Algorithms for Symbolic Abstraction

Author: Jiawei Ren

Supervisor: Yulei Sui

Algorithms for Symbolic Abstraction

Thakur, A., Elder, M., Reps, T. (2012). Bilateral Algorithms for Symbolic Abstraction. In: Miné, A., Schmidt, D. (eds) Static Analysis. SAS 2012. Lecture Notes in Computer Science, vol 7460. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-33125-1_10

Peisen Yao, Qingkai Shi, Heqing Huang, and Charles Zhang. 2021. Program analysis via efficient symbolic abstraction. Proc. ACM Program. Lang. 5, OOPSLA, Article 118 (October 2021), 32 pages. https://doi.org/10.1145/3485495

Abstract Interpretation

Definition (Abstraction Interpretation). The elements of the *abstract domain* $\mathbb{A}$ are *abstract values* that approximates a set of *concrete values*, i.e., the values that a variable can take in the *concrete domain* $\mathbb{C}$ during program execution.

Example.

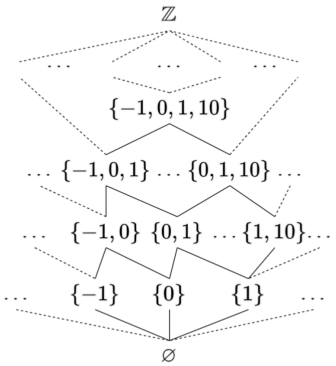
$$c = \{[x \rightarrow 2, y \rightarrow 200], [x \rightarrow 5, y \rightarrow 120], [x \rightarrow 10, y \rightarrow 20]\}$$
$$a = [x \rightarrow [2, 10], y \rightarrow [20, 200]]$$

Lattice

Definition (Complete Lattice). A partially ordered set $(\mathbb{L}, \leq)$ is said to be a complete lattice if every subset M of $\mathbb{L}$ has both a greatest lower bound (also called meet, denoted by $\sqcap M$) and a least upper bound (also called join, denoted by $\sqcup M$) in $(\mathbb{L}, \leq)$. A complete lattice has a greatest element, denoted by $\top$, and a least element, denoted by $\bot$, such that $\bot \leq m \leq \top$, for each $m \in \mathbb{L}$.

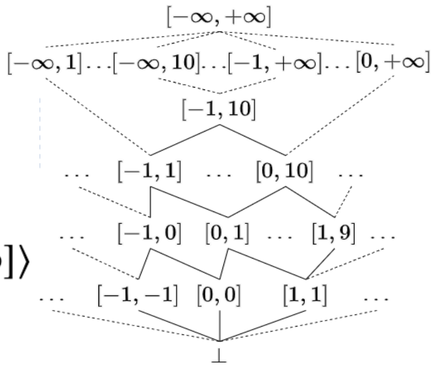
Powerset

$\mathfrak{C} = \langle \mathbb{C}, \subseteq, \cap, \cup, \varnothing, \mathbb{Z} \rangle$



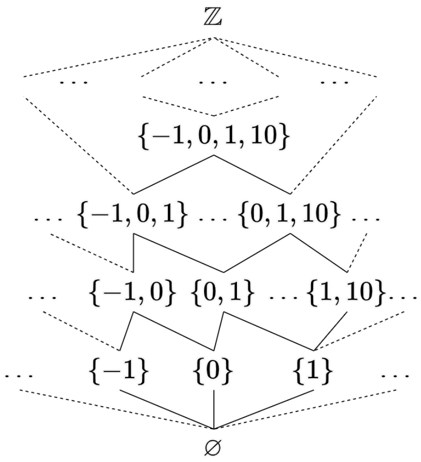
Interval

$\mathfrak{C} = \langle \mathbb{C}, \subseteq, \cap, \cup, \bot, [-\infty, +\infty] \rangle$



Concrete Domain

Definition (Concrete Domain). We use $\mathbb{S}$ to represent the set of concrete values that a program variable can have (e.g., integers, floats and strings) in any possible concrete execution. The concrete domain can be represented as $\mathbb{C} = \mathscr{P}(\mathbb{S})$, which is the powerset of $\mathbb{S}$ equipped with the powerset lattice defined as $\mathfrak{C} = \langle \mathbb{C}, \subseteq, \cap, \cup, \varnothing, \mathbb{S} \rangle$, where the partial order is $\subseteq$, and $\cap$ and $\cup$ represent the meet and join operations respectively, and $\varnothing$ and $\mathbb{S}$ are the unique least and greatest elements of $\mathbb{C}$.

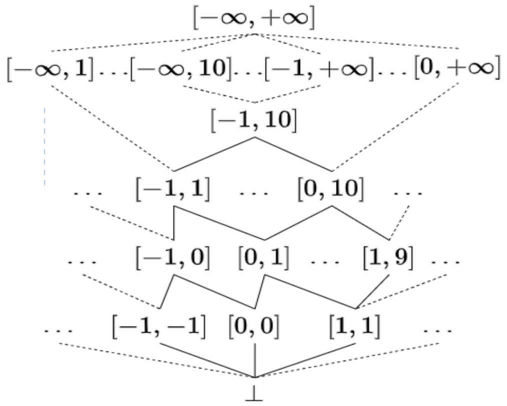


Abstract Domain

Definition (Abstract Domain). The abstract domain $\mathbb{A}$ is an over-approximate abstraction of $\mathbb{C}$ with a concretization function $\gamma \in \mathbb{A} \rightarrow \mathbb{C}$ based on a partial order $\sqsubseteq$ over $\mathbb{A}$ such that $\forall a, a' \in \mathbb{A}, a \sqsubseteq a' \Leftrightarrow \gamma(a) \subseteq \gamma(a')$. The partial order relations of an abstract domain $\mathbb{A}$ form a lattice $\mathfrak{A} = \langle \mathbb{A}, \sqsubseteq, \sqcap, \sqcup, \bot, \top \rangle$, where $\sqcap$ and $\sqcup$ are the meet and join operations, and $\bot_{\mathbb{A}}$ and $\top_{\mathbb{A}}$ are unique least and greatest elements of $\mathbb{A}$.

Interval domain: $\mathfrak{C} = \langle \mathbb{C}, \subseteq, \cap, \cup, \bot, [-\infty, +\infty] \rangle$

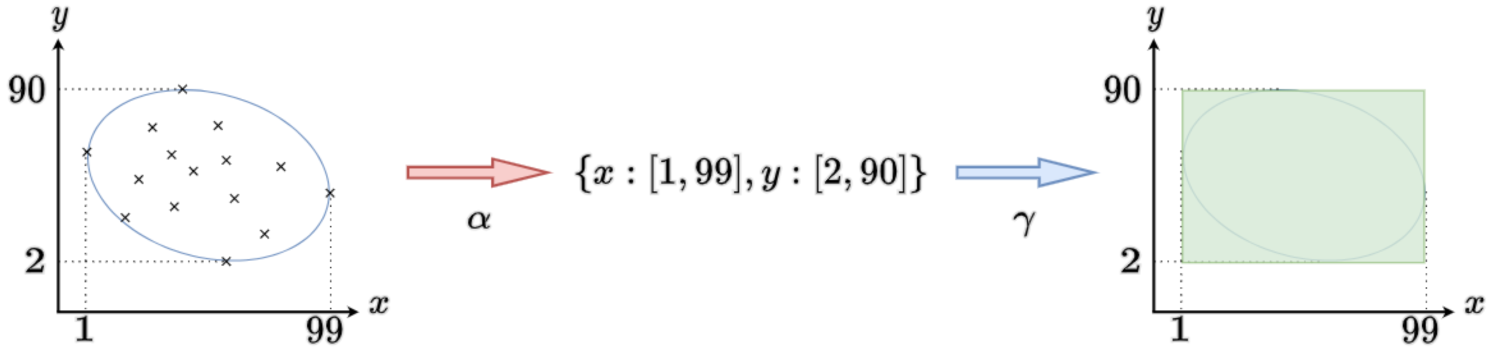
$$a_1 = [a, b], a_2 = [c, d] \rightarrow a_1 \cap a_2 = [b, c], a_1 \cup a_2 = [a, d]$$



Galois connection

Galois Connection expresses a two-way connections between $\mathfrak{A}$ and $\mathfrak{C}$ using
(1) an abstraction function $\alpha : \mathbb{C} \to \mathbb{A}$ mapping a set of concrete values to its abstract interpretation
(2) a concretization function $\gamma : \mathbb{A} \to \mathbb{C}$ mapping a set of abstract values to concrete ones
(3) satisfying:

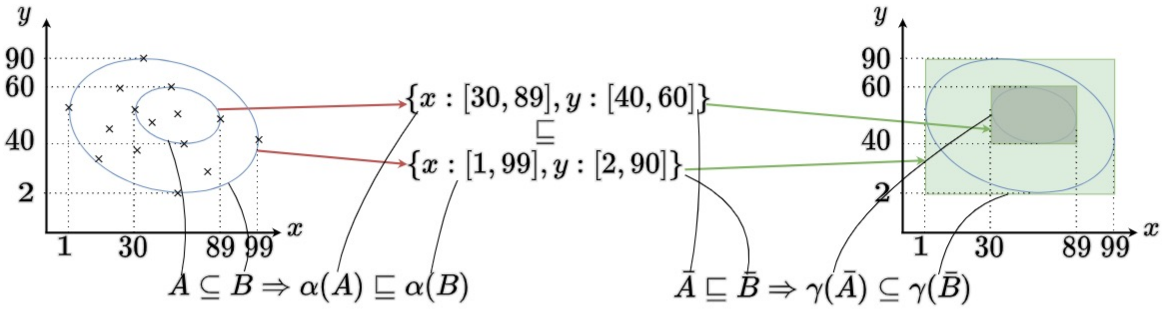
$$\alpha(c) \sqsubseteq_{\mathbb{A}} a \Leftrightarrow c \sqsubseteq_{\mathbb{C}} \gamma(a)$$



Galois connection

Galois Connection expresses a two-way connections between $\mathfrak{A}$ and $\mathfrak{C}$ using
(1) an abstraction function $\alpha : \mathbb{C} \longrightarrow \mathbb{A}$ mapping a set of concrete values to its abstract interpretation
(2) a concretization function $\gamma : \mathbb{A} \longrightarrow \mathbb{C}$ mapping a set of abstract values to concrete ones
(3) satisfying:

$$\alpha(c) \sqsubseteq_{\mathbb{A}} a \Leftrightarrow c \sqsubseteq_{\mathbb{C}} \gamma(a)$$



Galois connection

Properties of Galois connection:

(1) γ uniquely determines α by

$$\alpha(c) = \sqcap\{a | c \sqsubseteq_{\mathbb{C}} \gamma(a)\}$$

(2) α is completely additive; that is, given $C \in \mathbb{C}$,

$$\alpha(\sqcup C) = \sqcup\{\alpha(c) | c \in C\}$$

Definition (Representation Function). The representation function β maps a singleton concrete state σ such that $\sigma \in \mathbb{C}$ to the least value in $\mathbb{A}$ that over-approximates $\{\sigma\}$.

In other words, β returns the abstraction of a singleton concrete state; i.e.,

$$\beta(\sigma) = \alpha(\{\sigma\})$$

Example in interval domain.

$$\beta([x \to 1, y \to 2]) = [x \to [1, 1], y \to [2, 2]]$$

Moving from $\mathbb{A}$ to L

Definition (Symbolic Concretization). Given an abstract value $A \in \mathbb{A}$, the symbolic concretization of A, denoted by $\hat{\gamma}(A)$, maps A to a formula $\hat{\gamma}(A)$ such that A and $\hat{\gamma}(A)$ represent the same set of concrete states (i.e., $\gamma(A) = [\![\gamma(\hat{A})]\!]$).

Example in interval domain.

$$a = [x \longrightarrow [2, 10], y \longrightarrow [20, 200]]$$

$$\hat{\gamma}(a) = 2 \leq x \leq 10 \wedge 20 \leq y \leq 200$$

Moving from L to $\mathbb{A}$

Definition (Symbolic Abstraction). Given $\varphi \in L$, the symbolic abstraction of φ, denoted by $\hat{\alpha}(\varphi)$, maps φ to the *best value* in $\mathbb{A}$ that over-approximates $[\![\varphi]\!]$ (i.e., $\hat{\alpha}(\varphi) = \alpha([\![\varphi]\!])$).

Example in interval domain.

$$a = [x \longrightarrow [0,1], y \longrightarrow [0,1], z \longrightarrow [-\infty, +\infty]], \varphi = (x = y \wedge z = x - y)$$

Interval subtraction:

$$z = [x_{low} - y_{high}, x_{high} - y_{low}] = [-1, 1]$$

Symbolic abstraction:

$$\varphi^{'} = \hat{\gamma}(a) \wedge \varphi = (0 \leq x \leq 1) \wedge (0 \leq y \leq 1) \wedge (x = y) \wedge (z = x - y)$$

$$\hat{\alpha}(\varphi^{'}) = [x \longrightarrow [0,1], y \longrightarrow [0,1], z \longrightarrow [0,0]]$$

Two theorems

Theorem 1. $\hat{\alpha}(\varphi) = \bigsqcup\{\beta(S) | S \models \varphi\}$

Theorem 2. $\hat{\alpha}(\varphi) = \bigsqcap\{\alpha | \varphi \models \hat{\gamma}(a)\}$

double turnstile: if every sentence on the left is true, the sentence on the right must be true

Symbolic abstraction algorithms

- The RSY algorithm: a framework for computing $\hat{\alpha}$ that applies to any logic and abstract domain that satisfies certain conditions.
- The KS algorithm: an algorithm for computing $\hat{\alpha}$ that only applies to QFBV logic and the domain of affine equalities. (smaller query and faster)
- The Bilateral algorithm: combining the advantages of RSY and KS algorithms and resilient to timeout.

Thakur, A., Elder, M., Reps, T. (2012). Bilateral Algorithms for Symbolic Abstraction. In: Miné, A., Schmidt, D. (eds) Static Analysis. SAS 2012. Lecture Notes in Computer Science, vol 7460. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-33125-1_10

RSY algorithm

Theorem 1. $\hat{\alpha}(\varphi) = \sqcup\{\beta(S) | S \models \varphi\}$

$$A_0 = \bot$$

$$A_i = A_{i-1} \sqcup \beta(S_i), S_i \models \varphi, 1 \leq i \leq k \quad \text{(possibly no progress)}$$

$$A_i = A_{i-1} \sqcup \beta(S_i), S_i \models \varphi \wedge \neg\hat{\gamma}(A_{i-1}), 1 \leq i \leq k \quad \text{(progress is guaranteed)}$$

$$\bot = A_0 \sqsubset A_1 \sqsubset A_2 \sqsubset \cdots \sqsubset A_{k-1} \sqsubset A_k = \hat{\alpha}(\varphi)$$

Algorithm 6: $\widetilde{\alpha}^{\uparrow}_{\mathrm{RSY}}\langle \mathcal{L}, \mathcal{A} \rangle(\varphi)$

1 $lower \leftarrow \bot$
2
3 **while** `true` **do**
4
5   $S \leftarrow$ `Model`$(\varphi \wedge \neg\widehat{\gamma}(lower))$
6   **if** S **is** TimeOut **then**
7   **return** $\top$
8   **else if** S **is** None **then**
9   **break** // $\varphi \Rightarrow \widehat{\gamma}(lower)$
10   **else** // $S \not\models \widehat{\gamma}(lower)$
11     $lower \leftarrow lower \sqcup \beta(S)$
12 $ans \leftarrow lower$
13 **return** ans

Sampling and Generalization:

Sampling at Line 5 and update lower at Line 11

If Timeout, simply return $\top$ (Line 7)

If no solution (Line 8), $lower = \hat{\alpha}(\varphi)$

RSY algorithm

Example in interval domain.

$$lower = \bot, a = [x \to [0,1], y \to [0,1], z \to [-\infty, +\infty]], \varphi = (x = y \wedge z = x - y)$$

First round:

$$\varphi = (0 \le x \le 1) \wedge (0 \le y \le 1) \wedge (x = y) \wedge (z = x - y)$$
$$S = \{x \to 0, y \to 0, z \to 0\}$$
$$\beta(S) = [x \to [0,0], y \to [0,0], z \to [0,0]]$$
$$lower = [x \to [0,0], y \to [0,0], z \to [0,0]]$$

Second round:

$$\varphi \wedge \neg\hat{\gamma}(lower) = (0 \le x \le 1) \wedge (0 \le y \le 1) \wedge (x = y) \wedge (z = x - y) \wedge \neg(x = 0 \wedge y = 0 \wedge z = 0)$$
$$S = \{x \to 1, y \to 1, z \to 0\}$$
$$\beta(S) = [x \to [1,1], y \to [1,1], z \to [0,0]]$$
$$lower = [x \to [0,1], y \to [0,1], z \to [0,0]]$$

Third round:

$$\varphi \wedge \neg\hat{\gamma}(lower) = (0 \le x \le 1) \wedge (0 \le y \le 1) \wedge (x = y) \wedge (z = x - y) \wedge \neg(0 \le x \le 1 \wedge 1 \le y \le 1 \wedge z = 0)$$
$$S = unsat$$

Algorithm 6: $\widetilde{\alpha}^{\uparrow}_{\mathrm{RSY}}\langle\mathcal{L}, \mathcal{A}\rangle(\varphi)$

$lower \leftarrow \bot$

while true **do**

  $S \leftarrow \mathtt{Model}(\varphi \wedge \neg\widehat{\gamma}(lower))$
  if S **is** TimeOut **then**
    return $\top$
  else if S **is** None **then**
    break // $\varphi \Rightarrow \widehat{\gamma}(lower)$
  else // $S \not\models \widehat{\gamma}(lower)$
    $lower \leftarrow lower \sqcup \beta(S)$

$ans \leftarrow lower$
return ans

KS algorithm

Algorithm 6: $\widetilde{\alpha}^{\uparrow}_{\mathrm{RSY}}\langle\mathcal{L},\mathcal{A}\rangle(\varphi)$

$lower \leftarrow \bot$
while true **do**
 $S \leftarrow \mathtt{Model}(\varphi \wedge \neg\widehat{\gamma}(lower))$
 if S **is** TimeOut **then**
  return $\top$
 else if S **is** None **then**
  break   // $\varphi \Rightarrow \widehat{\gamma}(lower)$
 else   // $S \not\models \widehat{\gamma}(lower)$
  $lower \leftarrow lower \sqcup \beta(S)$
$ans \leftarrow lower$
return ans

Algorithm 7: $\widetilde{\alpha}^{\uparrow}_{\mathrm{KS}}(\varphi)$

1 $lower \leftarrow \bot$
2 $i \leftarrow 1$
3 **while** $i \leq \mathtt{rows}(lower)$ **do**
4 $p \leftarrow \mathtt{Row}(lower, -i)$ // $p \sqsupseteq lower$
5 $S \leftarrow \mathtt{Model}(\varphi \wedge \neg\widehat{\gamma}(p))$
6 **if** S **is** TimeOut **then**
7   **return** $\top$
8 **else if** S **is** None **then**
9   $i \leftarrow i + 1$ // $\varphi \Rightarrow \widehat{\gamma}(p)$
10 **else**   // $S \not\models \widehat{\gamma}(p)$
11   $lower \leftarrow lower \sqcup \beta(S)$
12 $ans \leftarrow lower$
13 **return** ans

An abstract value in the affine-equalities domain is a conjunction of affine equalities, which can be represented in a normal form as a matrix in which each row expresses a non-redundant affine equality. (Rows are 0-indexed.) Given a matrix m, $rows(m)$ returns the number of rows of m (as in line 3 in KS Algorithm), and $Row(m, i)$, for $1 \leq i \leq rows(m)$, returns row $(rows(m) - i)$ of m (as in line 4 in KS Algorithm).

KS algorithm

Algorithm 6: $\tilde{\alpha}^{\uparrow}_{\mathrm{RSY}}\langle\mathcal{L},\mathcal{A}\rangle(\varphi)$

$lower \leftarrow \bot$

while true **do**

  $S \leftarrow \mathtt{Model}(\varphi \wedge \neg\widehat{\gamma}(lower))$

  if S **is** TimeOut **then**

    return $\top$

  else if S **is** None **then**

    break // $\varphi \Rightarrow \widehat{\gamma}(lower)$

  else // $S \not\models \widehat{\gamma}(lower)$

    $lower \leftarrow lower \sqcup \beta(S)$

$ans \leftarrow lower$

return ans

Algorithm 7: $\tilde{\alpha}^{\uparrow}_{\mathrm{KS}}(\varphi)$

$lower \leftarrow \bot$

$i \leftarrow 1$

while $i \leq \mathtt{rows}(lower)$ **do**

  $p \leftarrow \mathtt{Row}(lower, -i)$ // $p \sqsupseteq lower$

  $S \leftarrow \mathtt{Model}(\varphi \wedge \neg\widehat{\gamma}(p))$

  if S **is** TimeOut **then**

    return $\top$

  else if S **is** None **then**

    $i \leftarrow i+1$ // $\varphi \Rightarrow \widehat{\gamma}(p)$

  else // $S \not\models \widehat{\gamma}(p)$

    $lower \leftarrow lower \sqcup \beta(S)$

$ans \leftarrow lower$

return ans

Comparison:

RSY algorithm uses all of lower to construct the query and KS algorithm uses a single column from lower.

- KS has a larger number of queries than RSY
- each individual query issued by KS is smaller than RSY

Empirical study shows that KS algorithm is faster than RSY algorithm.

Bilateral algorithm

Algorithm	Parametric	Resilient	Parsimonious
RSY	✓	×	×
KS	×	×	✓
Bilateral	✓	✓	✓

- Parametric: applicable to any abstract domain that satisfies certain conditions
- Resilient: (non-trivial) over-approximation when timeout
- Parsimonious: uses a successive-approximation algorithm that is parsimonious in its use of the decision procedure (similar to the KS algorithm)

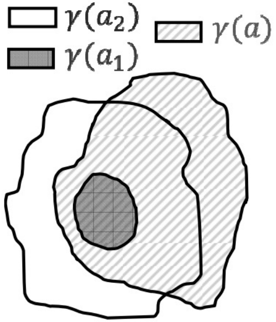
Bilateral algorithm

Definition (Abstract Consequence). An operation AbstractConsequence$(\cdot, \cdot)$ is an acceptable abstract-consequence operation iff for all $a_1, a_2 \in \mathbb{A}$ such that $a_1 \sqsubset a_2$, $a = \text{AbstractConsequence}(a_1, a_2)$ implies $a_1 \sqsubseteq a$ and $a2 \not\sqsubseteq a$.

Algorithm 12: `AbstractConsequence`(a_1, a_2) for conjunctive domains

if $a_1 = \bot$ **then return** $\bot$

Let $\Psi \subseteq \Phi$ be the set of formulas such that $\widehat{\gamma}(a_1) = \bigwedge \Psi$
foreach $\psi \in \Psi$ **do**
    $a \leftarrow \mu\widehat{\alpha}(\psi)$
    if $a \not\sqsupseteq a_2$ **then return** a



A simple Example from Github.

$$\text{upper} = [x \rightarrow [0, 5], y \rightarrow [1, 2], z \rightarrow [0, 1]]$$

$$\text{lower} = [x \rightarrow [0, 1], y \rightarrow [1, 2], z \rightarrow [0, 1]]$$

$$\text{a} = [x \rightarrow [0, 1], y \rightarrow [-\infty, +\infty], z \rightarrow [-\infty, +\infty]]$$

Bilateral algorithm

Algorithm 11: $\widetilde{\alpha}^{\ddagger}\langle\mathcal{L},\mathcal{A}\rangle(\varphi)$

1 $upper \leftarrow \top$
2 $lower \leftarrow \bot$
3 **while** $lower \neq upper \wedge$ ResourcesLeft **do**
// $lower \sqsubsetneq upper$
4     $p \leftarrow \texttt{AbstractConsequence}(lower, upper)$
// $p \sqsupseteq lower, p \not\sqsupseteq upper$
5     $S \leftarrow \texttt{Model}(\varphi \wedge \neg\widehat{\gamma}(p))$
6   **if** S **is** TimeOut **then**
7   **return** $upper$
8   **else if** S **is** None **then** // $\varphi \Rightarrow \widehat{\gamma}(p)$
9         $upper \leftarrow upper \sqcap p$
10   **else** // $S \not\models \widehat{\gamma}(p)$
11         $lower \leftarrow lower \sqcup \beta(S)$
12 $ans \leftarrow upper$
13 **return** ans

Line 6-7: Timeout return (non-trivial) over-approximation

Line 8-9: update *upper*

Line 10-11: update *lower*

Theorem 1. $\hat{\alpha}(\varphi) = \sqcup\{\beta(S) | S \models \varphi\}$

Theorem 2. $\hat{\alpha}(\varphi) = \sqcap\{\alpha | \varphi \models \hat{\gamma}(a)\}$

Paper 2

Peisen Yao, Qingkai Shi, Heqing Huang, and Charles Zhang. 2021. Program analysis via efficient symbolic abstraction. Proc. ACM Program. Lang. 5, OOPSLA, Article 118 (October 2021), 32 pages. https://doi.org/10.1145/3485495

Optimization Modulo Theories (OMT)

Definition 2.3. (Boxed OMT Problem) Given an SMT formula φ and a set of objectives $\{g_1, \ldots, g_n\}$, the goal of the *multiple-independent-objective OMT problem* [Sebastiani and Trentin 2015b], a.k.a. *boxed* OMT is to find a set of models $\{M_1, \ldots, M_n\}$ of φ such that each M_i maximizes the objective g_i respectively.

Symbolic abstraction of interval domain can be reduced to boxed OMT problem

Example 2.4. Consider the integer formula $\varphi(x, y) \equiv x \geq 0 \wedge y \geq 0 \wedge x + y \leq 10$ in Example 2.2. By setting the template as $\{x, y, -x, -y\}$ and solving the boxed OMT problem "max $\{x, y, -x, -y\}$ $s.t.\ \varphi$", we can obtain the maximal/minimal values of x and y. Clearly, the symbolic abstraction of φ in the interval domain is $x \in [0, 10] \wedge y \in [0, 10]$.

Optimizing an object g

Algorithm 1: SMT-based binary search for optimizing a single objective.

Input: A QF_BV formula φ and an objective g
Output: The maximum value of g $s.t.$ φ

1 **Function** `optimize_one_obj`(φ, g)
2     $ret, low, high \leftarrow \ldots$;
3   **while** $low \leq high$ **do**
4         $mid \leftarrow (low + high)/2$;
5         $\psi \leftarrow \varphi \wedge (mid \leq g \leq high)$;
6   **if** ψ is unsatisfiable **then**
7             $high \leftarrow mid - 1$;
8   **else**
9             $M \leftarrow$ a model of ψ;   `/* use` M `to update` ret `and` low `*/`
10             $ret \leftarrow M(g), low \leftarrow ret + 1$;
11   **return** ret;

Target: maximize an object g
Line 2: $mid = (low + high)/2$
Line 5: use SMT solver to solve $\varphi \wedge (mid \leq g \leq high)$
Line 6-7 (unsat) : $high = mid - 1$
Line 8-10 (sat) : $low = M(g) + 1$

Example 3.1. Consider a bit-vector formula $\varphi(x)$ where x encodes a 3-bits unsigned integer. On the first round of a binary search, we have $low = 0$, $high = 7$, and $mid = 4$. Thus, Algorithm 1 needs to solve the formula $\varphi \wedge 4 \leq x \leq 7$.

Naïve way to maximize multiple objects

Algorithm 2: Naive SMT-based binary search for optimizing multiple objectives.

Input: A QF_BV formula φ and a set of objectives $G = \{g_1, \ldots, g_n\}$

Output: The maximum values of $g_1, \ldots, g_n$ $s.t.$ φ

```
1 Function optimize_multi_obj(φ, G)
2     ret_1, ..., ret_n ← ...;
3     foreach g_i ∈ G do
4         ret_i ← optimize_one_obj(φ, g_i);          /* invoke Algorithm 1 */
5     return ret_1, ..., ret_n;
```

Naïve way to maximize multiple objects:

Line 4: call Ag1 for each g_i

Solving the conjunctive predicate

Algorithm 3: Solving the conjunctive predicate abstraction problem.

Input: A formula φ and a set of predicates $S = \{\phi_1, \ldots, \phi_n\}$

Output: Decide the satisfiability of each $\varphi \wedge \phi_i$ $(1 \leq i \leq n)$

1 **Function** `decide_cpa`(φ, S)
2   **while** $S \neq \emptyset$ **do**
3         $\Psi \leftarrow \bigvee_{\phi_i \in S} \phi_i$;   `/* merge the predicates */`
4   **if** $\varphi \wedge \Psi$ is unsatisfiable **then**
5   **foreach** $\phi_i \in S$ **do**
6   mark $\varphi \wedge \phi_i$ as unsatisfiable;
7   **return**;
8   **else**
9             $M \leftarrow$ a model of $\varphi \wedge \Psi$;   `/* use` M `to filter` ϕ_i `*/`
10   **foreach** $\phi_i \in S$ **do**
11   **if** $M \models \phi_i$ **then**
12   mark $\varphi \wedge \phi_i$ as satisfiable;
13   remove ϕ_i from S;

Line 3: $\Psi \leftarrow \bigvee_{\phi_i \in S} \phi_i$; => $\varphi \wedge \Psi$ due to $(\varphi \wedge \phi_1) \vee (\varphi \wedge \phi_1) \vee \cdots = \varphi \wedge \bigvee_{\phi_i \in S} \phi_i$

Line 4: unsat => no solution for each $\varphi \wedge \phi_i$

Line 9: sat => check if M entails ϕ_i

then we find a solution for object g_i

Solving the conjunctive predicate

Algorithm 3: Solving the conjunctive predicate abstraction problem.

Input: A formula φ and a set of predicates $S = \{\phi_1, \ldots, \phi_n\}$
Output: Decide the satisfiability of each $\varphi \wedge \phi_i$ ($1 \leq i \leq n$)

Function `decide_cpa`(φ, S)
  while $S \neq \emptyset$ **do**
    $\Psi \leftarrow \bigvee_{\phi_i \in S} \phi_i$; `/* merge the predicates */`
    if $\varphi \wedge \Psi$ is unsatisfiable **then**
      foreach $\phi_i \in S$ **do**
        mark $\varphi \wedge \phi_i$ as unsatisfiable;
        return;
    else
      $M \leftarrow$ a model of $\varphi \wedge \Psi$; `/* use` M `to filter` ϕ_i `*/`
      foreach $\phi_i \in S$ **do**
        if $M \models \phi_i$ **then**
          mark $\varphi \wedge \phi_i$ as satisfiable;
          remove ϕ_i from S;

Example 3.2. Consider a bit-vector formula $\varphi \equiv x \leq 2 \wedge \cdots \wedge y \leq 3$ where x and y encode two 3-bits unsigned integers. At the first round of the binary search, we need to decide the satisfiability of $\varphi \wedge 4 \leq x \leq 7$ and $\varphi \wedge 4 \leq y \leq 7$, respectively. Using Algorithm 3, we construct a formula $\varphi \wedge (4 \leq x \leq 7 \vee 4 \leq y \leq 7)$, which is unsatisfiable. Thus, we have that both $\varphi \wedge 4 \leq x \leq 7$ and $\varphi \wedge 4 \leq y \leq 7$ are unsatisfiable.

Combine all algorithms

Line 8: $\phi_i = mid_i \leq g_i \leq high_i$

Line 17-19: unsat => update $high_i$

Line 22-27: sat => update low_i

$$\phi_i = mid_i \leq g_i \leq high_i$$

Additional advantage:

Suppose there is a relation between g_i and g_{i+1}

When g_i is improved, g_{i+1} is also improved, so the algorithm converges faster.

Algorithm 4: Optimized boxed multi-objective optimization.

Input: A QF_BV formula φ and a set of objectives $G = \{g_1, \ldots, g_n\}$

Output: The maximal values of $g_1, \ldots, g_n$ $s.t.$ φ

1 **Function** optimize_multi_obj(φ, G)

2 initialize low_i, $high_i$, ret_i with an interval analysis [Gange et al. 2015];

3 **while** *true* **do**

4 $S \leftarrow \emptyset$;

5 **foreach** $g_i \in G$ **do**

6 **if** $low_i \leq high_i$ **then**

7 $mid_i \leftarrow (low_i + high_i)/2$;

8 $S \leftarrow S \cup \{mid_i \leq g_i \leq high_i\}$;

9 **if** $S == \emptyset$ **then**

10 **break**; /* all variables optimized */

11 **else**

12 decide_cpa_ext(φ, S); /* an extension of Algorithm 3 */

13 **return** $ret_1, \ldots, ret_n$;

14 **Function** decide_cpa_ext(φ, S):

15 **while** *true* **do**

16 $\Psi \leftarrow \bigvee_{\phi_i \in S} \phi_i$; /* merge the predicates */

17 **if** $\varphi \wedge \Psi$ is unsatisfiable **then**

18 **foreach** $\phi_i \in S$ **do**

19 $high_i \leftarrow mid_i - 1$;

20 **return**;

21 **else**

22 $M \leftarrow$ a model of $\varphi \wedge \Psi$; /* use M to update low_i and mid_i */

23 **foreach** $\phi_i \in S$ **do**

24 **if** $M \models \phi_i$ **then**

25 $ret_i \leftarrow M(g_i)$, $low_i \leftarrow ret_i + 1$;

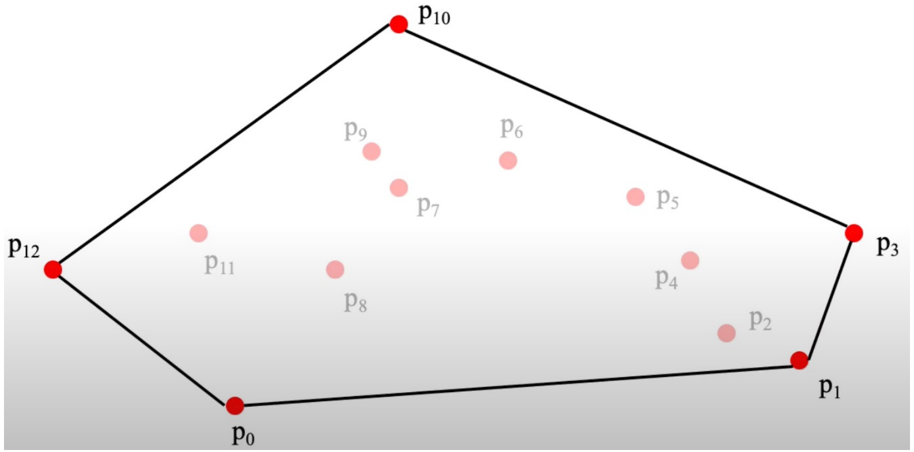
26 $mid_i \leftarrow (low_i + high_i)/2$;

27 $\phi_i \leftarrow mid_i \leq g_i \leq high_i$;

Polyhedral abstraction with symbolic intervals

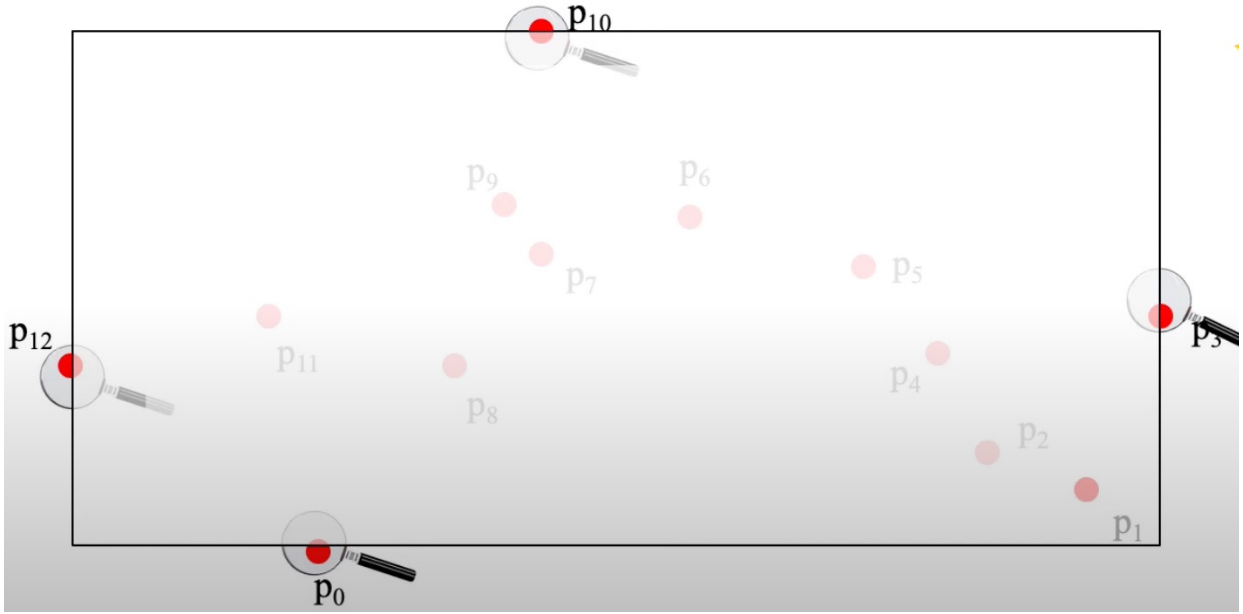
1. Find an interval, determine the extremal points and create a polyhedral abstraction.
2. Find an interval from uncovered points and add the new extremal points to the polyhedral abstraction.

Example.



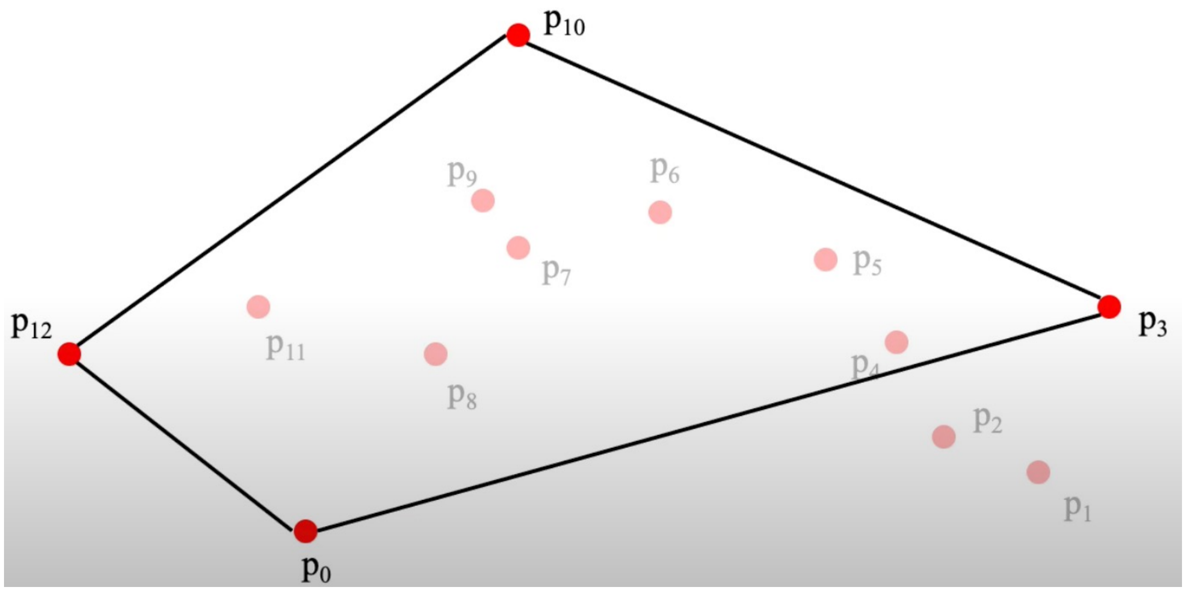
Polyhedral abstraction with symbolic intervals

Construct an interval abstraction and get extremal points $\{p_0, p_3, p_{10}, p_{12}\}$



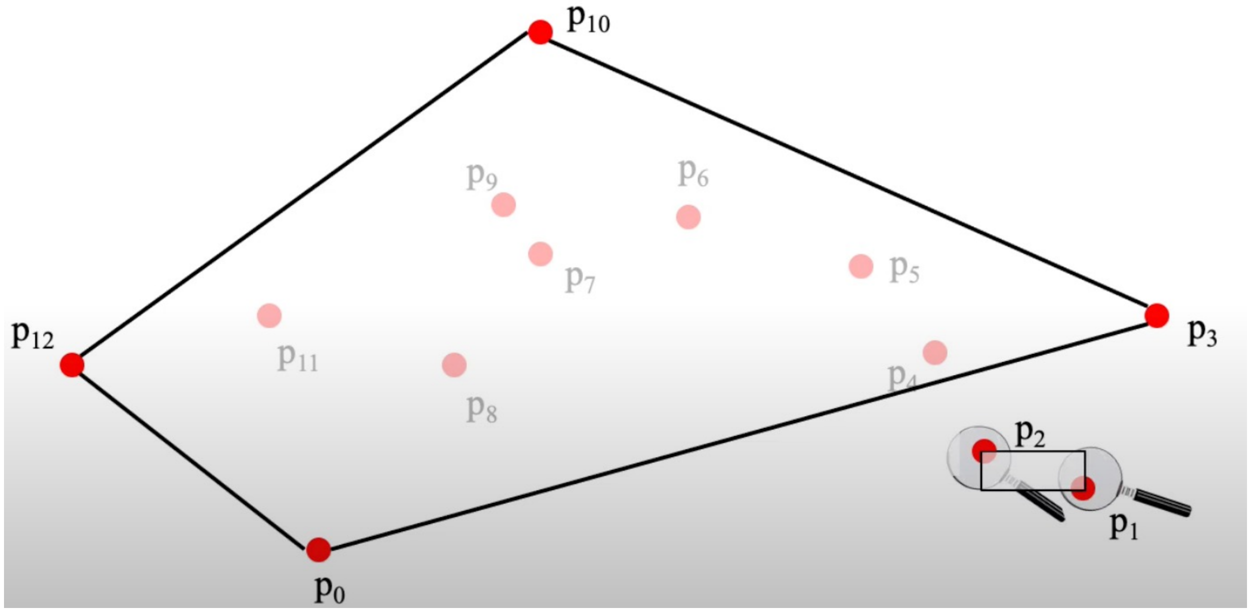
Polyhedral abstraction with symbolic intervals

Construct a polyhedral abstraction based on extremal points



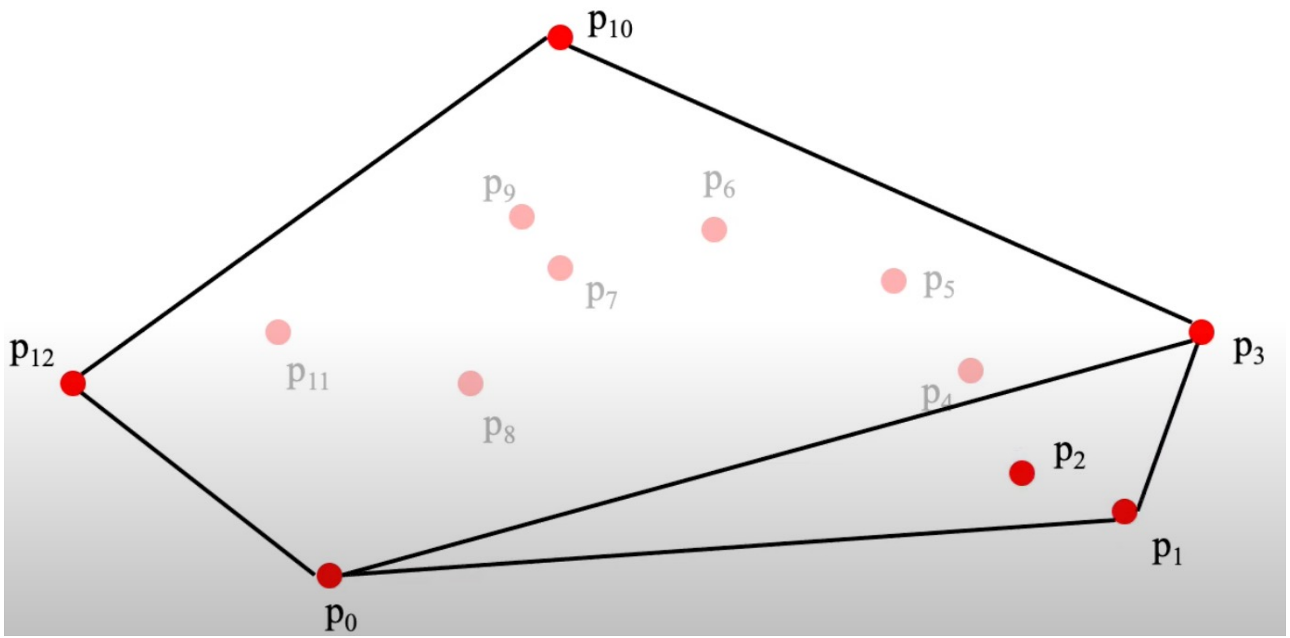
Polyhedral abstraction with symbolic intervals

Find uncovered points and generate a new interval abstraction



Polyhedral abstraction with symbolic intervals

Merge the interval abstraction to the polyhedral abstraction.



Thanks