Abstraction Domain with Congruence

Kihwan Kim (kimkihwan@kaist.ac.kr)

October 2021

0 Notations

* $\mathcal{P}(S)$ means power set of S
* 'iff' means 'if and only if'
* For two number n and m, $n \mid m$ means $n \equiv 0 \mod m$
* $n\%m = r$ iff $n \equiv r \mod m$ and $0 \leq r < m$
* gcd stands for greatest common divisor
* lcm stands for least common multiple.

Definition 1 (Greatest Common Divisor) *For $n, m \in \mathbb{Z}$,*

$$gcd(n, m) = \max\{k \in \mathbb{N} \mid n \equiv m \equiv 0 \mod k\}$$

Definition 2 (Least Common Multiple) *For $n, m \in \mathbb{Z} \setminus \{0\}$,*

$$lcm(n, m) = \min\{k \in \mathbb{N} \setminus \{0\} \mid \exists x, y \in \mathbb{Z} \setminus \{0\}, nx = my = k\}$$

1 Target Concrete State

This abstraction domain is targeting the program language which values are all integer number (we do not consider overflow or underflow). It doesn't have any kind of high-level structure such as functions or objects. Formally, set of all values $\mathbb{V}$ is equal to $\mathbb{Z}$, and let $\mathbb{S}$ be a set of all possible variable names.
We will consider only memory state. Memory state is a mapping from variable name to its value.

2 Definition of Abstraction Domain

The syntax of abstraction of value is pair of two natural number. Formally, abstraction domain of value $\mathbb{A}_v = \mathbb{Z} \times \mathbb{N}$
Second element of the pair means divisor and first element means remainder. For convenience, we will restrict second element to be non-negative. First element (remainder) is always integer, and if second element is non-zero, then the

absolute value of first element should be smaller than the second element.
We can think concretization function $\gamma : \mathbb{A}_v \to \mathbb{V}$.

$$\forall n \in \mathbb{Z}, \forall m \in \mathbb{N}, \gamma((n,m)) = \{mx + n \mid x \in \mathbb{Z}\}$$

Now we need to construct poset (partially ordered set) to make our abstraction domain worth it. For interval, we use inclusion for partial ordering, so we will try it again to this abstraction domain.

Definition 3 (Galois Connection) *Galois connection of the abstraction domain is*

$$(\mathbb{C}, \subseteq) \underset{\gamma}{\overset{\alpha}{\leftrightarrows}} (\mathbb{A}, \sqsubseteq)$$

where $\mathbb{C} = \mathcal{P}(\mathbb{S} \times \mathbb{V})$ *and* $\mathbb{A} = \mathbb{S} \mapsto \mathbb{A}_v$.

3 Definition of Functions

3.1 Typical Operators

For this subsection, we will fix two arbitrary abstract elements $a_1 = (r_1, d_1)$ and $a_2 = (r_2, d_2)$.

Definition 4 (Inclusion Test)

$$a_1 \sqsubseteq a_2 \text{ iff } d_1 \mid d_2 \wedge r_1 \equiv r_2 \mod d_2$$

Definition 5 (Abstract Union)

$$a_1 \sqcup^\sharp a_2 = (k, D) \text{ where } D = gcd(gcd(d_1, d_2), r_1 - r_2) \text{ and } k = \begin{cases} r_1 \% D & \text{if } D \neq 0 \\ r_1 & \text{otherwise} \end{cases}$$

Definition 6 (Abstract Joint for nonzero) *Assume* $d_1 \neq 0$ *and* $d_2 \neq 0$,

$$a_1 \sqcap^\sharp a_2 = \begin{cases} (k, lcm(d_1, d_2)) & \text{if } r_1 \equiv r_2 \equiv k \mod gcd(d_1, d_2) \\ \bot & \text{otherwise} \end{cases}$$

Definition 7 (Abstract Joint for zero) *W/o losing of generality, fix* $d_1 = 0$

$$a_1 \sqcap^\sharp a_2 = \begin{cases} (r_1, 0) & \text{if } a_1 \sqsubseteq a_2 \\ \bot & \text{otherwise} \end{cases}$$

We need abstract joint because it is used in implementation of 'filtering'. For scalability, while analyzing conditional command, filtering function joint proper mask abstraction with given abstraction.

Definition 8 (Widening)

$$a_1 \nabla a_2 = \begin{cases} a_1 & \text{if} \quad d_2 \mid d_1 \wedge (r_2 - r_1) \mid d_1 \\ \top & \text{otherwise} \end{cases}$$

Widening is a scheme that can analyze iteration even abstraction domain is not a finite lattice. First, set a finite logical constraints, and widening loosen the constraint or terminate, for each iteration. Logical constraint of abstraction domain with Congruence is only one, which is $n \equiv r \mod d$. So we just check that the states, after computing an iteration, still satisfies constraint $n \equiv r \mod d$.

3.2 Transfer Functions

First we will define abstraction of scalar expressions

Definition 9 (Abstraction of Scalar Constant) $[n]^\sharp(M^\sharp) = (n, 0)$

Definition 10 (Abstraction of variable) $[x]^\sharp(M^\sharp) = M^\sharp(x)$

Definition 11 (Abstraction of Addition and Subtraction) *Let* r_1, r_2, d_1, d_2 *for given* E_1 *and* E_2 *such that* $[E_1]^\sharp(M^\sharp) = (r_1, d_1)$, $[E_2]^\sharp(M^\sharp) = (r_2, d_2)$, *and* $gcd(d_1, d_2) = D$,

$$[E_1 + E_2]^\sharp(M^\sharp) = ((r_1 + r_2)\%D, D)$$
$$[E_1 - E_2]^\sharp(M^\sharp) = ((r_1 - r_2)\%D, D)$$

Second, we will define abstraction of program commands

Definition 12 (Abstraction of Skip) $[skip]^\sharp(M^\sharp) = M^\sharp$

Definition 13 (Abstraction of Sequence) $[C_1; C_2]^\sharp(M^\sharp) = [C_2]^\sharp([C_1]^\sharp(M^\sharp))$

Definition 14 (Abstraction of Assignment and Input) *Recall the abstraction of scalar expressions.*

$$[x := E]^\sharp(M^\sharp) = M^\sharp[x \mapsto [E]^\sharp(M^\sharp)]$$
$$[\text{Input}(x)]^\sharp(M^\sharp) = M^\sharp[x \mapsto \top]$$

Definition 15 (Abstraction of Conditional Statements)

$$[\text{if}(B)\text{then}\{C_t\}\text{else}\{C_e\}]^\sharp(M^\sharp) = [C_t]^\sharp(filter_B(M^\sharp)) \sqcup^\sharp [C_e]^\sharp(filter_{\neg B}(M^\sharp))$$

Definition 16 (Abstraction of Loop) *Recall the abstract iteration function* abs_iter *which use widening,*

$$[\text{While}(B)\{C\}]^\sharp(M^\sharp) = filter_{\neg B}(\text{abs\_iter}([C]^\sharp \circ filter_B, M^\sharp))$$

4 Proofs

W.I.P.