—There is increasing interest in applying verification tools to programs that have bitvector operations. SMT solvers, which serve as a foundation for these tools, have thus increased support for bitvector reasoning through bit-blasting and linear arithmetic approximations. Still, verification tools are limited on termination and LTL verification of bitvector programs.**

In this work, we show that similar linear arithmetic approximation of bitvector operations can be done at the source level through transformations. Specifically, we introduce new paths that over-approximate bitvector operations with linear conditions/constraints, increasing branching but allowing us to better exploit the well-developed integer reasoning and interpolation of verification tools. We present two sets of rules, namely rewriting rules and weakening rules, that can be implemented as bitwise branching of program transformation, the branching path can facilitate verification tools widen verification tasks over bitvector programs. Our experiment shows this exploitation of integer reasoning and interpolation enables competitive termination verification of bitvector programs and leads to the first effective technique for LTL verification of bitvector programs.

Finally, for the cases that are not covered by our bitwise branching rules, we explore a dynamic approach combined with static analysis to tackle more complicated bitvector programs. We execute the program and collect concrete traces in the locations of interest, inferring program invariants from concrete traces, these linear invariants can be used to approximate the bitwise expressions, therefore the static analysis tools can reason about the approximate programs and return the verification results.

Problems

- Bit-blasting in SMT practical applications, leads to exponential growth ($\mathcal{O}(2^n)$).
- Verification tools (e.g. CPAchecker, Ultimate) have limited support for liveness verification over the bitvector domain.
- LTL verification tasks are absent from SV-COMP.
- Very limited bitvector benchmarks in SV-COMP.

Theory of Bitwise Branching

- **Rewriting Rules**: $\mathcal{C} \vdash_E e_{bv} \rightsquigarrow e_{int}$ means under **condition** $\mathcal{C}$ bitvector expression e_{bv} can be approx. with linear expression e_{int}.
- **Weakening Rules**: $\mathcal{C} \vdash_S s_{bv} \rightsquigarrow s_{int}$ means under **condition** $\mathcal{C}$ bitvector statement s_{bv} can be approx. with linear statement s_{int}.

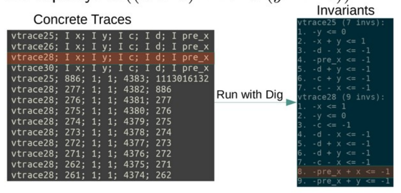
Linear Condition		BV Expr.	Linear Apx.	
$e_1 = 0$	$\vdash_E$	$e_1\&e_2$	$\rightsquigarrow 0$	[R-And-0]
$(e_1 = 0 \vee e_1 = 1) \wedge e_2 = 1$	$\vdash_E$	$e_1\&e_2$	$\rightsquigarrow e_1$	[R-And-1]
$(e_1 = 0 \vee e_1 = 1) \wedge (e_2 = 0 \vee e_2 = 1)$	$\vdash_E$	$e_1\&e_2$	$\rightsquigarrow e_1\&\&e_2$	[R-And-LOG]
$e_1 \geq 0 \wedge e_2 = 1$	$\vdash_E$	$e_1\&e_2$	$\rightsquigarrow e_1\%2$	[R-And-LBS]
$e_2 = 0$	$\vdash_E$	$e_1 \mid e_2$	$\rightsquigarrow e_1$	[R-Or-0]
$(e_1 = 0 \vee e_1 = 1) \wedge e_2 = 1$	$\vdash_E$	$e_1 \mid e_2$	$\rightsquigarrow 1$	[R-Or-1]
$e_2 = 0$	$\vdash_E$	$e_1\hat{\ }e_2$	$\rightsquigarrow e_1$	[R-Xor-0]
$e_1 = e_2 = 0 \vee e_1 = e_2 = 1$	$\vdash_E$	$e_1\hat{\ }e_2$	$\rightsquigarrow 0$	[R-Xor-Eq]
$(e_1 = 1 \wedge e_2 = 0) \vee (e_1 = 0 \wedge e_2 = 1)$	$\vdash_E$	$e_1\hat{\ }e_2$	$\rightsquigarrow 1$	[R-Xor-Neq]
$e_1 \geq 0 \wedge e_2 =$ `CHAR_BIT * sizeof`$(e_1) - 1$	$\vdash_E$	e_1>>e_2	$\rightsquigarrow 0$	[R-RightShift-Pos]
$e_1 < 0 \wedge e_2 =$ `CHAR_BIT * sizeof`$(e_1) - 1$	$\vdash_E$	e_1>>e_2	$\rightsquigarrow -1$	[R-RightShift-Neg]

Linear Condition		BV STMT	Linear Apx	
$e_1 \geq 0 \wedge e_2 \geq 0$	$\vdash_S$	r op$_{le}$ $e_1\&e_2$	$\rightsquigarrow r$<=e_1 && r<=e_2	[W-And-Pos]
$e_1 < 0 \wedge e_2 < 0$	$\vdash_S$	r op$_{le}$ $e_1\&e_2$	$\rightsquigarrow r$<=e_1 && r<=e_2 && r<0	[W-And-Neg]
$e_1 \geq 0 \wedge e_2 < 0$	$\vdash_S$	r op$_{eq}$ $e_1\&e_2$	$\rightsquigarrow$ 0<=r && r<=e_1	[W-And-Mix]
$(e_1 = 0 \vee e_1 = 1) \wedge (e_2 = 0 \vee e_2 = 1)$	$\vdash_S$	$(e_1 \mid e_2)$==0	$\rightsquigarrow e_1$==0 && e_2==0	[R-Or-LOG]
$e_1 \geq 0 \wedge$ is_const(e_2)	$\vdash_S$	r op$_{ge}$ $e_1 \mid e_2$	$\rightsquigarrow r$>=e_2	[W-Or-Const]
$e_1 \geq 0 \wedge e_2 \geq 0$	$\vdash_S$	r op$_{ge}$ $e_1 \mid e_2$	$\rightsquigarrow r$>=e_1 && r>=e_2	[W-Or-Pos]
$e_1 < 0 \wedge e_2 < 0$	$\vdash_S$	r op$_{eq}$ $e_1 \mid e_2$	$\rightsquigarrow r$>=e_1 && r>=e_2 && r<0	[W-Or-Neg]
$e_1 \geq 0 \wedge e_2 < 0$	$\vdash_S$	r op$_{eq}$ $e_1 \mid e_2$	$\rightsquigarrow e_2$<=r && r<0	[W-Or-Mix]
$e_1 \geq 0 \wedge e_2 \geq 0$	$\vdash_S$	r op$_{ge}$ $e_1\hat{\ }e_2$	$\rightsquigarrow r$>=0	[W-XOr-Pos]
$e_1 < 0 \wedge e_2 < 0$	$\vdash_S$	r op$_{ge}$ $e_1\hat{\ }e_2$	$\rightsquigarrow r$>=0	[W-XOr-Neg]
$e_1 \geq 0 \wedge e_2 < 0$	$\vdash_S$	r op$_{le}$ $e_1\hat{\ }e_2$	$\rightsquigarrow r$<0	[W-XOr-Mix]
$e_1 \geq 0$	$\vdash_S$	r op$_{le}$ $\sim e_1$	$\rightsquigarrow r$<0	[W-Cpl-Pos]
$e_1 < 0$	$\vdash_S$	r op$_{ge}$ $\sim e_1$	$\rightsquigarrow r$>=0	[W-Cpl-Neg]

Dynamic LTL Verification

LTL Property: $G((x > 0) \implies F(y == 0))$

```
1 while(c<50){
2   c++;
3   d = nondet();
4   if(d<0) d = d*(-1);
5   x = nondet();
6   y = 1;
7   while(x>0){
8     d--;
9      x = (x&d) -1;
10     if (x<=1){
11       y=0;
12     }
13  }
```



```
vtrace25; I x; I y; I c; I d; I pre_x
vtrace26; I x; I y; I c; I d; I pre_x
vtrace28; I x; I y; I c; I d; I pre_x
vtrace30; I x; I y; I c; I d; I pre_x
vtrace25; 886; 1; 1; 4383; 1113016132
vtrace28; 277; 1; 1; 4382; 886
vtrace28; 276; 1; 1; 4381; 277
vtrace28; 275; 1; 1; 4380; 276
vtrace28; 274; 1; 1; 4379; 275
vtrace28; 273; 1; 1; 4378; 274
vtrace28; 272; 1; 1; 4377; 273
vtrace28; 271; 1; 1; 4376; 272
vtrace28; 262; 1; 1; 4375; 271
vtrace28; 261; 1; 1; 4374; 262
```

```
vtrace25 (7 invs):
1. -y <= 0
2. -x + y <= 1
3. -d - x <= -1
4. -pre_x <= -1
5. -d + y <= -1
6. -c + y <= -1
7. -c - x <= -1
vtrace28 (9 invs):
1. -x <= 1
2. -y <= 0
3. -c <= -1
4. -d - x <= -1
5. -d + x <= -1
6. -d + y <= -1
7. -c - x <= -1
8. -pre_x + x <= -1
9. -pre_x + y <= -1
```

Invariant $-pre\_x + x \leq -1$ shows x is decreasing at bitwise location 9.

- Locate bitvector expression, instrument source code with traces.
- Compile source code, random sampling concrete traces.
- Infer invariants at trace locations.
- Replace bitwise expression with effective invariant, run with LTL verifier.

Rules Application

$e_1 \geq 0 \wedge e_2 \geq 0 \quad \vdash_S \quad r$ op$_{le}$ $e_1\&e_2 \quad \rightsquigarrow r$<=$e_1$ && r<=e_2 [W-And-Pos]

```
a = *;
assume(a>0);
while(x>0){
  a--;
  x = x & a;
}
```

- Tools struggle.
- Ultimate, e.g., reports Unknown.
- $\mathcal{I} : x > 0 \wedge a > 0$
- $T : x{>}0 \wedge a'{=}a{-}1 \wedge x'{=}x\&a'$
- Tools fail to show: $\mathcal{I} \wedge T \wedge x'{>}0 \implies \mathcal{I}'$

```
a = *; assume(a > 0);
while (x > 0) {
  { x > 0 ∧ a > 0 }
  a--;
  if (x >= 0 && a >= 0)
  then { x = *; assume(x <= a); }
  else { x = x & a; }
}
```

- $T' = x{>}0 \wedge a' = a{-}1 \wedge ((x{\geq}0 \wedge a'{\geq}0 \wedge x'{\leq}a') \vee$ ~~$(\neg(x \geq 0 \wedge a' \geq 0) \wedge x' = x\&a')$~~$)$
- Tools can prove that $\mathcal{I} \wedge T' \wedge x'{>}0 \implies \mathcal{I}'$, ranking function $\mathcal{R}(x, a) = a$

Termination and LTL Experiments

Termination Benchmarks

- No SV-COMP benchmarks for term. of bitvector programs.
- AProVE Benchmarks, although only 18/118 are bitvector programs.
- 31 new benchmarks, adapted from "Bit Hacks".

State-of-the-art Tools

Tool	BitVec.	Term.	LTL
Ultimate	Limited	Yes	Yes
AProVE	Yes	Yes	No
KITTeL	Yes	Yes	No
CPAchecker	Limited	Yes	No
2LS	Yes	Yes	No
UltimateBwB	Yes	Yes	Yes

	(ii) TermBitBench						(i) AproveBench					
	AProVE	CPA	KITTeL	2LS	Ult	UltBwB	AProVE	CPA	KITTeL	2LS	Ult	UltBwB
✔ (Terminating)	5	1	7	8	2	18	1	3	3	14	2	2
⚡✔ (FN)	1	-	-	-	-	-	-	-	-	-	-	-
✗ (Nonterminating)	6	10	-	8	-	13	-	-	-	-	-	-
⚡✗ (FP)	2	7	-	3	-	-	-	10	-	-	2	6
?(Unknown)	14	13	-	-	29	-	10	3	-	1	14	8
T (Time Out)	3	-	19	12	-	-	7	-	10	2	-	1
M (Out of Memory)	-	-	-	-	-	-	-	-	-	1	-	1
☢ (Crash)	-	-	5	-	-	-	-	2	5	-	-	-

Take Away: First effective strategy for termination of bitvector programs.

Example: ($\varphi = \Box(\Diamond(n < 0))$)

```
while(1) {
  n = *; x = *; y = x-1;
  while(x>0 && n>0) {
    n++;
     y = x | n;
    x = x - y;
  }
  n = -1;
}
```

Tools and Benchmarks

- No techniques can prove LTL of bitvector programs. The closest possible verifier is Ultimate.
- No LTL verification tasks in SV-COMP.
- Contribute **BitHacks** (26) and **LTLBitBench** (42).

	(iv) Bithacks		(iii) LTLBit Bench	
	Ultimate	w. BwB	Ultimate	w. BwB
✔ (Satisfied)	3	10	-	21
✗ (Unsatisfied)	-	7	-	20
?(Unknown)	21	5	42	-
T (Time Out)	1	1	-	1
M (Out of Memory)	1	3	-	-

Take Away: First technique for verifying LTL of bitvector programs.