Verified Isabelle/HOL Tactic for the Theory of Bounded Linear Integer Arithmetic Based on Quantifier Instantiation and SMT

Rafael Sadykov , Mikhail Mandrykin*

Moscow State University and *ISP RAS

May 28, 2020

Outline

Introduction

In the process of verification software in the C language, SMT solvers are widely applied. This work propose a method for validating formulas in the theory of bounded integers based on the use of an SMT solver for QF UFLIA logic, intended to automatically reconstruct proofs within the Isabelle/HOL system for interactive theorem proving.

Introduction

- We will start with examples of formulas from real verified programs that are implemented in our TSMT Isabelle/HOL tactic. And we will start our introduction from the document in our repository `https://forge.ispras.ru/projects/tsmt/repository` TSMT_Bounded_Examples.
- It will show that formulas of bounded integers are often found in practice.
- Basically, Isabelle tactics unat_arith and uint_arith are mainly used.

Introduction

- Then we will formulate logical theory and definition of decision procedure completeness and conclude that a complete decision procedure proves any formula in the corresponding logical theory.
- Formulate the definition of the theory axiomatically and give examples of satisfiable and unsatisfiable formulas to make semantics of operations more understandable.
- We have proved completeness, including formalization and we can show the formulation of the last completeness theorem. And then show how the new implementation resolves formulas, where uint_arith does not work and which counterexamples on solvable formulas tsmt produces.

An example program

First example formalized within AutoCorres framework.

```
void *memcpy(void *dest, void *src,...size) {
    unsigned long i;
    char *d = (char*)dest, *s = (char*)src;
    for (i = 0; i < size; i++) {
        d[i] = s[i];
    }
    return dest;
}
```

The function copies size bytes from the memory area pointed by src to the memory area pointed by dest. The function returns the destination address dest. Memory areas should not overlap $src - dst \geq size$, otherwise the data may not be copied correctly.

An example program

- In these section we show some goals from our Isabelle/HOL tactic in file TSMT_Bounded_Examples. The goals were taken from the real-world examples featuring verified C implementations of functions *memcpy* and *quicksort* formalized within AutoCorres framework.

memcpy example

```
lemma
  "c_guard src ⟹
   c_guard dst ⟹
   sz = of_int64 (length bs) ⟹
   bytes_at s0 src bs ⟹
   no_wrap src (uint64 (of_int64 (length bs))) ⟹
   no_wrap dst (uint64 (of_int64 (length bs))) ⟹
   no_overlap src dst (uint64 (of_int64 (length bs))) ⟹
   uint64 i ≤ uint64 (of_int64 (length bs)) ⟹
   bytes_at (update_bytes s0 dst (take (nat (uint64 i))) bs) dst (take (nat (uint64 i)) bs) ⟹
   bytes_at (update_bytes s0 dst (take (nat (uint64 i))) bs) src bs ⟹
   i < of_int64 (int (length bs)) ⟹ uint64 (i + 1) ≤ uint64 (of_int64 (int (length bs)))"
  by uint_arith

lemma memcpy_wp'_1:
  "c_guard src ⟹
   c_guard dst ⟹
   sz = of_int64 (length bs) ⟹
   bytes_at s0 src bs ⟹
   no_wrap src (uint64 (of_int64 (length bs))) ⟹
   no_wrap dst (uint64 (of_int64 (length bs))) ⟹
   no_overlap src dst (uint64 (of_int64 (length bs))) ⟹
   uint64 i ≤ uint64 (of_int64 (length bs)) ⟹
   bytes_at (update_bytes s0 dst (take (nat (uint64 i))) bs) dst (take (nat (uint64 i)) bs) ⟹
   bytes_at (update_bytes s0 dst (take (nat (uint64 i))) bs) src bs ⟹
   i < of_int64 (int (length bs)) ⟹ uint64 (i + 1) ≤ uint64 (of_int64 (int (length bs)))"
  by (tsmt (ubound))
```

Basic definitions

If an assignment α satisfies (according to the truth tables) a formula φ, we write: $\alpha \vDash \varphi$.
Notation $\vdash_H \varphi$ there exists a proof of φ in H.

Definition

A formula φ is satisfiable if $\exists \alpha . \alpha \vDash \varphi$

Definition

A formula φ is valid if $\forall \alpha . \alpha \vDash \varphi$. If φ valid we write $\vDash \varphi$.

Soundness and completeness of a deductive system

For a deductive system D,

1. D is sound for a logic L, if for every formula f in L,

$$\vdash_D f \longrightarrow \vDash f$$

   I.e., all formulas proven by the deductive system are valid.

2. D is complete if for every formula f in

$$\vDash f \longrightarrow \vdash_D f$$

   I.e., the deductive system can prove all valid formulas

The decision problem

Definition (Decision problem)

The decision problem for a formula: given φ, is φ valid?

Definition (Decision Procedure)

A decision procedure for a logic is an algorithm that solves the decision problem for any formula in this logic.

We are naturally interested in a sound and complete decision procedure.

Soundness and Completeness

What does it mean that a decision procedure is sound and complete?

- Soundness: the answer returned by the decision procedure is always correct
- Completeness: returns with a yes/no answer in finite time.

Soundness and Completeness

Consider formula

$$\models F \iff \models F^*,$$

F and F^* will be discussed later, F^* is a result of some translation procedure of formula F. And assume that F^* already have decision procedure in some formula.

Then in our paper we need to show following statements:

- Soundness: if $\models F$ is True then $\models F^*$ is also True.
- Completeness: if $\models F^*$ is True then $\models F$ is also True.

Definition

A logic is decidable $\iff$ there is a sound and complete algorithm that decides if a well-formed expression in this logic is valid.

Bounded integers

- Theories of bounded integers both with overflow (for unsigned integers in C) and without overflow (for signed integers), and also theory of finite interpreted sets are good examples of such theories.
- One of the possible ways to support such theories is to directly implement them in SMT-solvers. But this method is often time-consuming.
- Another way is to implement custom quantifier instantiation strategies to reduce formulas in unsupported theories to formulas in widespread decidable logics such as QF UFLIA.

QF UFLIA

The QF UFLIA logic is combination of

- QF LIA (QF LIA includes quantifier-free formulas with equality in the theory of linear integer arithmetic (LIA). Its signature includes the following function symbols: $\{+, c\times, \leq\}$)
- QF UF (QF UF includes quantifier-free formulas with equality in the theory of uninterpreted functions.)

Quantifier-free means free from quantifiers such as $(\forall, \exists)$.
Example of equality and uninterpreted function f: if $x_1 = x_2$ then $f(x_1) = f(x_2)$ else $f(x_1) \neq f(x_2)$.

Bounded integers

In our work, we present an instantiation procedure for translating formulas in the theory of bounded integers, which is called BLIA, without overflow into the QF UFLIA logic. We formally proved soundness and completeness of our instantiation procedure in Isabelle. The paper presents an informal description of this proof as well as some considerations on the efficiency of the proposed procedure.

Axioms

$\Sigma = \{+_b, \times_b, \leq_b\},\ a,\ b \in \mathbb{Z}_b,\ (c)_b \in \mathbb{Z}_b,\ v(a) \in \mathbb{Z}, c \in \mathbb{Z}$ — constant

$$
\begin{array}{rcr}
\forall a\, b \in \mathbb{Z}_b. & L \leq v(a) + v(b) \leq U \implies v(a +_b b) = v(a) + v(b), & (A1) \\
\forall a \in \mathbb{Z}_b. \forall c \in \mathbb{Z}. & L \leq c \times v(a) \leq U \implies v(c \times_b a) = c \times v(a), & (A2) \\
\forall a\, b \in \mathbb{Z}_b. & a \leq_b b \iff v(a) \leq v(b), & (A3) \\
\forall a \in \mathbb{Z}_b. & L \leq v(a) \leq U, & (A4) \\
\forall c \in \mathbb{Z}. & L \leq c \leq U \implies v\big((c)_b\big) = c, & (A5) \\
\forall a\, b \in \mathbb{Z}_b. & v(a) = v(b) \implies a = b. & (A6)
\end{array}
$$

$$
\forall a \in \mathbb{Z}_b. \qquad \big(v(a)\big)_b \; = \; a. \qquad (A6')
$$

Triggers

- $(A1)$ is instantiated with the terms a and b for every subterm of the form $a +_b b$ occurring in F;
- $(A2)$ is instantiated with the terms c and a for every subterm of the form $c \times_b a$ occurring in F;
- $(A3)$ is instantiated with the terms a and b for every subterm of the form $a \leq_b b$ occurring in F;
- $(A4)$ is instantiated with the term a for every bounded integer subterm occurring in F;
- $(A5)$ is instantiated with the term c for every subterm of the form $(c)_b$ occurring in F;
- $(A6')$ is instantiated with the term a for every bounded integer subterm occurring in F.

Isabelle implementation

In TSMT_Bounded file:

```
lemma subT[when "a - b", tsmt ubound]:
  "uint a - uint b ≥ 0 ⟹ uint a - uint b ≤ U ⟹ uint (a - b) = uint a - uint b"
  for a b :: "'a word" unfolding U_def using U' len_of uint_sub_if'[of a b] by unat_arith
lemma leT[when "a ≤ b", tsmt ubound]: "a ≤ b ⟷ uint a ≤ uint b" for a b :: "'a word" by
    (rule word_le_def)
lemma ltT[when "a < b", tsmt ubound]: "a < b ⟷ uint a < uint b" for a b :: "'a word" by
    (rule word_less_def)
lemma minT[when "min a b", tsmt ubound]:
  "a ≤ b ⟹ min a b = a" "b ≤ a ⟹ min a b = b" for a b :: "'a word" by
    - (rule min_absorb1 | rule min_absorb2 | assumption)+
lemma maxT[when "max a b", tsmt ubound]:
  "a ≥ b ⟹ max a b = a" "b ≥ a ⟹ max a b = b" for a b :: "'a word" by
    - (rule max_absorb1 | rule max_absorb2 | assumption)+
lemma uintT[when "uint a", tsmt ubound]:
  "uint a ≥ 0" "uint a ≤ U" for a :: "'a word" unfolding U_def using U' len_of by
    - (simp, uint_arith)
lemma word_of_intT[when "uint a", tsmt ubound]:
  "word_of_int (uint a) = a" for a :: "'a word" by (rule word_of_int_uint)
```

Theoretical example

$$F = (25)_b \leq_b a \wedge -1 \times_b a +_b (25)_b \neq (0)_b$$

$$\begin{aligned}
\underline{-1 \times_b a} +_b \underline{(25)_b} \in F \Rightarrow & \; -25 \leq v(-1 \times_b a) + v\big((25)_b\big) \leq 25 \Longrightarrow v\big(-1 \times_b a +_b (25)_b\big) = v(-1 \times_b a) + v\big((25)_b\big), && (A1)\\
\underline{-1} \times_b \underline{a} \in F \Rightarrow & \; -25 \leq -1 \times v(a) \leq 25 \Longrightarrow v(-1 \times_b a) = -1 \times v(a), && (A2)\\
\underline{(25)_b} \leq_b \underline{a} \in F \Rightarrow & \; (25)_b \leq_b a \Longleftrightarrow v\big((25)_b)\big) \leq v(a), && (A3)\\
a \in F \Rightarrow & \; -25 \leq v(a) \leq 25, && (A4)\\
(\underline{25})_b \in F \Rightarrow & \; -25 \leq 25 \leq 25 \Longrightarrow v\big((25)_b\big) = 25, && (A5)\\
-1 \times_b a +_b (25)_b \in F \Rightarrow & \; \Big(v\big(-1 \times_b a +_b (25)_b\big)\Big)_b = -1 \times_b a +_b (25)_b. && (A6')
\end{aligned}$$

$$F^* = F \wedge \bigwedge F^+ \wedge \bigwedge F^\times \wedge \bigwedge F^\leq \wedge \bigwedge F^\in \wedge \bigwedge F^m \wedge \bigwedge F^v$$

Theoretical example

The formula F is unsatisfiable.

Proof.

In the proof we use only the instantiations shown in previous frame. From the instantiation of $(A5)$ we have $v\big((25)_b\big) = 25$. Then from the instantiation of $(A3)$ we have $25 = v\big((25)_b)\big) \leq v(a)$ and also from the instantiation of $(A4)$ have $v(a) \leq 25$. Thus $v(a) = 25$. Now from the instantiation of $(A2)$ we have $v(-1 \times_b a) = -1 \times v(a) = -25$, then from $(A1)$ have $v\big(-1 \times_b a +_b (25)_b\big) = v(-1 \times_b a) + v\big((25)_b\big) = -25 + 25 = 0$. Finally, from the instantiation of $(A6')$ we have $-1 \times_b a +_b (25)_b = \big(v\big(-1 \times_b a +_b (25)_b\big)\big)_b = (0)_b$. This is in contradiction with the second conjunct $-1 \times_b a +_b (25)_b \neq (0)_b$ of the formula F. Thus F is unsatisfiable. □

Theoretical example: Isabelle analog

In TSMT_Bounded_Example file:

```
lemma simple_example:
  "255 ≤ a ⟹ 1 * a - 255 = 0" for a :: ubound8 using [[smt_trace]] by  (tsmt ubound)
```

```
theorem simple_example: 255 ≤ ?a ⟹ 1 * ?a - 255 = 0
SMT: Assertions:
       255 ≤ a
       True
       U = 255
       1 * a = a
       255 ≤ U ⟶ uint 255 = 255
       uint 0 = 0
       uint 1 = 1
       0 ≤ uint a - uint 255 ∧ uint a - uint 255 ≤ U ⟶ uint (a - 255) = uint a - uint 255
       (255 ≤ a) = (uint 255 ≤ uint a)
       0 ≤ uint 1
       0 ≤ uint a
       0 ≤ uint 255
       0 ≤ uint 0
       0 ≤ uint (a - 255)
       uint 1 ≤ U
       uint a ≤ U
       uint 255 ≤ U
       uint 0 ≤ U
       uint (a - 255) ≤ U
       word_of_int (uint 1) = 1
       word_of_int (uint a) = a
       word_of_int (uint 255) = 255
       word_of_int (uint 0) = 0
       word_of_int (uint (a - 255)) = a - 255
       1 * a - 255 ≠ 0
```

Formal proof

Theorem

Every ground formula F in BLIA is satisfiable if and only if its translation F^* is satisfiable in QF_UFLIA.

A proof of this completeness theorem consists of 14 lemmas. And it is based on idea about equality of some model M in BLIA and model R in QF_UFLIA. Therefore there is an identical truth in R, it is also satisfied by M.

Formal proof

At this point we introduce some notation. We denote the image of a subset X of the domain A under function $f : A \to B$ as $f[X]$. We also introduce the following shorthand:

$$\{f(x) \mid P(x)\} \equiv f\big[\{x \mid P(x)\}\big],$$

And define the following two sets:

$$\begin{array}{rcl} B^R & \equiv & \left\{t^R \mid v(t) \in F^*\right\}, \\ C^R & \equiv & \left\{n^R \mid (n)_b \in F^*\right\}, \end{array}$$

Free bounded integer terms are denoted with letters t and u, free integer terms are denoted with k or n, while arbitrarily chosen bounded integer values are debited with letters a and b, and unbounded integer values are denoted with x, y or c.

Formal proof

Let's define main lemmas.

Lemma

The reconstructed model M is well-defined.

Formal proof

We showed that the functions defined as shown in the Figure do indeed map any combination of their arguments taken from the corresponding domains to the elements from the corresponding ranges.

$$
\begin{aligned}
\mathbb{Z}_b^M &= B^R \cup \mathbb{Z}_b', \\
v^M(a) &= \begin{cases} v^R(a), & a \in B^R, \\ v'(a), & a \notin B^R, \end{cases} \\
(c)_b^M &= \begin{cases} (c)_b^R, & c \in C^R, \\ (c)_b', & c \in [L, U] \setminus C^R, \\ \epsilon \mathbb{Z}_b^M, & c \notin C^R \cup [L, U], \end{cases} \\
a +_b^M b &= \begin{cases} a +_b^R b, & (a, b) \in \left\{ (t^R, u^R) \mid t +_b u \in F^* \right\}, \\ \left(v^M(a) + v^M(b) \right)_b^M, & \\ & (a, b) \notin \left\{ (t^R, u^R) \mid t +_b u \in F^* \right\}, \\ & \quad L \leq v^M(a) + v^M(b) \leq U, \\ (0)_b, & \text{otherwise}, \end{cases} \\
c \times_b^M a &= \begin{cases} c \times_b^R a, & (c, a) \in \left\{ (n, t^R) \mid n \times_b t \in F^* \right\}, \\ \left(c \times v^M(a) \right)_b^M, & \\ & (c, a) \notin \left\{ (n, t^R) \mid n \times_b t \in F^* \right\}, \\ & \quad L \leq c \times v^M(a) \leq U, \\ (0)_b, & \text{otherwise}, \end{cases} \\
a \leq_b^M b &= v^M(a) \leq v^M(b).
\end{aligned}
$$

Formal proof

Lemma

The axioms (A4), (A5) and (A6') of the BLIA theory hold in the model M.

Lemma

The axioms (A1), (A2) and (A3) of the BLIA theory hold in the model M.

Lemma

The model M can be extended with uninterpreted constants occurring in F so that for any subterm $t \in F$ its interpretations in the model M and the realization R coincide i. e. $t^M = t^R$.

Formal proof: auxiliary lemmas

Lemma

A term of the form $v(t)$ occurs in F^* if and only if the bounded integer term t occurs in F.

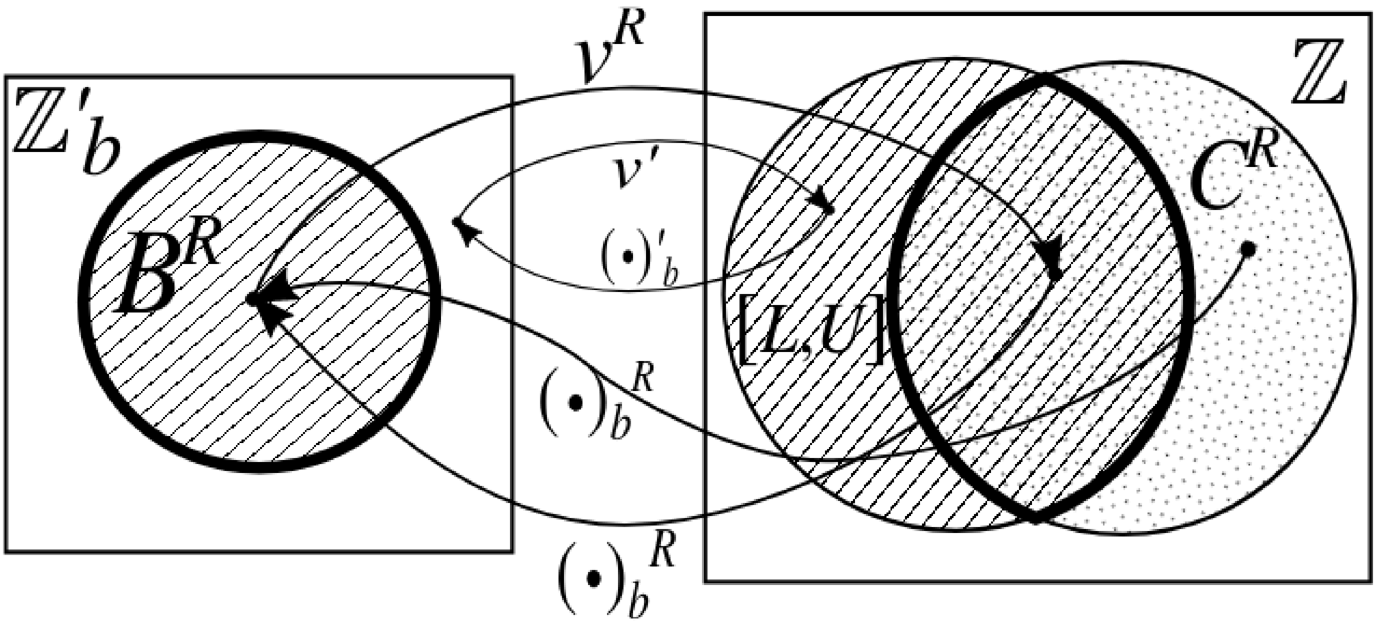
Lemma

A term of the form $\big(v(t)\big)_b$ occurs in F^* if and only if the bounded integer term t occurs in F.

Lemma

A term of the form $(n)_b$ occurs in F^* if and only if either it already occurs in F or the term n has the form $v(t)$ where t occurs in F.

Formal proof: auxiliary lemmas



Formal proof: auxiliary lemmas

Lemma

$v^R[B^R] \subseteq C^R \cap [L, U]$.

Lemma

v^R is a bijection between B^R and $C^R \cap [L, U]$ and $(\cdot)^R_b$ is its unique inverse.

Lemma

$(\cdot)^R_b[C^R] \subseteq B^R$

Counterexamples in Isabelle

Questions