HiFrog: Interpolation-based Software Verification using Theory Refinement

Sepideh Asadi, Karine Even Mendoza, Grigory Fedyukovich, Antti E.J. Hyvarinen, Hana Chockler, Natasha Sharygina
University of Lugano (USI), Switzerland, King's College London, UK, University of Washington, USA

What ? A bounded model checker for Verification of safety property for C programs.

How ? Uses function summaries based on Craig interpolation and supports a novel technique for abstraction refinement.

Why ? To avoid repetition of same verification tasks while checking multiple properties of same code and to eliminate spurious behaviors!

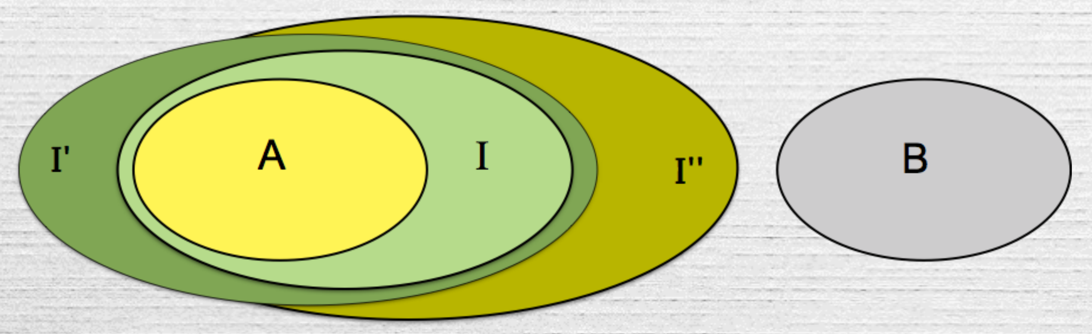
Features ?

- SMT-based Bounded Model Checker
- **Controllable** interpolation system for SMT (flexible in **Size** & **Strength**)
- A new approach called **Theory Refinement** to have simple proofs using SMT
- Automatically identifies where precision is needed and uses precise theories only when necessary
- Support of user-defined summaries

Background

Interpolation

- For A ∧ B is unsatisfiable, I is a quantifier-free formula such that:
- A ⟶ I
- B ∧ I is unsatisfiable
- I is defined over common symbols of A and B



Example

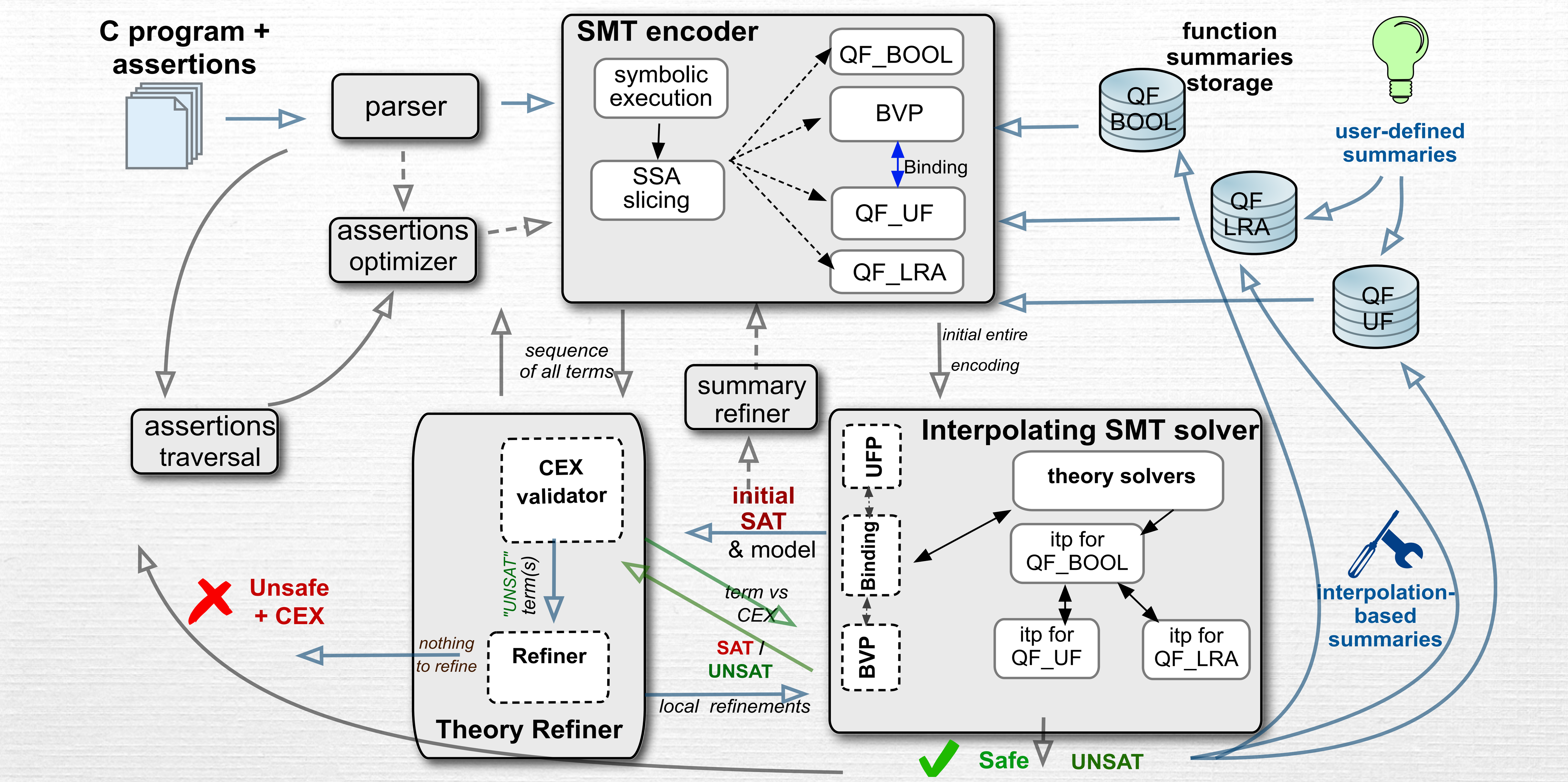
- Use of function summary in a C program with assertions

```
void main(){
  int y = 1;
  int x =
nondet();
  if (x > 0)
   y = f(x);
  assert (y >= 0);
  assert (y >= 1);
}

int f(int a) {
  if (a < 10)
    return a;
  return a - 10;
}
```

```
void main()
{
  int y = 1;
  int x = nondet();

  if (x > 0){
    assume(y >= 0);
  }
  assert(y >= 0);
  assert(y >= 1);
}
```

(a > 0) → (f_return >= 0)

Architecture

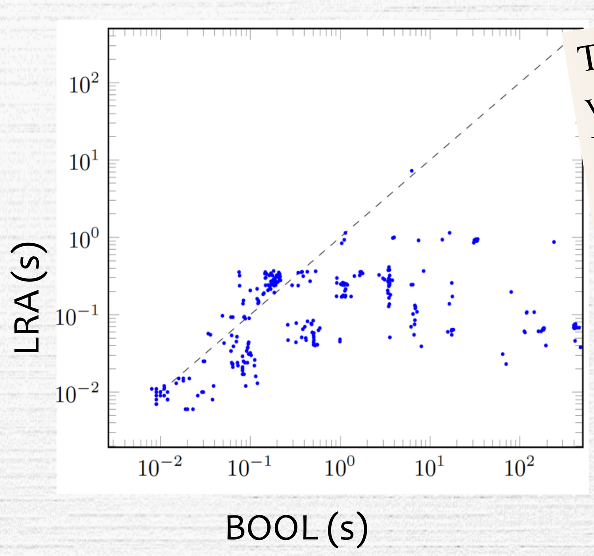


Theory Refinement Algorithm

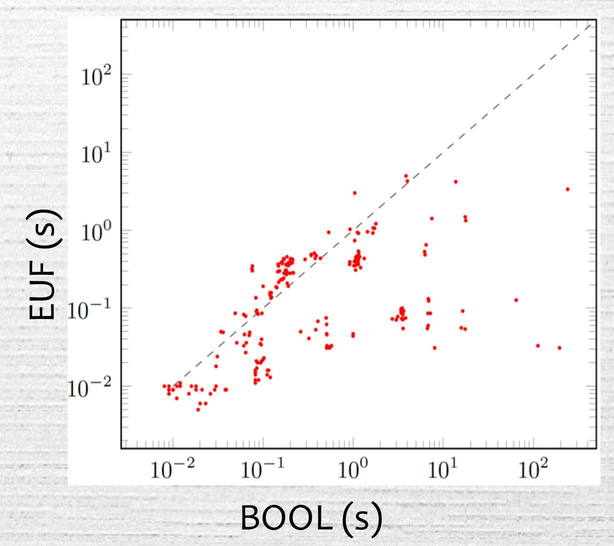
input : $P = \{(x_1 = t_1), \ldots, (x_n = t_n)\}$: a program, and t: a safety property
output: $\langle \textbf{Safe}, \bot \rangle$ or $\langle \textbf{Unsafe}, CE^b \rangle$
For all $1 \leq i \leq n$ initialize $\rho[x_i = t_i] \leftarrow [x_i = t_i]^u$
$\rho[t] \leftarrow [t]^u$
$F_B \leftarrow \top$
while true do
  $Query \leftarrow \rho[x_1 = t_1] \wedge \ldots \wedge \rho[x_n = t_n] \wedge \neg\rho[t] \wedge F_B$
  $\langle result, CE \rangle \leftarrow \texttt{checkSAT}(Query)$
  if *result is* **UnSAT then**
    return $\langle \textbf{Safe}, \bot \rangle$
  end
  $CE^b \leftarrow \texttt{getValues}(CE)$
  foreach $s \in P \cup \{t\}$ *s.t.* $\rho[s] \not\models [s]^b$ **do**
    $\langle result, \_ \rangle \leftarrow \texttt{checkSAT}([s]^b \wedge CE^b)$
    if *result is* **UnSAT then**
      $\rho[s] \leftarrow \texttt{refine}^s(\rho[s])$
      $F_B \leftarrow \texttt{computeBinding}(\rho)$
      break
    end
  end
  if *No s was refined at line 14* **then**
    return $\langle \textbf{Unsafe}, CE^b \rangle$
  end
end

Experiments on SMT vs. Boolean Logic

C Benchmarks	#assertion	EUF	LRA	Bool
token.c	54	34	34	34
s3.c	131	18	21	26
mem.c	149	96	96	96
disk.c	79	6	6	23
ddv.c	152	47	47	142
café.c	115	15	20	30
tcas_asrt.c	162	16	29	29
p2p.c	244	8	20	94
floppy1.c	18	15	16	18
Percentage of success		50.65%	69.2%	100%

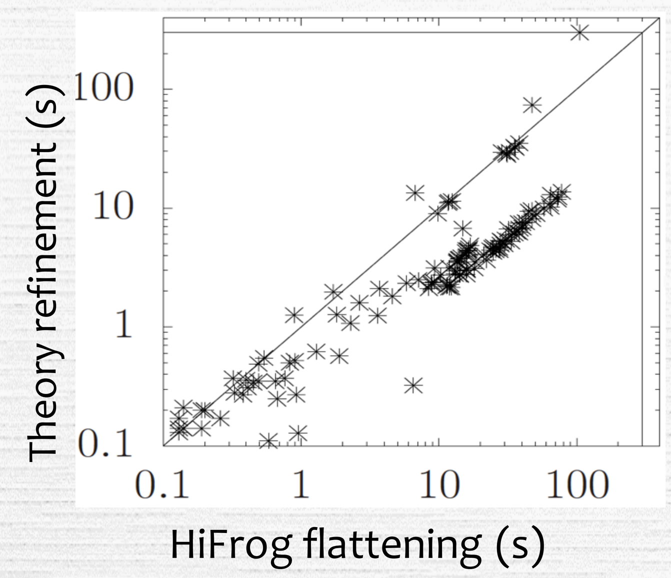


The verification with EUF and LRA is an order of magnitude faster than the verification with BOOL.

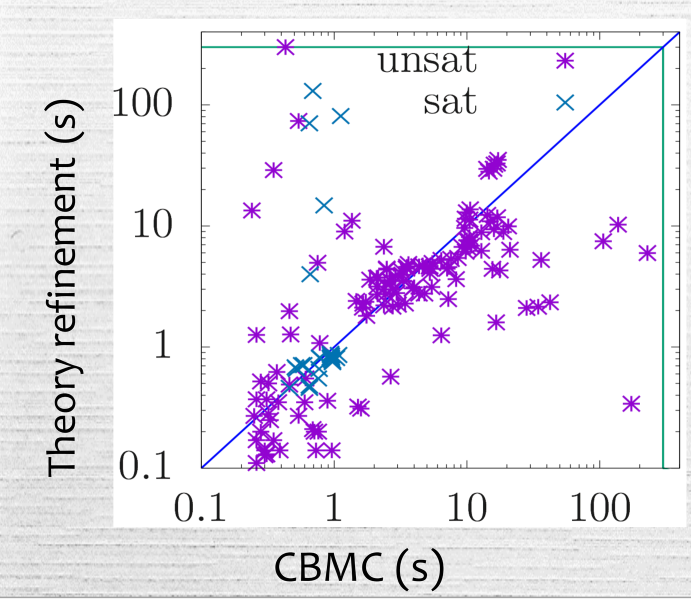


Experiments: 1100 verification task of SV-COMP benchmarks from which **490** were proven to hold using QF-BOOL. Our experiments show a large amount of properties were also proven to be correct by employing the light-weight theories of HiFrog (namely, **50.65%** and **69.2%** of validated properties out of 490 for EUF and LRA respectively)

Experiments on Theory Refinement



Improvement is seen both in the running time and in the size of the resulting formula, demonstrating that the spurious counter-examples are usually eliminated by refining a small number of statements in the formula.



Benchmarks:
safe (128 instances)
unsafe (30 instances)

In 101 cases, HiFrog was either as fast or faster than CBMC

Publications:

1) HiFrog: SMT-based Function Summarization for Software Verification, TACAS, 2017.

2) Theory Refinement for Program Verification, SAT 2017.

3) Duality-Based Interpolation for Quantifier-Free Equalities and Uninterpreted Functions, FMCAD 2017.

Webpage: http://verify.inf.usi.ch/hifrog