Program Analysis using Quantifier Elimination Heuristics

Deepak Kapur

Department of Computer Science
University of New Mexico
Albuquerque, NM, USA

with Zhihai Zhang (Peking University) and
Hengjun Zhao (Chinese Academy of Sciences) and
Matthias Forbach, Qi Lu and Thanh Vu Nguyen (UNM)
(work in progress)

Outline

- Quantifier Elimination Approach for Generating (Loop) Invariants.

Outline

- Quantifier Elimination Approach for Generating (Loop) Invariants.
- Octagonal Constraints.

Outline

- Quantifier Elimination Approach for Generating (Loop) Invariants.
- Octagonal Constraints.
- Geometric and Local Quantifier Elimination Heuristic

Outline

- Quantifier Elimination Approach for Generating (Loop) Invariants.
- Octagonal Constraints.
- Geometric and Local Quantifier Elimination Heuristic
- A fast $O(n^2)$ to generate the strongest octagonal invariant

Outline

- Quantifier Elimination Approach for Generating (Loop) Invariants.
- Octagonal Constraints.
- Geometric and Local Quantifier Elimination Heuristic
- A fast $O(n^2)$ to generate the strongest octagonal invariant
- Disjunctive Invariants – **max plus** constraints

Invariants: Integer Square Root

Example

```
x := 1, y := 1, z := 0;
while (x <= N) {
  x := x + y + 2;
  y := y + 2;
  z := z + 1
}
return z
```

Generating Loop Invariant

- Guess/fix the shape of invariants of interest at various program locations with some parameters which need to be determined.

Generating Loop Invariant

- Guess/fix the shape of invariants of interest at various program locations with some parameters which need to be determined.
- For instance, let

$\mathbf{I}: \quad A\ x^2 + B\ y^2 + C\ z^2 + D\ xy + E\ xz + F\ yz + G\ x + H\ y + J\ z + K = 0.$

Generating Loop Invariant

- Guess/fix the shape of invariants of interest at various program locations with some parameters which need to be determined.
- For instance, let

  $\mathbf{I}: \quad A\ x^2 + B\ y^2 + C\ z^2 + D\ xy + E\ xz + F\ yz + G\ x + H\ y + J\ z + K = 0.$
- Generate verification conditions using the hypothesized invariants from the code.

Generating Loop Invariant

- Guess/fix the shape of invariants of interest at various program locations with some parameters which need to be determined.
- For instance, let

  $\mathbf{I}: \quad A\ x^2 + B\ y^2 + C\ z^2 + D\ xy + E\ xz + F\ yz + G\ x + H\ y + J\ z + K = 0.$
- Generate verification conditions using the hypothesized invariants from the code.
  - **VC1:** At first possible entry of the loop:

    $$A + B + D + G + H + K = 0.$$

Generating Loop Invariant

- Guess/fix the shape of invariants of interest at various program locations with some parameters which need to be determined.
- For instance, let

  $\mathbf{I}: \quad A\ x^2 + B\ y^2 + C\ z^2 + D\ xy + E\ xz + F\ yz + G\ x + H\ y + J\ z + K = 0.$
- Generate verification conditions using the hypothesized invariants from the code.
  - **VC1:** At first possible entry of the loop:

    $A + B + D + G + H + K = 0.$
  - **VC2:** For every iteration of the loop body:

    $(I(x, y, z) \wedge x \leq N) \Longrightarrow I(x + y + 2, y + 2, z + 1).$

Generating Loop Invariant

- Guess/fix the shape of invariants of interest at various program locations with some parameters which need to be determined.
- For instance, let

  $\mathbf{I}: \quad A\ x^2 + B\ y^2 + C\ z^2 + D\ xy + E\ xz + F\ yz + G\ x + H\ y + J\ z + K = 0.$
- Generate verification conditions using the hypothesized invariants from the code.
  - **VC1:** At first possible entry of the loop:

    $A + B + D + G + H + K = 0.$
  - **VC2:** For every iteration of the loop body:

    $(I(x, y, z) \wedge x \leq N) \Longrightarrow I(x + y + 2, y + 2, z + 1).$
- Using quantifier elimination, find constraints on parameters $A, B, C, D, E, F, G, H, J, K$ which ensure that the verification conditions are valid for all possible program variables.

- Constraints on parameters are:

$$C = -F, \;\; J = -2B - F + 2K, \;\; G = -4B - F, \;\; H = 3B + F - K.$$

- Constraints on parameters are:

$$C = -F, \;\; J = -2B - F + 2K, \;\; G = -4B - F, \;\; H = 3B + F - K.$$

- Every value of parameters satisfying the above constraints leads to an invariant.

- Constraints on parameters are:

$$C = -F,\ \ J = -2B - F + 2K,\ \ G = -4B - F,\ \ H = 3B + F - K.$$

- Every value of parameters satisfying the above constraints leads to an invariant.
- 7 parameters and 4 equations, so 3 independent parameters, say B, F, K. Make each to be 1 separately with other independent parameters being 0. Get values of dependent parameters.

- Constraints on parameters are:

$$C = -F,\ \ J = -2B - F + 2K,\ \ G = -4B - F,\ \ H = 3B + F - K.$$

- Every value of parameters satisfying the above constraints leads to an invariant.
- 7 parameters and 4 equations, so 3 independent parameters, say B, F, K. Make each to be 1 separately with other independent parameters being 0. Get values of dependent parameters.
- Most general invariants describing all invariants are:

$$y = 2z + 1; \quad z^2 - yz + z + x - y = 0 \quad y^2 - 2z - 4x + 3y = 0,$$

- Constraints on parameters are:

$$C = -F, \;\; J = -2B - F + 2K, \;\; G = -4B - F, \;\; H = 3B + F - K.$$

- Every value of parameters satisfying the above constraints leads to an invariant.
- 7 parameters and 4 equations, so 3 independent parameters, say B, F, K. Make each to be 1 separately with other independent parameters being 0. Get values of dependent parameters.
- Most general invariants describing all invariants are:

$$y = 2z + 1; \quad z^2 - yz + z + x - y = 0 \quad y^2 - 2z - 4x + 3y = 0,$$

from which $x = (z + 1)^2$ follows.

Method for Automatically Generating Invariants by Quantifier Elimination

- Hypothesize assertions, which are parametrized formulas, at various points in a program.

Method for Automatically Generating Invariants by Quantifier Elimination

- Hypothesize assertions, which are parametrized formulas, at various points in a program.
  - Typically entry of every loop and entry and exit of every procedure suffice.

Method for Automatically Generating Invariants by Quantifier Elimination

- Hypothesize assertions, which are parametrized formulas, at various points in a program.
  - Typically entry of every loop and entry and exit of every procedure suffice.
- Generate verification conditions for every path in the program (a path from an assertion to another assertion including itself).

Method for Automatically Generating Invariants by Quantifier Elimination

- Hypothesize assertions, which are parametrized formulas, at various points in a program.
  - Typically entry of every loop and entry and exit of every procedure suffice.
- Generate verification conditions for every path in the program (a path from an assertion to another assertion including itself).
  - Depending upon the logical language chosen to write invariants,
    approximations of assignments and test conditions may be necessary.

Method for Automatically Generating Invariants by Quantifier Elimination

- Hypothesize assertions, which are parametrized formulas, at various points in a program.
  - Typically entry of every loop and entry and exit of every procedure suffice.
- Generate verification conditions for every path in the program (a path from an assertion to another assertion including itself).
  - Depending upon the logical language chosen to write invariants,
    approximations of assignments and test conditions may be necessary.
- Find a formula expressed in terms of parameters eliminating all program variables (using quantifier elimination).

Quality of Invariants

Soundness and Completeness

- Every assignment of parameter values which make the formula true, gives an inductive invariant.

Quality of Invariants

Soundness and Completeness

- Every assignment of parameter values which make the formula true, gives an inductive invariant.
  - If no parameter values can be found, then invariants of hypothesized forms may not exist. Invariants can be guaranteed not to exist if no approximations are made, while generating verification conditions.

Quality of Invariants

Soundness and Completeness

- Every assignment of parameter values which make the formula true, gives an inductive invariant.
  - If no parameter values can be found, then invariants of hypothesized forms may not exist. Invariants can be guaranteed not to exist if no approximations are made, while generating verification conditions.
- If all assignments making the formula true can be finitely described, invariants generated may be the strongest of the hypothesized form. Invariants generated are guaranteed to be the strongest if no approximations are made, while generating verification conditions.

Domains Admitting Quantifier-Elimination

- Generalized Presburger Arithmetic (for invariants expressed using linear inequalities)

Domains Admitting Quantifier-Elimination

- Generalized Presburger Arithmetic (for invariants expressed using linear inequalities)

- Polynomials over an algebraic closed field of charateristic 0: Parametric Gröbner Basis Algorithm (Kapur, 1994), Comprehensive Gröbner Basis System (Weispfenning, 1992).

Domains Admitting Quantifier-Elimination

- Generalized Presburger Arithmetic (for invariants expressed using linear inequalities)

- Polynomials over an algebraic closed field of charateristic 0: Parametric Gröbner Basis Algorithm (Kapur, 1994), Comprehensive Gröbner Basis System (Weispfenning, 1992).

- Quantifier Elimination Techniques for Real Closed Fields (REDLOG, QEPCAD)

Domains Admitting Quantifier-Elimination

- Generalized Presburger Arithmetic (for invariants expressed using linear inequalities)
- Polynomials over an algebraic closed field of charateristic 0: Parametric Gröbner Basis Algorithm (Kapur, 1994), Comprehensive Gröbner Basis System (Weispfenning, 1992).
- Quantifier Elimination Techniques for Real Closed Fields (REDLOG, QEPCAD)
- Combination of Theories–Presburger Arithmetic with Theory of Equality over Uninterpreted Symbols (Shostak, 1979; Nelson, 1981), and with Boolean Algebra (Kuncak, 2007), etc.

Domains Admitting Quantifier-Elimination

- Generalized Presburger Arithmetic (for invariants expressed using linear inequalities)
- Polynomials over an algebraic closed field of charateristic 0: Parametric Gröbner Basis Algorithm (Kapur, 1994), Comprehensive Gröbner Basis System (Weispfenning, 1992).
- Quantifier Elimination Techniques for Real Closed Fields (REDLOG, QEPCAD)
- Combination of Theories–Presburger Arithmetic with Theory of Equality over Uninterpreted Symbols (Shostak, 1979; Nelson, 1981), and with Boolean Algebra (Kuncak, 2007), etc.
- Reduction Approach to Decision Procedures for Theories over Abstract Data Structures, including Finite Lists, Finite Sets, Finite Arrays, Finite Multisets (Kapur and Zarba, 2005).

Constraint Solving

- Quantifier Elimination based approach proposed in a Technical Report of Univ. of New Mexico in 2003.

Constraint Solving

- Quantifier Elimination based approach proposed in a Technical Report of Univ. of New Mexico in 2003.
- Sankaranarayan, Sipma and Manna proposed it using Farkas' Lemma in CAV 2003 and using Gröebner basis algorithms in POPL 2004.

Constraint Solving

- Quantifier Elimination based approach proposed in a Technical Report of Univ. of New Mexico in 2003.
- Sankaranarayan, Sipma and Manna proposed it using Farkas' Lemma in CAV 2003 and using Gröebner basis algorithms in POPL 2004.
- Extensively investigated in many areas including program analysis, program synthesis, termination of programs, as well as hybrid system analysis, particularly safety check and controller synthesis.

Constraint Solving

- Quantifier Elimination based approach proposed in a Technical Report of Univ. of New Mexico in 2003.
- Sankaranarayan, Sipma and Manna proposed it using Farkas' Lemma in CAV 2003 and using Gröebner basis algorithms in POPL 2004.
- Extensively investigated in many areas including program analysis, program synthesis, termination of programs, as well as hybrid system analysis, particularly safety check and controller synthesis.
- Closely related to choosing an abstract domain in the abstract interpretation approach.

How to Scale this Approach

- Quantifier Elimination Methods typically do not scale up due to high complexity.

How to Scale this Approach

- Quantifier Elimination Methods typically do not scale up due to high complexity.
  - Even for Presburger arithmetic, complexity is doubly exponential in the number of quantifier alternations and triply exponential in the number of quantified variables

How to Scale this Approach

- Quantifier Elimination Methods typically do not scale up due to high complexity.
  - Even for Presburger arithmetic, complexity is doubly exponential in the number of quantifier alternations and triply exponential in the number of quantified variables
  - Output is huge and difficult to decipher.

How to Scale this Approach

- Quantifier Elimination Methods typically do not scale up due to high complexity.

  - Even for Presburger arithmetic, complexity is doubly exponential in the number of quantifier alternations and triply exponential in the number of quantified variables
  - Output is huge and difficult to decipher.
  - In practice, they often do not work (i.e., run out of memory or hang).

How to Scale this Approach

- Quantifier Elimination Methods typically do not scale up due to high complexity.
  - Even for Presburger arithmetic, complexity is doubly exponential in the number of quantifier alternations and triply exponential in the number of quantified variables
  - Output is huge and difficult to decipher.
  - In practice, they often do not work (i.e., run out of memory or hang).
- Linear constraint solving on rationals and reals (polyhedral domain), while of polynomial complexity, has been found in practice to be inefficient and slow, especially when used repeatedly as in abstract interpretation approach [Miné]

Octagonal Constraints

- **Octagonal Constraints** : $l \leq \pm x \pm y \leq h$, a highly restricted subset of linear constraints (at most two variables with coefficients from $\{-1, 0, 1\}$).

Octagonal Constraints

- **Octagonal Constraints** : $l \leq \pm x \pm y \leq h$, a highly restricted subset of linear constraints (at most two variables with coefficients from $\{-1, 0, 1\}$).
- This fragment is the most expressive fragment of linear arithmetic over the integers with a polynomial time decision procedure.

Octagonal Constraints

- **Octagonal Constraints** : $l \leq \pm x \pm y \leq h$, a highly restricted subset of linear constraints (at most two variables with coefficients from $\{-1, 0, 1\}$).
- This fragment is the most expressive fragment of linear arithmetic over the integers with a polynomial time decision procedure.
- Extending constraints to contain three variables (with just unit coefficients) per inequality makes satisfiability check over the integers NP-complete.

Octagonal Constraints

- **Octagonal Constraints** : $l \leq \pm x \pm y \leq h$, a highly restricted subset of linear constraints (at most two variables with coefficients from $\{-1, 0, 1\}$).
- This fragment is the most expressive fragment of linear arithmetic over the integers with a polynomial time decision procedure.
- Extending constraints to contain three variables (with just unit coefficients) per inequality makes satisfiability check over the integers NP-complete.
- Two variable inequalities with non-unit coefficients over the integers makes the satisfiability check NP-complete.

Octagonal Constraints

- **Octagonal Constraints** : $l \leq \pm x \pm y \leq h$, a highly restricted subset of linear constraints (at most two variables with coefficients from $\{-1, 0, 1\}$).
- This fragment is the most expressive fragment of linear arithmetic over the integers with a polynomial time decision procedure.
- Extending constraints to contain three variables (with just unit coefficients) per inequality makes satisfiability check over the integers NP-complete.
- Two variable inequalities with non-unit coefficients over the integers makes the satisfiability check NP-complete.
- Class of programs that can be analyzed are very restricted. Still using octagonal constraints (and other heuristics), ASTREE is able to successfully analyze hundreds of thousands of lines of code of numerical software.

Octagonal Constraints

- **Octagonal Constraints** : $l \leq \pm x \pm y \leq h$, a highly restricted subset of linear constraints (at most two variables with coefficients from $\{-1, 0, 1\}$).
- This fragment is the most expressive fragment of linear arithmetic over the integers with a polynomial time decision procedure.
- Extending constraints to contain three variables (with just unit coefficients) per inequality makes satisfiability check over the integers NP-complete.
- Two variable inequalities with non-unit coefficients over the integers makes the satisfiability check NP-complete.
- Class of programs that can be analyzed are very restricted. Still using octagonal constraints (and other heuristics), ASTREE is able to successfully analyze hundreds of thousands of lines of code of numerical software.
- Miné gave well-designed algorithms based on Difference Bound Matrices (DBMs) and graph representation for performing various operations needed for program analysis using the abstract interpretation approach.

Octagonal Constraints

- **Octagonal Constraints** : $l \leq \pm x \pm y \leq h$, a highly restricted subset of linear constraints (at most two variables with coefficients from $\{-1, 0, 1\}$).
- This fragment is the most expressive fragment of linear arithmetic over the integers with a polynomial time decision procedure.
- Extending constraints to contain three variables (with just unit coefficients) per inequality makes satisfiability check over the integers NP-complete.
- Two variable inequalities with non-unit coefficients over the integers makes the satisfiability check NP-complete.
- Class of programs that can be analyzed are very restricted. Still using octagonal constraints (and other heuristics), ASTREE is able to successfully analyze hundreds of thousands of lines of code of numerical software.
- Miné gave well-designed algorithms based on Difference Bound Matrices (DBMs) and graph representation for performing various operations needed for program analysis using the abstract interpretation approach.
- Miné's algorithms are of $O(n^3)$ (sometimes, $O(n^4)$), where n is the number of variables.

Octagonal Constraints and Quantifier Elimination

- Octagonal constraints have a fixed shape. Given n variables, the most general formula (after simplification) is of the following form
  $\bigwedge_{i,j}(\;\; I_{i,j} : \;\; a_{i,j} \leq x_i - x_j \leq b_{i,j}, \;\; c_{i,j} \leq x_i + x_j \leq d_{i,j}, \;\; e_i \leq x_i \leq f_i \;\; g_j \leq x_j \leq h_j)$

  for every pair of variables x_i, x_j, where $a_{i,j}, b_{i,j}, c_{i,j}, d_{i,j}, e_i, f_i, g_j, h_j$ are parameters.

Octagonal Constraints and Quantifier Elimination

- Octagonal constraints have a fixed shape. Given n variables, the most general formula (after simplification) is of the following form
$\bigwedge_{i,j}(\; l_{i,j} : \; a_{i,j} \leq x_i - x_j \leq b_{i,j}, \; c_{i,j} \leq x_i + x_j \leq d_{i,j}, \; e_i \leq x_i \leq f_i \; g_j \leq x_j \leq h_j)$
for every pair of variables x_i, x_j, where $a_{i,j}, b_{i,j}, c_{i,j}, d_{i,j}, e_i, f_i, g_j, h_j$ are parameters.
- For a finite program path consisting of a sequence of assignment statements interspersed with tests, its behavior is approximated so that the post condition is also of the above form.

Octagonal Constraints and Quantifier Elimination

- Octagonal constraints have a fixed shape. Given n variables, the most general formula (after simplification) is of the following form
  $\bigwedge_{i,j}(\; I_{i,j} : \; a_{i,j} \leq x_i - x_j \leq b_{i,j}, \; c_{i,j} \leq x_i + x_j \leq d_{i,j}, \; e_i \leq x_i \leq f_i \; g_j \leq x_j \leq h_j)$

  for every pair of variables x_i, x_j, where $a_{i,j}, b_{i,j}, c_{i,j}, d_{i,j}, e_i, f_i, g_j, h_j$ are parameters.
- For a finite program path consisting of a sequence of assignment statements interspersed with tests, its behavior is approximated so that the post condition is also of the above form.
- A verification condition is expressed using atomic formulas that are all octagonal constraints.

$$\bigwedge_{i,j}((I_{i,j} \wedge \alpha(x_i, x_j)) \Rightarrow I'_{i,j}),$$

along with additional parameter-free constraints $\alpha(x_i, x_j)$, of the same form in which lower and upper bounds are constants.

Approach: Local QE Heuristics

- Analysis of a big conjunctive constraint on every possible pair of variables can be considered individually by considering the subformula on each distinct pair.

Approach: Local QE Heuristics

- Analysis of a big conjunctive constraint on every possible pair of variables can be considered individually by considering the subformula on each distinct pair.
- Consider a precondition, which is a conjunction,
  $a_{i,j} \leq x_i - x_j \leq b_{i,j}, \quad c_{i,j} \leq x_i + x_j \leq d_{i,j}, \quad e_i \leq x_i \leq f_i, \quad g_j \leq x_j \leq h_j$
  Assignment statements are of the form $x_i := x_i + a$ or $x_i := -x_i + a$. And, tests are lower and upper bounds on variables and expressions of the form $\pm x_i \pm x_j$. Otherwise, tests and assignments must be approximated.

Approach: Local QE Heuristics

- Analysis of a big conjunctive constraint on every possible pair of variables can be considered individually by considering the subformula on each distinct pair.
- Consider a precondition, which is a conjunction, $a_{i,j} \leq x_i - x_j \leq b_{i,j}, \quad c_{i,j} \leq x_i + x_j \leq d_{i,j}, \quad e_i \leq x_i \leq f_i, \quad g_j \leq x_j \leq h_j$
  Assignment statements are of the form $x_i := x_i + a$ or $x_i := -x_i + a$. And, tests are lower and upper bounds on variables and expressions of the form $\pm x_i \pm x_j$. Otherwise, tests and assignments must be approximated.
- Quantifier elimination heuristics can be developed using which it is possible to generate constraints on lower and upper bounds by table look ups in $O(n^2)$ steps, where n is the number of program variables.

Geometric QE Heuristic

- Analyze how an octagon is affected by transformations due to assignments.

Geometric QE Heuristic

- Analyze how an octagon is affected by transformations due to assignments.
- Identify conditions under which the transformed octagon includes the portion of the original octagon satisfying tests along a program path.

Geometric QE Heuristic

- Analyze how an octagon is affected by transformations due to assignments.
- Identify conditions under which the transformed octagon includes the portion of the original octagon satisfying tests along a program path.
- For each assignment case, a table is built showing the effect on the parameter values by determining the effect on every type of constraints.

Table 1: Assignments with signs of variables reversed

$$x_i := -x_i + A, \quad x_j := -x_j + B, \quad \Delta_1 = B - A, \ \ \Delta_2 = -A - B, \ \ \Delta_3 = -A, \ \ \Delta_4 = -B.$$

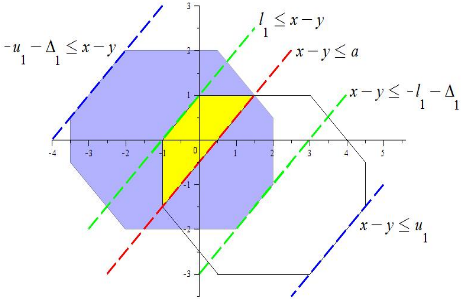
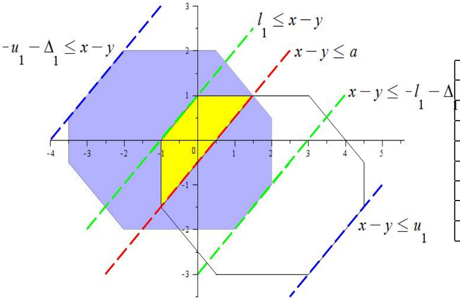


Table 1: Assignments with signs of variables reversed

$$x_i := -x_i + A, \quad x_j := -x_j + B, \quad \Delta_1 = B - A, \quad \Delta_2 = -A - B, \quad \Delta_3 = -A, \quad \Delta_4 = -B.$$



	present	absent
$x_i - x_j \le a$	$a \le -l_1 - \Delta_1$	$u_1 \le -l_1 - \Delta_1$
$x_i - x_j \ge b$	$-u_1 - \Delta_1 \le b$	$-u_1 - \Delta_1 \le l_1$
$x_i + x_j \le c$	$c \le -l_2 - \Delta_2$	$u_2 \le -l_2 - \Delta_2$
$x_i + x_j \ge d$	$-u_2 - \Delta_2 \le d$	$-u_2 - \Delta_2 \le l_2$
$x_i \le e$	$e \le -l_3 - \Delta_3$	$u_3 \le -l_3 - \Delta_3$
$x_i \ge f$	$-u_3 - \Delta_3 \le f$	$-u_3 - \Delta_3 \le l_3$
$x_j \le g$	$g \le -l_4 - \Delta_4$	$u_4 \le -l_4 - \Delta_4$
$x_j \ge h$	$-u_4 - \Delta_4 \le h$	$-u_4 - \Delta_4 \le l_4$

Table 2: No changing signs of variables

$$x_i := x_i + A, \quad x_j := x_j + B, \quad \Delta_1 = A - B, \quad \Delta_2 = A + B, \quad \Delta_3 = A, \quad \Delta_4 = B.$$

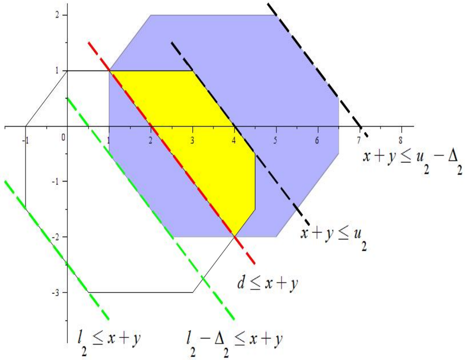
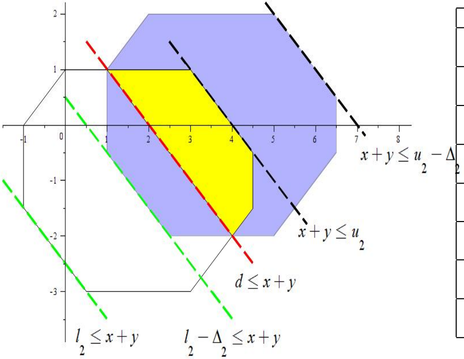


Table 2: No changing signs of variables

$$x_i := x_i + A, \quad x_j := x_j + B, \quad \Delta_1 = A - B, \quad \Delta_2 = A + B, \quad \Delta_3 = A, \quad \Delta_4 = B.$$



	present	absent
$x_i - x_j \leq a$ $\Delta_1 > 0$	$u_1 \geq a + \Delta_1$	$u_1 = +\infty$
$x_i - x_j \geq b$ $\Delta_1 < 0$	$l_1 \leq b + \Delta_1$	$l_1 = -\infty$
$x_i + x_j \leq c$ $\Delta_2 > 0$	$u_2 \geq c + \Delta_2$	$u_2 = +\infty$
$x_i + x_j \geq d$ $\Delta_2 < 0$	$l_2 \leq d + \Delta_2$	$l_2 = -\infty$
$x_i \leq e$ $\Delta_3 > 0$	$u_3 \geq e + \Delta_3$	$u_3 = +\infty$
$x_i \geq f$ $\Delta_3 < 0$	$l_3 \leq f + \Delta_3$	$l_3 = -\infty$
$x_j \leq g$ $\Delta_4 > 0$	$u_4 \geq g + \Delta_4$	$u_4 = +\infty$
$x_j \geq h$ $\Delta_4 < 0$	$l_4 \leq h + \Delta_4$	$l_4 = -\infty$

Table 3: Sign of exactly one variable is changed

$$x_i := -x_i + A, \quad x_j := x_j + B, \quad \Delta_1 = B - A, \quad \Delta_2 = -A - B, \quad \Delta_3 = -A, \quad \Delta_4 = B.$$

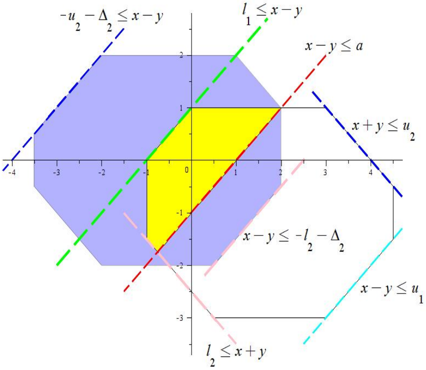
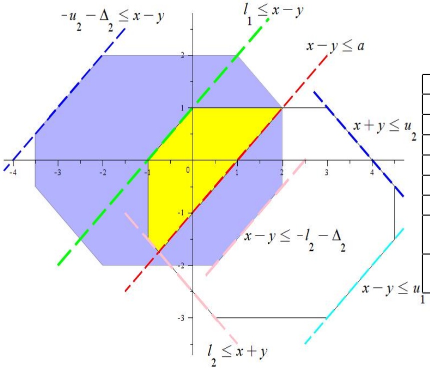


Table 3: Sign of exactly one variable is changed

$$x_i := -x_i + A, \quad x_j := x_j + B, \quad \Delta_1 = B - A, \quad \Delta_2 = -A - B, \quad \Delta_3 = -A, \quad \Delta_4 = B.$$



	present	absent
$x_i - x_j \leq a$	$a \leq -l_2 - \Delta_2$	$u_1 \leq -l_2 - \Delta_2$
$x_i - x_j \geq b$	$-u_2 - \Delta_2 \leq b$	$-u_2 - \Delta_2 \leq l_1$
$x_i + x_j \leq c$	$c \leq -l_1 - \Delta_1$	$u_2 \leq -l_1 - \Delta_1$
$x_i + x_j \geq d$	$-u_1 - \Delta_1 \leq d$	$-u_1 - \Delta_1 \leq l_2$
$x_i \leq e$	$e \leq -l_3 - \Delta_3$	$u_3 \leq -l_3 - \Delta_3$
$x_i \geq f$	$-u_3 - \Delta_3 \leq f$	$-u_3 - \Delta_3 \leq l_3$
$x_j \leq g$ $\Delta_4 > 0$	$u_4 \geq g + \Delta_4$	$u_4 = +\infty$
$x_j \geq h$ $\Delta_4 < 0$	$l_4 \leq h + \Delta_4$	$l_4 = -\infty$

A Simple Example

Example

```
 x := 4; y := 6;
while (x + y >= 0) do
 if (y >= 6) then { x := -x;  y := y - 1 }
    else { x := x - 1; y := -y }
endwhile
```

A Simple Example

Example

```
 x := 4; y := 6;
while (x + y >= 0) do
 if (y >= 6) then { x := -x;  y := y - 1 }
    else { x := x - 1; y := -y }
endwhile
```

VC0: $I(4, 6)$
VC1: $(I(x, y) \wedge (x + y) \geq 0 \wedge y \geq 6) \implies I(-x, y - 1)$.
VC2: $(I(x, y) \wedge (x + y) \geq 0 \wedge y < 6) \implies I(x - 1, -y)$.

Generating Constraints on Parameters

- **VC0:**
  $l_1 \leq -2 \leq u_1 \wedge l_2 \leq 10 \leq u_2 \wedge l_3 \leq 4 \geq u_3 \wedge l_4 \leq 6 \leq u_4.$

Generating Constraints on Parameters

- **VC0:**
  $l_1 \leq -2 \leq u_1 \wedge l_2 \leq 10 \leq u_2 \wedge l_3 \leq 4 \geq u_3 \wedge l_4 \leq 6 \leq u_4$.
- **VC1:** $x - y$: $-u_2 - 1 \leq l_1 \wedge u_1 \leq -l_2 - 1$.
  $x + y$: $-u_1 + 1 \leq 0 \wedge u_2 \leq -l_1 + 1$.
  x: $l_3 + u_3 = 0$.
  y: $l_4 \leq 5$.

Generating Constraints on Parameters

- **VC0:**
  $l_1 \le -2 \le u_1 \wedge l_2 \le 10 \le u_2 \wedge l_3 \le 4 \ge u_3 \wedge l_4 \le 6 \le u_4$.
- **VC1:** $x - y$: $-u_2 - 1 \le l_1 \wedge u_1 \le -l_2 - 1$.
  $x + y$: $-u_1 + 1 \le 0 \wedge u_2 \le -l_1 + 1$.
  x: $l_3 + u_3 = 0$.
  y: $l_4 \le 5$.
- **VC2:** $x - y$: $-u_2 - 1 \le -u_1 \wedge 10 \le -l_2 - 1$.
  $x + y$: $l_1 + 1 \le 0 \wedge u_2 \le u_1 + 1$.
  x: $l_3 \le -6$.
  y: $-u_4 \le l_4 \wedge 5 \le -l_4$.

Generating Constraints on Parameters

- **VC0:**
  $l_1 \le -2 \le u_1 \wedge l_2 \le 10 \le u_2 \wedge l_3 \le 4 \ge u_3 \wedge l_4 \le 6 \le u_4.$
- **VC1:** $x - y$: $-u_2 - 1 \le l_1 \wedge u_1 \le -l_2 - 1.$
  $x + y$: $-u_1 + 1 \le 0 \wedge u_2 \le -l_1 + 1.$
  x: $l_3 + u_3 = 0.$
  y: $l_4 \le 5.$
- **VC2:** $x - y$: $-u_2 - 1 \le -u_1 \wedge 10 \le -l_2 - 1.$
  $x + y$: $l_1 + 1 \le 0 \wedge u_2 \le u_1 + 1.$
  x: $l_3 \le -6.$
  y: $-u_4 \le l_4 \wedge 5 \le -l_4.$
- Making the l_i's as large as possible and u_i's as small as possible:

$$l_1 = -10, u_1 = 9, l_2 = -11, u_2 = 10, l_3 = -6, u_3 = 6, l_4 = -5, u_4 = 6$$

Generating Constraints on Parameters

- **VC0:**
  $l_1 \leq -2 \leq u_1 \wedge l_2 \leq 10 \leq u_2 \wedge l_3 \leq 4 \geq u_3 \wedge l_4 \leq 6 \leq u_4.$
- **VC1:** $x - y$: $-u_2 - 1 \leq l_1 \wedge u_1 \leq -l_2 - 1.$
  $x + y$: $-u_1 + 1 \leq 0 \wedge u_2 \leq -l_1 + 1.$
  x: $l_3 + u_3 = 0.$
  y: $l_4 \leq 5.$
- **VC2:** $x - y$: $-u_2 - 1 \leq -u_1 \wedge 10 \leq -l_2 - 1.$
  $x + y$: $l_1 + 1 \leq 0 \wedge u_2 \leq u_1 + 1.$
  x: $l_3 \leq -6.$
  y: $-u_4 \leq l_4 \wedge 5 \leq -l_4.$
- Making the l_i's as large as possible and u_i's as small as possible:

  $$l_1 = -10, u_1 = 9, l_2 = -11, u_2 = 10, l_3 = -6, u_3 = 6, l_4 = -5, u_4 = 6$$

- The corresponding invariant is:

  $$-10 \leq x-y \leq 9 \wedge -11 \leq x+y \leq 10 \wedge -6 \leq x \leq 6 \wedge -5 \leq y \leq 6.$$

Generating Invariants using Table Look-ups

- To determine parameter constraints corresponding to a specific program path, read the corresponding entries from the table.

Generating Invariants using Table Look-ups

- To determine parameter constraints corresponding to a specific program path, read the corresponding entries from the table.
- Accumulate all such constraints on parameter values. They are also of octagonal form.

Generating Invariants using Table Look-ups

- To determine parameter constraints corresponding to a specific program path, read the corresponding entries from the table.
- Accumulate all such constraints on parameter values. They are also of octagonal form.
- Every parameter value that satisfies the parameter constraints leads to an invariant.

Generating Invariants using Table Look-ups

- To determine parameter constraints corresponding to a specific program path, read the corresponding entries from the table.
- Accumulate all such constraints on parameter values. They are also of octagonal form.
- Every parameter value that satisfies the parameter constraints leads to an invariant.
- Maximum values of lower bounds and minimal values of upper bounds satisfying the parameter constraints gives the strongest invariants.

Generating Invariants using Table Look-ups

- To determine parameter constraints corresponding to a specific program path, read the corresponding entries from the table.
- Accumulate all such constraints on parameter values. They are also of octagonal form.
- Every parameter value that satisfies the parameter constraints leads to an invariant.
- Maximum values of lower bounds and minimal values of upper bounds satisfying the parameter constraints gives the strongest invariants.
- The above can be computed using Floyd-Warshall's algorithm.

Generating Invariants using Table Look-ups

- To determine parameter constraints corresponding to a specific program path, read the corresponding entries from the table.
- Accumulate all such constraints on parameter values. They are also of octagonal form.
- Every parameter value that satisfies the parameter constraints leads to an invariant.
- Maximum values of lower bounds and minimal values of upper bounds satisfying the parameter constraints gives the strongest invariants.
- The above can be computed using Floyd-Warshall's algorithm.
- Overall Complexity: $O(n^2)$.

Local QE Heuristics: Propagation of Tests?

- Local propagation (i.e., propagate bounds only for the pair of variables appearing in the constraints) on tests to bring them in a canonical form can sometimes improve the bounds. But that is not always clear. The complexity still remains $O(n^2)$.

Local QE Heuristics: Propagation of Tests?

- Local propagation (i.e., propagate bounds only for the pair of variables appearing in the constraints) on tests to bring them in a canonical form can sometimes improve the bounds. But that is not always clear. The complexity still remains $O(n^2)$.
- Global propagation (meaning propagate bounds from the given constraints on a pair of variables to other constraints in which these variables appear) can sometimes improve the bounds even further. But, then the complexity is $O(n^3)$.

Local QE Heuristics: Propagation of Tests?

- Local propagation (i.e., propagate bounds only for the pair of variables appearing in the constraints) on tests to bring them in a canonical form can sometimes improve the bounds. But that is not always clear. The complexity still remains $O(n^2)$.
- Global propagation (meaning propagate bounds from the given constraints on a pair of variables to other constraints in which these variables appear) can sometimes improve the bounds even further. But, then the complexity is $O(n^3)$.
- There are examples for which both local and global propagation (closure) lead to worse results.

Comparative Analysis

- QE based methods generate inductive invariants.

Comparative Analysis

- QE based methods generate inductive invariants.
- Invariants generated by a complete QE based method are the strongest invariants of the given shape.

Comparative Analysis

- QE based methods generate inductive invariants.
- Invariants generated by a complete QE based method are the strongest invariants of the given shape.
- Any method for generating invariants that uses fixed point computation, also generates inductive invariants, and hence, cannot generate stronger invariants than those generated by a complete QE method insofar as these invariants are of the same form.

Comparative Analysis

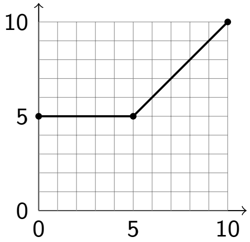
- QE based methods generate inductive invariants.
- Invariants generated by a complete QE based method are the strongest invariants of the given shape.
- Any method for generating invariants that uses fixed point computation, also generates inductive invariants, and hence, cannot generate stronger invariants than those generated by a complete QE method insofar as these invariants are of the same form.
- It has been found in practice that QE based methods generate stronger inductive invariants than methods based on abstract interpretation for polyhedral domain.

Comparative Analysis

- QE based methods generate inductive invariants.
- Invariants generated by a complete QE based method are the strongest invariants of the given shape.
- Any method for generating invariants that uses fixed point computation, also generates inductive invariants, and hence, cannot generate stronger invariants than those generated by a complete QE method insofar as these invariants are of the same form.
- It has been found in practice that QE based methods generate stronger inductive invariants than methods based on abstract interpretation for polyhedral domain.
- An open question: the strength of invariants generated by the proposed incomplete heuristics.

Disjunctive Invariants

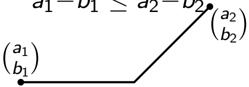
$x := 0; y := 5;$
while $(x < 10)$ do
  if $(x < 5)$ then
    $x := x + 1;$
  else
    x := x+1; y := y+1;



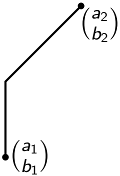
$I = (0 \leq x \leq 5 \wedge y = 5) \vee (5 \leq x \leq 10 \wedge x = y)$.
It consists of two lines and is clearly not convex.
It cannot be expressed as a conjunction of linear constraints, including octagonal constraints or even general polyhedra.

Shapes

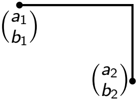
Shape 1:
$b_1 \leq b_2$
$a_1 - b_1 \leq a_2 - b_2$



Shape 2:
$b_1 \leq b_2$
$a_1 - b_1 \geq a_2 - b_2$



Shape 3:
$b_1 \geq b_2$



Shape 1

assignment statements	max-plus polyhedra to be cut off
$A = 0, B = 0$	–
$A > 0, B = 0, \quad A \leq \Delta_a - \Delta_b$	$\mathrm{co}(\{\binom{a_2-\Delta_b-A+1}{b_1}, \binom{a_2}{b_2}\})$
$A > 0, B = A, \quad A \leq \Delta_b$	$\mathrm{co}(\{\binom{a_1}{b_1}, \binom{a_2-\Delta_b-1}{b_1}\})$, if $\Delta_a \neq \Delta_b$ $\mathrm{co}(\{\binom{a_2-A+1}{b_2-A+1}, \binom{a_2}{b_2}\})$
$A > B, B > 0, \quad \Delta \leq \Delta_a - \Delta_b, B \leq \Delta_b$	$\mathrm{co}(\{\binom{a_1}{b_1}, \binom{a_2-\Delta_b-\Delta-1}{b_1}\})$, if $\Delta_b + \Delta \neq \Delta_a$ $\mathrm{co}(\{\binom{a_2}{b_2}, \binom{a_2-\Delta_b-\Delta+1}{b_1}\})$
$A < 0, B = 0, \quad \lvert A\rvert \leq \Delta_a - \Delta_b$	$\mathrm{co}(\{\binom{a_1}{b_1}, \binom{a_1-A-1}{b_1}\})$ $\mathrm{co}(\{\binom{a_2-\Delta_b+1}{b_1+1}, \binom{a_2}{b_2}\})$, if $\Delta_b \neq 0$
$A < 0, B = A, \quad \lvert A\rvert \leq \Delta_b$	$\mathrm{co}(\{\binom{a_1}{b_1}, \binom{a_2-\Delta_b-A+1}{b_1-A+1}\})$
$A < B, B < 0, \quad \Delta \leq \Delta_a - \Delta_b, \lvert B\rvert < \Delta_b$	$\mathrm{co}(\{\binom{a_1}{b_1}, \binom{a_2-\Delta_b-B-1}{b_1-B-1}\})$ $\mathrm{co}(\{\binom{a_2}{b_2}, \binom{a_2-\Delta_b-B+1}{b_1-B+1}\})$, if $B + \Delta_b \neq 0$
all other cases	$\mathrm{co}(\{\binom{a_1}{b_1}, \binom{a_2}{b_2}\})$

Shape 2

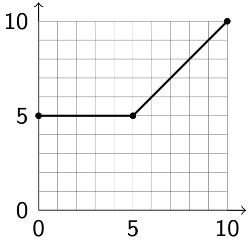
assignment statements		max-plus polyhedra to be cut off
$A = 0, B = 0$		–
$A = 0, B > 0,$	$B \leq \Delta_b - \Delta_a$	$\mathrm{co}(\{\binom{a_1}{b_2 - \Delta_a - B + 1}, \binom{a_2}{b_2}\})$
$A = 0, B < 0,$	$\lvert B \rvert \leq \Delta_b - \Delta_a$	$\mathrm{co}(\{\binom{a_1}{b_1}, \binom{a_1}{b_1 - B + 1}\})$ $\mathrm{co}(\{\binom{a_1 + 1}{b_2 - \Delta_a + 1}, \binom{a_2}{b_2}\})$, if $a_1 \neq a_2$
$A > 0, B = A,$	$A \leq \Delta_a$	$\mathrm{co}(\{\binom{a_1}{b_1}, \binom{a_1}{b_2 - \Delta_a - 1}\})$, if $\Delta_a \neq \Delta_b$ $\mathrm{co}(\{\binom{a_2}{b_2}, \binom{a_2 - A + 1}{b_2 - A + 1}\})$
$A > 0, B > A,$	$A \leq \Delta_a, \Delta \geq \Delta_b - \Delta_a$	$\mathrm{co}(\{\binom{a_1}{b_1}, \binom{a_1}{b_2 - \Delta_a + \Delta - 1}\})$ $\mathrm{co}(\{\binom{a_2}{b_2}, \binom{a_1}{b_2 - \Delta_a + \Delta + 1}\})$, if $\Delta_a - \Delta \neq \Delta_b$
$A < 0, B = A,$	$\lvert A \rvert \leq \Delta_a$	$\mathrm{co}(\{\binom{a_1}{b_1}, \binom{a_1 - A - 1}{b_2 - \Delta_a - A - 1}\})$
$A < 0, B < A,$	$\lvert A \rvert \leq \Delta_a, \Delta \leq \Delta_b - \Delta_a$	$\mathrm{co}(\{\binom{a_1}{b_1}, \binom{a_1 - A - 1}{b_2 - \Delta_a - A - 1}\})$ $\mathrm{co}(\{\binom{a_2}{b_2}, \binom{a_1 - A + 1}{b_2 - \Delta_a - A + 1}\})$, if $\Delta_b + A \neq 0$
all other cases		$\mathrm{co}(\{\binom{a_1}{b_1}, \binom{a_2}{b_2}\})$

Shape 3

assignment statements	max-plus polyhedra to be cut off
$A = 0, B = 0$	–
$A = 0, B > 0, \quad B \leq -\Delta_b$	$\mathrm{co}(\{\binom{a_1}{b_1}, \binom{a_2}{b_1 - B + 1}\})$
$A = 0, B < 0, \quad B \geq \Delta_b$	$\mathrm{co}(\{\binom{a_1}{b_1}, \binom{a_2 - 1}{b_1}\})$, if $a_1 \neq a_2$ $\mathrm{co}(\{\binom{a_2}{b_2}, \binom{a_2}{b_2 - B - 1}\})$
$A > 0, B = 0, \quad A \leq \Delta_a$	$\mathrm{co}(\{\binom{a_2}{b_2}, \binom{a_2 - A + 1}{b_1}\})$
$A > 0, B < 0, \quad A \leq \Delta_a, B \geq \Delta_b$	$\mathrm{co}(\{\binom{a_1}{b_1}, \binom{a_2 - A - 1}{b_1}\})$, if $A \neq \Delta_a$ $\mathrm{co}(\{\binom{a_2}{b_2}, \binom{a_2 - A + 1}{b_1}\})$
$A < 0, B = 0, \quad \lvert A \rvert \leq \Delta_a$	$\mathrm{co}(\{\binom{a_1}{b_1}, \binom{a_1 - A - 1}{b_1}\})$ $\mathrm{co}(\{\binom{a_2}{b_2}, \binom{a_2}{b_1 - 1}\})$, if $b_1 \neq b_2$.
$A < 0, B > 0, \quad \lvert A \rvert \leq \Delta_a, B \leq -\Delta_b$	$\mathrm{co}(\{\binom{a_1}{b_1}, \binom{a_2}{b_1 - B + 1}\})$ $\mathrm{co}(\{\binom{a_2}{b_2}, \binom{a_2}{b_1 - B - 1}\})$, if $\Delta_b + B \neq 0$.
all other cases	$\mathrm{co}(\{\binom{a_1}{b_1}, \binom{a_2}{b_2}\})$

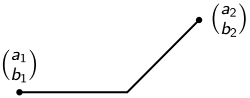
Example

$x := 0; y := 5;$
while $(x < 10)$ do
  if $(x < 5)$ then
    $x := x + 1;$
  else
    x := x+1; y := y+1;



Example

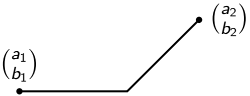
Assumed shape of M:



Loop invariant M must be satisfied when the loop is entered.
The initial value is situated on the horizontal branch of M giving the constraints
$a_1 \leq 0 \leq a_2 - (b_2 - b_1)$ and $b_1 = 0$.

Example

Assumed shape of M:



First branch: VC is

$$\binom{x}{y} \in \mathrm{co}(\{\binom{a_1}{b_1}, \binom{a_2}{b_2}\}) \wedge (x < 10 \wedge x < 5) \implies \binom{x+1}{y} \in \mathrm{co}(\{\binom{a_1}{b_1}, \binom{a_2}{b_2}\})$$

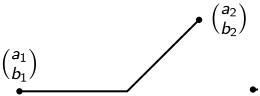
or equivalently

$$\binom{x}{y} \in \mathrm{co}(\{\binom{a_1}{b_1}, \binom{a_2}{b_2}\}) \wedge (x < 10 \wedge x < 5) \implies \binom{x}{y} \in \mathrm{co}(\{\binom{a_1-1}{b_1}, \binom{a_2-1}{b_2}\}) \,.$$

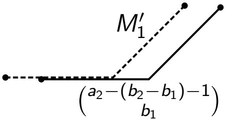
Resulting constraint: $a_2 - (b_2 - b_1) - 1 \geq 4$

Example

Assumed shape of M:



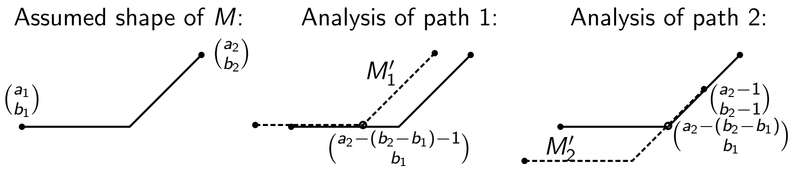
Analysis of path 1:



Second branch: let M_2' be the max-plus polyhedron generated by $\binom{a_1-1}{b_1-1}$ and $\binom{a_2-1}{b_2-1}$.

For M to be an invariant, every point in $M \setminus M_2'$, i.e. every point in M left of $\binom{a_2-(b_2-b_1)}{b_1}$ or right of $\binom{a_2-1}{b_2-1}$, is cut off by the constraints $x < 10 \wedge x \geq 5$ (or equivalently $x \leq 9 \wedge x \leq 4$) along the path. This implies $a_2 - (b_2 - b_1) \leq 5$ and $a_2 - 1 \geq 9$.

Example



All these constraints can be combined to $a_1 \leq 0$, $b_1 = 0$, $a_2 \geq 10$, and $a_2 - b_2 = 5$.
Every instantiation of the parameters that satisfies these constraints leads to a valid loop invariant.
To find the strongest invariant, maximize a_1 and b_1 and minimize a_2 and b_2, just as we before maximized the l_i and minimized the u_i. This yields the max-plus polyhedron generated by the two points $\binom{0}{5}$ and $\binom{10}{10}$.

Summary

- Quantifier-elimination heuristics might be an alternative to abstract interpretation for program analysis.

Summary

- Quantifier-elimination heuristics might be an alternative to abstract interpretation for program analysis.
- Since general (complete) QE methods are very expensive and their outputs are hard to decipher, it is better to consider special cases, sacrificing completeness as well as generality.

Summary

- Quantifier-elimination heuristics might be an alternative to abstract interpretation for program analysis.
- Since general (complete) QE methods are very expensive and their outputs are hard to decipher, it is better to consider special cases, sacrificing completeness as well as generality.
- There is a real trade-off between resources/efficiency and precision/incompleteness.
- Many bells and whistles are needed, just like in the abstract interpretation approach.

Summary

- Quantifier-elimination heuristics might be an alternative to abstract interpretation for program analysis.
- Since general (complete) QE methods are very expensive and their outputs are hard to decipher, it is better to consider special cases, sacrificing completeness as well as generality.
- There is a real trade-off between resources/efficiency and precision/incompleteness.
- Many bells and whistles are needed, just like in the abstract interpretation approach.
- An implementation is in progress.