Satisfiability Checking

Interval Constraint Propagation

Prof. Dr. Erika Ábrahám

RWTH Aachen University
Informatik 2
LuFG Theory of Hybrid Systems

WS 16/17

Motivation

We consider input formulae φ from the theory of quantifier-free nonlinear real arithmetic (QFNRA):

$$
\begin{array}{lll}
p & := const \mid x \mid (p + p) \mid (p - p) \mid \boxed{(p \cdot p)} & \text{polynomials} \\
c & := p < 0 \mid p = 0 & \text{(polynomial) constraints} \\
\varphi & := c \mid (\varphi \wedge \varphi) \mid \neg\varphi & \text{QFNRA formulas}
\end{array}
$$

$\texttt{const} \in \mathbb{Q}$, $x \in Var(\varphi)$ (variables appearing in φ) take real values from $\mathbb{R}$

Motivation

We consider input formulae φ from the theory of quantifier-free nonlinear real arithmetic (QFNRA):

$$\begin{array}{lll} p & := \mathit{const} \mid x \mid (p+p) \mid (p-p) \mid (p \cdot p) & \text{polynomials} \\ c & := p < 0 \mid p = 0 & \text{(polynomial) constraints} \\ \varphi & := c \mid (\varphi \wedge \varphi) \mid \neg\varphi & \text{QFNRA formulas} \end{array}$$

$\texttt{const} \in \mathbb{Q}$, $x \in \mathit{Var}(\varphi)$ (variables appearing in φ) take real values from $\mathbb{R}$

- Best known methods for solving QFNRA problems have exponential complexity $\rightarrow$ hard to solve

Motivation

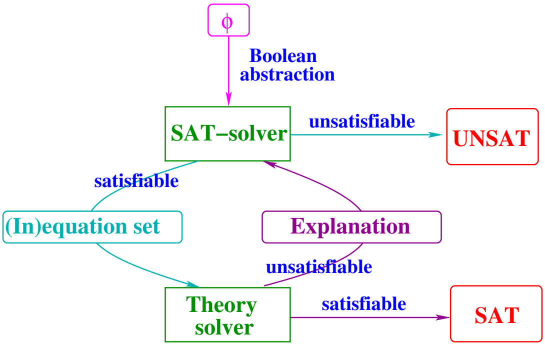
We consider input formulae φ from the theory of quantifier-free nonlinear real arithmetic (QFNRA):

$$
\begin{array}{lll}
p & := const \mid x \mid (p+p) \mid (p-p) \mid (p \cdot p) & \text{polynomials} \\
c & := p < 0 \mid p = 0 & \text{(polynomial) constraints} \\
\varphi & := c \mid (\varphi \wedge \varphi) \mid \neg\varphi & \text{QFNRA formulas}
\end{array}
$$

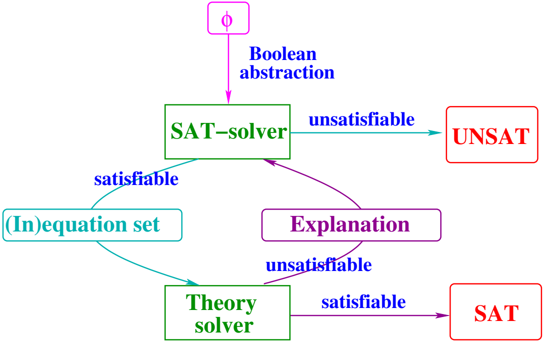
$\mathtt{const} \in \mathbb{Q}$, $x \in Var(\varphi)$ (variables appearing in φ) take real values from $\mathbb{R}$

- Best known methods for solving QFNRA problems have exponential complexity $\rightarrow$ hard to solve
- Approaches for solving QFNRA:
  - ICP
  - Virtual substitution (VS)
  - Cylindrical algebraic decomposition (CAD)
  - Gröbner bases

Interval constraint propagation (ICP) in SMT



Interval constraint propagation (ICP) in SMT



Interval constraint propagation:

- Incomplete method
- Cheap reduction of the search space

Intervals

$\mathbb{R} = (-\infty ; +\infty)$

Definition (Interval)

Intervals I are subsets of $\mathbb{R}$ of the form

- $[\ell; u] = \{x \in \mathbb{R} | \ell \leq x \leq u\}$, $\ell, u \in \mathbb{R}$
- $[\ell; u) = \{x \in \mathbb{R} | \ell \leq x < u\}$, $\ell \in \mathbb{R}$, $u \in \mathbb{R} \cup \{\infty\}$
- $(\ell; u] = \{x \in \mathbb{R} | \ell < x \leq u\}$, $\ell \in \mathbb{R} \cup \{-\infty\}$, $u \in \mathbb{R}$
- $(\ell; u) = \{x \in \mathbb{R} | \ell < x < u\}$, $\ell \in \mathbb{R} \cup \{-\infty\}$, $u \in \mathbb{R} \cup \{\infty\}$

where ℓ is called the lower bound and u the upper bound of I. Let $\mathbb{I}$ be the set of all intervals.

Intervals

Definition (Interval)

Intervals I are subsets of $\mathbb{R}$ of the form

- $[\ell; u] = \{x \in \mathbb{R} | \ell \leq x \leq u\}$, $\ell, u \in \mathbb{R}$
- $[\ell; u) = \{x \in \mathbb{R} | \ell \leq x < u\}$, $\ell \in \mathbb{R}$, $u \in \mathbb{R} \cup \{\infty\}$
- $(\ell; u] = \{x \in \mathbb{R} | \ell < x \leq u\}$, $\ell \in \mathbb{R} \cup \{-\infty\}$, $u \in \mathbb{R}$
- $(\ell; u) = \{x \in \mathbb{R} | \ell < x < u\}$, $\ell \in \mathbb{R} \cup \{-\infty\}$, $u \in \mathbb{R} \cup \{\infty\}$

where ℓ is called the lower bound and u the upper bound of I. Let $\mathbb{I}$ be the set of all intervals.

For simplicity, in the following we consider intervals of the form $I = [\ell, u]$, $[\ell, \infty)$ and $(-\infty, u]$ only.

Intervals and boxes

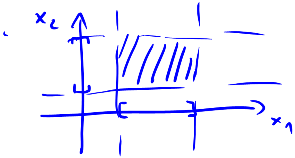
Definition (Interval diameter)

Width/diameter of interval $I = [\ell, u]$: $D_I = u - \ell$.

Definition (Interval box)

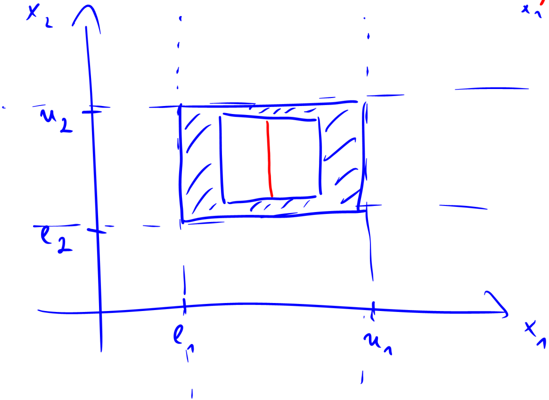
n-dimensional box $B = [\ell_1; u_1] \times \ldots \times [\ell_n; u_n] \in \mathbb{I}^n$.

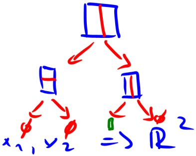
Let in the following $A = [\ell_a, u_a]$ and $B = [\ell_b, u_b]$ be non-empty intervals from $\mathbb{I}$.



Basic interval arithmetic

Idea: Extend real arithmetic to intervals.





Example

$x = y$

$x \in [1,2]$

$y \in [0,4]$

$B = [1,2] \times [0,4]$

$\Rightarrow y \in [1,2]$

$x = y^2$

$\Rightarrow y \in [1, \sqrt{2}]$

$x = y + z$

Basic interval arithmetic

Idea: Extend real arithmetic to intervals.

- Constants and variables are interval-valued
- Operations maintain all correct solutions

$$\left.\begin{array}{l} z = x \cdot y \\ y = 1 \end{array}\right\} \quad z = x$$

Operations:

$+ \,,\, - \,,\, \cdot \,,\, :$

$$\left.\begin{array}{l} z = x \cdot y \\ y = [0;2] \end{array}\right\} \quad z = [0;2] \cdot x$$

Basic interval arithmetic

Idea: Extend real arithmetic to intervals.

- Constants and variables are interval-valued
- Operations maintain all correct solutions

Example (Interval Addition)

$[-1;5] + [1;4] =$ $[0\,;\,9]$

$A + B \supseteq \{c \in \mathbb{R} \mid c = a + b,\ a \in A,\ b \in B\}$

Basic interval arithmetic

Idea: Extend real arithmetic to intervals.

- Constants and variables are interval-valued
- Operations maintain all correct solutions

Example (Interval Addition)

$[-1; 5] + [1; 4] = [0; 9]$

Basic interval arithmetic

Idea: Extend real arithmetic to intervals.

- Constants and variables are interval-valued
- Operations maintain all correct solutions

Example (Interval Addition)

$[-1;5] + [1;4] = [0;9]$
$[-2;3] + 4 =$

Basic interval arithmetic

Idea: Extend real arithmetic to intervals.

- Constants and variables are interval-valued
- Operations maintain all correct solutions

Example (Interval Addition)

$[-1; 5] + [1; 4] = [0; 9]$

$[-2; 3] + 4 = [-2; 3] + \underbrace{[4; 4]}_{[4]} = [2; 7]$

Basic interval arithmetic

Idea: Extend real arithmetic to intervals.

- Constants and variables are interval-valued
- Operations maintain all correct solutions

Example (Interval Addition)

$[-1;5] + [1;4] = [0;9]$

$[-2;3] + 4 = [-2;3] + \underbrace{[4;4]}_{[4]} = [2;7]$

Definition (Interval addition)

$A + B = [l_a + l_b, u_a + u_b]$

$A = [l_a, u_a] \quad B = [l_b, u_b]$

Basic interval arithmetic

Idea: Extend real arithmetic to intervals.

- Constants and variables are interval-valued
- Operations maintain all correct solutions

Example (Interval Addition)

$[-1;5] + [1;4] = [0;9]$

$[-2;3] + 4 = [-2;3] + \underbrace{[4;4]}_{[4]} = [2;7]$

Definition (Interval addition)

$A + B = [\ell_a + \ell_b; u_a + u_b]$

Basic interval arithmetic

Example (Subtraction)

$[-1; 5] - [1; 4] =$ $[-1-4\ ;\ 5-1] = [-5, 4]$

Basic interval arithmetic

Example (Subtraction)

$[-1;5] - [1;4] = [-5;4]$
$[-2;3] - 4 = [-2-4, 3-4] = [-6, -1]$

Basic interval arithmetic

Example (Subtraction)

$[-1;5] - [1;4] = [-5;4]$
$[-2;3] - 4 = [-2;3] + [4] = [-6;-1]$

Definition (Interval subtraction)

$A - B = [l_a - u_b, u_a - l_b]$

$A - B = \{a - b \mid a \in A, b \in B\}$ $\Rightarrow$ exact

Basic interval arithmetic

Example (Subtraction)

$[-1;5] - [1;4] = [-5;4]$
$[-2;3] - 4 = [-2;3] + [4] = [-6;-1]$

Definition (Interval subtraction)

$A - B = [\ell_a - u_b; u_a - \ell_b]$

Example (Multiplication)

$[-1;5] \cdot [1;4] = [\underset{-1 \cdot 4}{\underset{\uparrow}{-4}} ; \underset{5 \cdot 4}{\underset{\uparrow}{20}}]$

$[1;2] \cdot [2;3] = [1 \cdot 2 ; 2 \cdot 3] = [2;6]$
$[-1;2] \cdot [2;3] = [-1 \cdot 3 ; 2 \cdot 3] = [-3;6]$
$[-1;2] \cdot [-2;3] = [-4;6]$

Basic interval arithmetic

Example (Subtraction)

$[-1;5] - [1;4] = [-5;4]$
$[-2;3] - 4 = [-2;3] + [4] = [-6;-1]$

Definition (Interval subtraction)

$A - B = [\ell_a - u_b; u_a - \ell_b]$

Example (Multiplication)

$[-1;5] \cdot [1;4] = [-4;20]$
$[-2;3] \cdot 4 =$

Basic interval arithmetic

Example (Subtraction)

$[-1;5] - [1;4] = [-5;4]$
$[-2;3] - 4 = [-2;3] + [4] = [-6;-1]$

Definition (Interval subtraction)

$A - B = [\ell_a - u_b; u_a - \ell_b]$

Example (Multiplication)

$[-1;5] \cdot [1;4] = [-4;20]$
$[-2;3] \cdot 4 = \cancel{[-2;3] + [4]} = [-8;12]$ $[-2;3] \cdot [-4;-4] = [-12;8]$

Definition (Interval multiplication)

Basic interval arithmetic

Example (Subtraction)

$[-1;5] - [1;4] = [-5;4]$
$[-2;3] - 4 = [-2;3] + [4] = [-6;-1]$

Definition (Interval subtraction)

$A - B = [\ell_a - u_b; u_a - \ell_b]$

Example (Multiplication)

$[-1;5] \cdot [1;4] = [-4;20]$
$[-2;3] \cdot 4 = [-2;3] + [4] = [-8;12]$

Definition (Interval multiplication)

$A \cdot B = [min(\ell_a \cdot \ell_b, \ell_a \cdot u_b, u_a \cdot \ell_b, u_a \cdot u_b); max(\ell_a \cdot \ell_b, \ell_a \cdot u_b, u_a \cdot \ell_b, u_a \cdot u_b)]$

Basic interval arithmetic

Special case of multiplication: Square

Basic interval arithmetic

Special case of multiplication: Square

Example (Square)

$[-1; 5]^2 =$

$$\underset{x}{[-1;5]} \cdot \underset{y}{[-1;5]} = \underset{x \cdot y}{[-5;25]}$$

$$\underset{x}{[-1;5]} \cdot \underset{x}{[-1;5]} = [0;25]$$

Basic interval arithmetic

Special case of multiplication: Square

Example (Square)

$[-1;5]^2 = [0;25]$

Definition (Interval square)

Squaring an interval can only result in positive values:
$A \in \mathbb{I} : A^2 = (A \cdot A) \cap [0;+\infty)$

$[l;u] \cdot [l;u]$

$[l;u]^2$

Basic interval arithmetic

Special case of multiplication: Square

Example (Square)

$[-1;5]^2 = [0;25]$

Definition (Interval square)

Squaring an interval can only result in positive values:
$A \in \mathbb{I} : A^2 = (A \cdot A) \cap [0; +\infty)$

Example (Division)

$[2;3] \div [4;5] = \left[\frac{2}{5} ; \frac{3}{4}\right] = [2;3] \cdot \frac{1}{[4;5]} = [2;3] \cdot \left[\frac{1}{5} ; \frac{1}{4}\right]$

Basic interval arithmetic

Special case of multiplication: Square

Example (Square)

$[-1;5]^2 = [0;25]$

Definition (Interval square)

Squaring an interval can only result in positive values:
$A \in \mathbb{I} : A^2 = (A \cdot A) \cap [0; +\infty)$

Example (Division)

$[2;3] \div [4;5] = [2;3] \cdot \frac{1}{B} = [2;3] \cdot [\frac{1}{5}; \frac{1}{4}] = [\frac{2}{5}; \frac{3}{4}]$

Definition (Interval division ($0 \notin B$))

$A / B = \left[\frac{l_a}{u_b} ; \frac{u_a}{l_b} \right]$

Basic interval arithmetic

Special case of multiplication: Square

Example (Square)

$[-1;5]^2 = [0;25]$

Definition (Interval square)

Squaring an interval can only result in positive values:
$A \in \mathbb{I} : A^2 = (A \cdot A) \cap [0;+\infty)$

Example (Division)

$[2;3] \div [4;5] = [2;3] \cdot \frac{1}{B} = [2;3] \cdot [\frac{1}{5};\frac{1}{4}] = [\frac{2}{5};\frac{3}{4}]$

Definition (Interval division ($0 \notin B$))

$A, B \in \mathbb{I} : A \div B = A \cdot \frac{1}{B} = A \cdot [\frac{1}{u_b};\frac{1}{\ell_b}]$

$[-1;2] \div [2;4] = [-1;2] \cdot [\frac{1}{4};\frac{1}{2}]$
$= [-\frac{1}{2};1]$

Extended interval arithmetic

Problem: Division A/B (B may contain 0):

Extended interval arithmetic

Problem: Division A/B (B may contain 0):

Example (Division by zero)

$[1;3] \div [-2;3] = [1;3] \cdot \underbrace{[\frac{1}{3};\frac{1}{-2}]}_{\emptyset\ !} = \emptyset$

Extended interval arithmetic

Problem: Division A/B (B may contain 0):

Example (Division by zero)

$[1;3] \div [-2;3] = [1;3] \cdot [\frac{1}{3}; -\frac{1}{2}] \rightarrow$ invalid bounds

Extended interval arithmetic

Problem: Division A/B (B may contain 0):

Example (Division by zero)

$[1;3] \div [-2;3] = [1;3] \cdot [\frac{1}{3}; -\frac{1}{2}] \rightarrow$ invalid bounds

Boundwise analysis on divisors:

- $\frac{1}{3} \in A/B$
- $\frac{1}{2} \in A/B$
- $\frac{1}{0.5} \in A/B$

$a \in A$ $\qquad b \in B \cap (0, \infty)$

$\frac{a}{b} < \frac{a+\delta}{b} \qquad \delta > 0$

$\lim_{v \to 0} \frac{1}{v} \rightarrow \infty$

Extended interval arithmetic

Problem: Division A/B (B may contain 0):

Example (Division by zero)

$[1;3] \div [-2;3] = [1;3] \cdot [\frac{1}{3}; -\frac{1}{2}] \rightarrow$ invalid bounds
Boundwise analysis on divisors:

- $\frac{1}{3} \in A/B$
- $\frac{1}{2} \in A/B$
- $\frac{1}{0.5} \in A/B$

$\rightarrow$ maximal divisor $b = 3 \rightarrow [\frac{1}{3}; \infty)$

Extended interval arithmetic

Problem: Division A/B (B may contain 0):

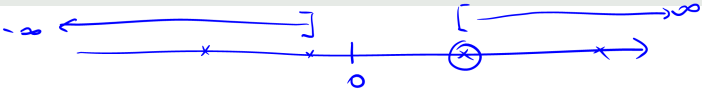
Example (Division by zero)

$[1;3] \div [-2;3] = [1;3] \cdot [\frac{1}{3}; -\frac{1}{2}] \rightarrow$ invalid bounds

Boundwise analysis on divisors:

- $\frac{1}{3} \in A/B$
- $\frac{1}{2} \in A/B$
- $\frac{1}{0.5} \in A/B$
- $\frac{1}{-2} \in A/B$
- $\frac{1}{-1} \in A/B$
- $\frac{1}{-0.5} \in A/B$

$\rightarrow$ maximal divisor $b = 3 \rightarrow [\frac{1}{3}; \infty)$



$l_a \cdot l_b$
$l_a \cdot u_b$
$u_a \cdot l_b$
$u_a \; u_b$

Extended interval arithmetic

Problem: Division A/B (B may contain 0):

Example (Division by zero)

$[1;3] \div [-2;3] = [1;3] \cdot [\frac{1}{3}; -\frac{1}{2}] \rightarrow$ invalid bounds

Boundwise analysis on divisors:

- $\frac{1}{3} \in A/B$
- $\frac{1}{2} \in A/B$
- $\frac{1}{0.5} \in A/B$

$\rightarrow$ maximal divisor $b = 3 \rightarrow [\frac{1}{3}; \infty)$

- $\frac{1}{-2} \in A/B$
- $\frac{1}{-1} \in A/B$
- $\frac{1}{-0.5} \in A/B$

$\rightarrow$ minimal divisor $b = -2 \rightarrow (-\infty; -\frac{1}{2}]$

Extended interval arithmetic

Problem: Division A/B (B may contain 0): $0 \notin A \quad A = [l_a, u_a]$

Example (Division by zero)

$[1;3] \div [-2;3] = [1;3] \cdot [\frac{1}{3}; -\frac{1}{2}] \rightarrow$ invalid bounds

Boundwise analysis on divisors:

- $\frac{1}{3} \in A/B$
- $\frac{1}{2} \in A/B$
- $\frac{1}{0.5} \in A/B$

$\rightarrow$ maximal divisor $b = 3 \rightarrow [\frac{1}{3}; \infty)$

- $\frac{1}{-2} \in A/B$
- $\frac{1}{-1} \in A/B$
- $\frac{1}{-0.5} \in A/B$

$\rightarrow$ minimal divisor $b = -2 \rightarrow (-\infty; -\frac{1}{2}]$

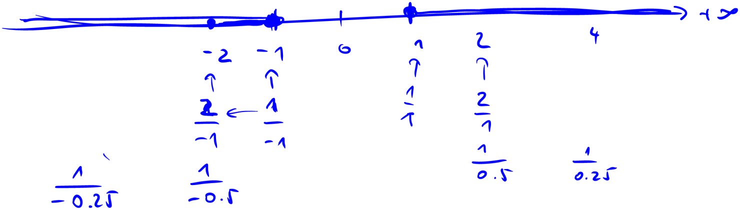
Introduce new $\boxed{\text{rules}}$ for extended interval division such that:

$[1;3]/[-2;3] = (-\infty; \frac{1}{-2}] \cup [\frac{1}{3}; +\infty)$

Note: Resulting inteval may contain a gap!

$1 \cdot (-2) \rightarrow -2 \Rightarrow$ largest neg.
$1 \cdot (3) \rightarrow \frac{1}{3} \Rightarrow$ smallest pos
$3 \cdot (-2) \rightarrow -6$
$3 \cdot 3 \rightarrow 9$

$$\underbrace{[1,2]}_{A} \div \underbrace{[-1,1]}_{B} \qquad 0 \notin A$$



Interval constraint propagation

Assume a set of QFNRA constraints (as theory solvers get in SMT solving) and some interval bounds for the variables in the constraints.

Interval constraint propagation

Assume a set of QFNRA constraints (as theory solvers get in SMT solving) and some interval bounds for the variables in the constraints.
Use interval constraint propagation to sharpen bounds:

- Insert current bounds for all except one variable in one constraint
- Apply interval arithmetic to gain new bounds
- Update bounds via intersection

Interval constraint propagation

Assume a set of QFNRA constraints (as theory solvers get in SMT solving) and some interval bounds for the variables in the constraints.

Use interval constraint propagation to sharpen bounds:

- Insert current bounds for all except one variable in one constraint
- Apply interval arithmetic to gain new bounds
- Update bounds via intersection

Example (Propagation)

$x \in [1;3], y \in [1,2], c_1 : y = x, c_2 : y = x^2$

Interval constraint propagation

Assume a set of QFNRA constraints (as theory solvers get in SMT solving) and some interval bounds for the variables in the constraints.
Use interval constraint propagation to sharpen bounds:

- Insert current bounds for all except one variable in one constraint
- Apply interval arithmetic to gain new bounds
- Update bounds via intersection

Example (Propagation)

$x \in [1;3], y \in [1,2], c_1 : y = x, c_2 : y = x^2$
$c_2, x : x = \pm\sqrt{y}$

Interval constraint propagation

Assume a set of QFNRA constraints (as theory solvers get in SMT solving) and some interval bounds for the variables in the constraints.

Use interval constraint propagation to sharpen bounds:

- Insert current bounds for all except one variable in one constraint
- Apply interval arithmetic to gain new bounds
- Update bounds via intersection

Example (Propagation)

$x \in [1;3], y \in [1,2], c_1 : y = x, c_2 : y = x^2$

$c_2, x : x = \pm\sqrt{y} \rightarrow x = \pm\sqrt{[1;2]} = [-\sqrt{2};-1]] \cup [1;\sqrt{2}]$

Interval constraint propagation

Assume a set of QFNRA constraints (as theory solvers get in SMT solving) and some interval bounds for the variables in the constraints.

Use interval constraint propagation to sharpen bounds:

- Insert current bounds for all except one variable in one constraint
- Apply interval arithmetic to gain new bounds
- Update bounds via intersection

Example (Propagation)

$x \in [1;3], y \in [1,2], c_1 : y = x, c_2 : y = x^2$

$c_2, x : x = \pm\sqrt{y} \rightarrow x = \pm\sqrt{[1;2]} = [-\sqrt{2}; -1]] \cup [1; \sqrt{2}] \rightarrow$

$\quad x \in [1;3] \cap ([-\sqrt{2}; -1] \cup [1; \sqrt{2}]) = [1; \sqrt{2}]$

Interval constraint propagation

Assume a set of QFNRA constraints (as theory solvers get in SMT solving) and some interval bounds for the variables in the constraints.

Use interval constraint propagation to sharpen bounds:

- Insert current bounds for all except one variable in one constraint
- Apply interval arithmetic to gain new bounds
- Update bounds via intersection

Example (Propagation)

$x \in [1;3], y \in [1,2], c_1 : y = x, c_2 : y = x^2$

$c_2, x : x = \pm\sqrt{y} \rightarrow x = \pm\sqrt{[1;2]} = [-\sqrt{2}; -1]] \cup [1; \sqrt{2}] \rightarrow$

$\quad x \in [1;3] \cap ([-\sqrt{2}; -1] \cup [1; \sqrt{2}]) = [1; \sqrt{2}]$

$c_1, y : y = x$

Interval constraint propagation

Assume a set of QFNRA constraints (as theory solvers get in SMT solving) and some interval bounds for the variables in the constraints.
Use interval constraint propagation to sharpen bounds:

- Insert current bounds for all except one variable in one constraint
- Apply interval arithmetic to gain new bounds
- Update bounds via intersection

Example (Propagation)

$x \in [1;3], y \in [1,2], c_1 : y = x, c_2 : y = x^2$
$c_2, x : x = \pm\sqrt{y} \rightarrow x = \pm\sqrt{[1;2]} = [-\sqrt{2};-1]] \cup [1;\sqrt{2}] \rightarrow$
$\quad x \in [1;3] \cap ([-\sqrt{2};-1] \cup [1;\sqrt{2}]) = [1;\sqrt{2}]$
$c_1, y : y = x \rightarrow y = [1;\sqrt{2}]$

Interval constraint propagation

Assume a set of QFNRA constraints (as theory solvers get in SMT solving) and some interval bounds for the variables in the constraints.

Use interval constraint propagation to sharpen bounds:

- Insert current bounds for all except one variable in one constraint
- Apply interval arithmetic to gain new bounds
- Update bounds via intersection

Example (Propagation)

$x \in [1;3], y \in [1,2], c_1 : y = x, c_2 : y = x^2$

$c_2, x : x = \pm\sqrt{y} \to x = \pm\sqrt{[1;2]} = [-\sqrt{2};-1]] \cup [1;\sqrt{2}] \to$

$\quad x \in [1;3] \cap ([-\sqrt{2};-1] \cup [1;\sqrt{2}]) = [1;\sqrt{2}]$

$c_1, y : y = x \to y = [1;\sqrt{2}] \to y \in [1;2] \cap [1;\sqrt{2}] = [1;\sqrt{2}]$

Interval constraint propagation

Assume a set of QFNRA constraints (as theory solvers get in SMT solving) and some interval bounds for the variables in the constraints.

Use interval constraint propagation to sharpen bounds:

- Insert current bounds for all except one variable in one constraint
- Apply interval arithmetic to gain new bounds
- Update bounds via intersection

Example (Propagation)

$x \in [1;3], y \in [1,2], c_1 : y = x, c_2 : y = x^2$

$c_2, x : x = \pm\sqrt{y} \rightarrow x = \pm\sqrt{[1;2]} = [-\sqrt{2};-1]] \cup [1;\sqrt{2}] \rightarrow$

$\quad x \in [1;3] \cap ([-\sqrt{2};-1] \cup [1;\sqrt{2}]) = [1;\sqrt{2}]$

$c_1, y : y = x \rightarrow y = [1;\sqrt{2}] \rightarrow y \in [1;2] \cap [1;\sqrt{2}] = [1;\sqrt{2}]$

Contraction sequence:

$x : [1;3] \overset{c_2,x}{\rightarrow} [1;\sqrt{2}] \overset{c_2,x}{\rightarrow} [1;\sqrt[4]{2}] \overset{c_2,x}{\rightarrow} [1;\sqrt[8]{2}] \overset{c_2,x}{\rightarrow} \ldots \rightsquigarrow [1;1]$

$y : [1;2] \overset{c_1,y}{\rightarrow} [1;\sqrt{2}] \overset{c_1,y}{\rightarrow} [1;\sqrt[4]{2}] \overset{c_1,y}{\rightarrow} [1;\sqrt[8]{2}] \overset{c_1,y}{\rightarrow} \ldots \rightsquigarrow [1;1]$

Interval constraint propagation

Assume a set of QFNRA constraints (as theory solvers get in SMT solving) and some interval bounds for the variables in the constraints.

Use interval constraint propagation to sharpen bounds:

- Insert current bounds for all except one variable in one constraint
- Apply interval arithmetic to gain new bounds
- Update bounds via intersection

Example (Propagation)

$x \in [1;3], y \in [1,2], c_1 : y = x, c_2 : y = x^2$

$c_2, x : x = \pm\sqrt{y} \to x = \pm\sqrt{[1;2]} = [-\sqrt{2};-1]] \cup [1;\sqrt{2}] \to$

$\quad x \in [1;3] \cap ([-\sqrt{2};-1] \cup [1;\sqrt{2}]) = [1;\sqrt{2}]$

$c_1, y : y = x \to y = [1;\sqrt{2}] \to y \in [1;2] \cap [1;\sqrt{2}] = [1;\sqrt{2}]$

Contraction sequence:

$x : [1;3] \overset{c_2,x}{\to} [1;\sqrt{2}] \overset{c_2,x}{\to} [1;\sqrt[4]{2}] \overset{c_2,x}{\to} [1;\sqrt[8]{2}] \overset{c_2,x}{\to} \ldots \rightsquigarrow [1;1]$

$y : [1;2] \overset{c_1,y}{\to} [1;\sqrt{2}] \overset{c_1,y}{\to} [1;\sqrt[4]{2}] \overset{c_1,y}{\to} [1;\sqrt[8]{2}] \overset{c_1,y}{\to} \ldots \rightsquigarrow [1;1]$

We define a pair of constraint and variable as a contraction candidate (CC).

Preprocessing

Propagation requires constraints to be solvable in one variable.

Preprocessing

Propagation requires constraints to be solvable in one variable.
$\rightarrow$ Preprocessing required: Linearization.

Preprocessing

Propagation requires constraints to be solvable in one variable.
$\rightarrow$ Preprocessing required: Linearization.

- Separate linear and nonlinear constraints

Preprocessing

Propagation requires constraints to be solvable in one variable.
$\rightarrow$ Preprocessing required: Linearization.

- Separate linear and nonlinear constraints
- Introduce new variables for top-level nonlinear operands

Preprocessing

Propagation requires constraints to be solvable in one variable.
$\rightarrow$ Preprocessing required: Linearization.

- Separate linear and nonlinear constraints
- Introduce new variables for top-level nonlinear operands
- Obtain two sets: Linear constraints (including linearizations) and "smaller" nonlinear constraints

Preprocessing

Propagation requires constraints to be solvable in one variable.
$\rightarrow$ Preprocessing required: Linearization.

- Separate linear and nonlinear constraints
- Introduce new variables for top-level nonlinear operands
- Obtain two sets: Linear constraints (including linearizations) and "smaller" nonlinear constraints

Preprocessing

Toy example:

$$x^2 \cdot y + z = 0$$

Preprocessing

Propagation requires constraints to be solvable in one variable.
$\rightarrow$ Preprocessing required: Linearization.

- Separate linear and nonlinear constraints
- Introduce new variables for top-level nonlinear operands
- Obtain two sets: Linear constraints (including linearizations) and "smaller" nonlinear constraints

Preprocessing

Toy example:

$$x^2 \cdot y + z = 0$$

$$v_1 + z = 0 \land v_1 = x^2 \cdot y$$

Algorithm

Input:

- Conjunction of constraints
- Variables bounded by intervals (initial box)

Algorithm

Input:

- Conjunction of constraints
- Variables bounded by intervals (initial box)

Goal

Reduce the size of the input box until either an empty box is achieved (in which case the current problem is unsatisfiable) or a specified diameter is reached, while guaranteeing that all solutions (if any) are still contained in the box.

Algorithm

Input:

- Conjunction of constraints
- Variables bounded by intervals (initial box)

Goal

Reduce the size of the input box until either an empty box is achieved (in which case the current problem is unsatisfiable) or a specified diameter is reached, while guaranteeing that all solutions (if any) are still contained in the box.

If the box did not get empty then we pass the problem restricted to the gained box (as solution candidate) to a backend implementing a complete method.

Algorithm

Input:

- Conjunction of constraints
- Variables bounded by intervals (initial box)

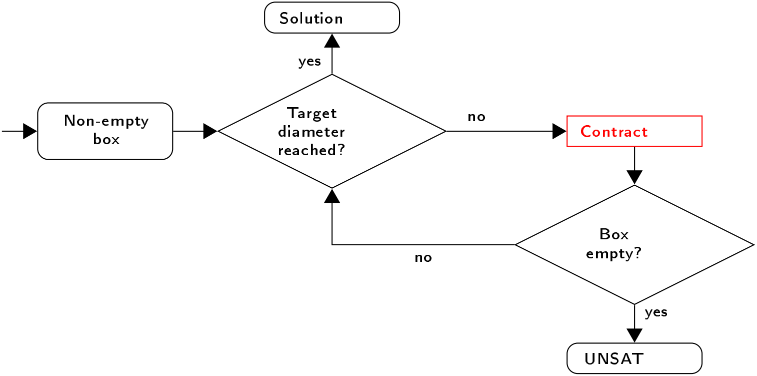
Goal

Reduce the size of the input box until either an empty box is achieved (in which case the current problem is unsatisfiable) or a specified diameter is reached, while guaranteeing that all solutions (if any) are still contained in the box.

If the box did not get empty then we pass the problem restricted to the gained box (as solution candidate) to a backend implementing a complete method.

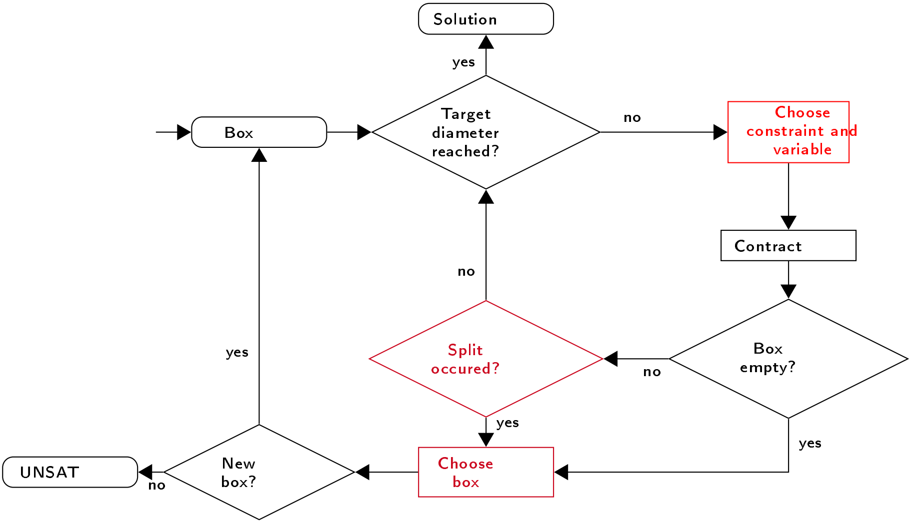
- We stop contraction when some diameter is reached (after a number of contraction steps or)
- Complete methods profit from reduced search space (but suffer from more calls)

Algorithm



Remark: Due to interval division propagation may result in two intervals (split)

Algorithm - extended



Algorithm

Input:

- Conjunction of constraints
- Variables bounded by intervals (initial box)

Goal:

- Goal: Reduce box for complete methods (backend)

Algorithmic aspects:

- Method for contraction

Algorithm

Input:

- Conjunction of constraints
- Variables bounded by intervals (initial box)

Goal:

- Goal: Reduce box for complete methods (backend)

Algorithmic aspects:

- Method for contraction
- Data structure to keep track of boxes

Algorithm

Input:

- Conjunction of constraints
- Variables bounded by intervals (initial box)

Goal:

- Goal: Reduce box for complete methods (backend)

Algorithmic aspects:

- Method for contraction
- Data structure to keep track of boxes
- Heuristics to choose CCs (constraints and variables)

Algorithm

Input:

- Conjunction of constraints
- Variables bounded by intervals (initial box)

Goal:

- Goal: Reduce box for complete methods (backend)

Algorithmic aspects:

- Method for contraction
- Data structure to keep track of boxes
- Heuristics to choose CCs (constraints and variables)
- Heuristics to choose next box

Contraction

General approach: Contract via interval constraint propagation.

Contraction

General approach: Contract via interval constraint propagation.

Problems:

- Contraction gain is in general not predictable

Contraction

General approach: Contract via interval constraint propagation.
Problems:

- Contraction gain is in general not predictable
- Contraction may stop before target diameter reached

Contraction

General approach: Contract via interval constraint propagation.
Problems:

- Contraction gain is in general not predictable
- Contraction may stop before target diameter reached
- Contraction may cause a split (heteronomous split)

Contraction

General approach: Contract via interval constraint propagation.
Problems:

- Contraction gain is in general not predictable
- Contraction may stop before target diameter reached
- Contraction may cause a split (heteronomous split)

Example (Contraction candidate choice)

Contraction

General approach: Contract via interval constraint propagation.
Problems:

- Contraction gain is in general not predictable
- Contraction may stop before target diameter reached
- Contraction may cause a split (heteronomous split)

Example (Contraction candidate choice)

Consider $\{c_1 : y = x, c_2 : y = x^2\}$ with initial intervals $I_x := [1, 3]$ and $I_y := [1, 2]$

Contraction

General approach: Contract via interval constraint propagation.
Problems:

- Contraction gain is in general not predictable
- Contraction may stop before target diameter reached
- Contraction may cause a split (heteronomous split)

Example (Contraction candidate choice)

Consider $\{c_1 : y = x, c_2 : y = x^2\}$ with initial intervals $I_x := [1,3]$ and $I_y := [1,2]$
At each step we can consider 4 contractions:

- $I_x \overset{c_1,x}{\rightarrow} [1,2]$ $\quad (gain_{rel} : 0.5)$
- $I_y \overset{c_1,y}{\rightarrow} [1,2]$ $\quad (gain_{rel} : 0)$
- $I_x \overset{c_2,x}{\rightarrow} [1,\sqrt{2}]$ $\quad (gain_{rel} : 0.793)$
- $I_y \overset{c_2,y}{\rightarrow} [1,2]$ $\quad (gain_{rel} : 0)$

Relative contraction:

$$gain_{rel} = \frac{D_{old} - D_{new}}{D_{old}} = 1 - \frac{D_{new}}{D_{old}}$$

$\rightarrow$ Contraction gain varies.

Reinforced learning

We can improve the choice of CCs by heuristics:

- The algorithm selects the next contraction candidate with the highest weight $W_k^{(ij)} \in [0; 1]$.

Reinforced learning

We can improve the choice of CCs by heuristics:

- The algorithm selects the next contraction candidate with the highest weight $W_k^{(ij)} \in [0;1]$.
- Afterwards the weight is updated (according to the relative contraction $r_{k+1}^{(ij)} \in [0;1]$).

Reinforced learning

We can improve the choice of CCs by heuristics:

- The algorithm selects the next contraction candidate with the highest weight $W_k^{(ij)} \in [0; 1]$.
- Afterwards the weight is updated (according to the relative contraction $r_{k+1}^{(ij)} \in [0; 1]$).

Weight updating:

$$W_{k+1}^{(ij)} = W_k^{(ij)} + \alpha(r_{k+1}^{(ij)} - W_k^{(ij)})$$

Reinforced learning

We can improve the choice of CCs by heuristics:

- The algorithm selects the next contraction candidate with the highest weight $W_k^{(ij)} \in [0;1]$.
- Afterwards the weight is updated (according to the relative contraction $r_{k+1}^{(ij)} \in [0;1]$).

Weight updating:

$$W_{k+1}^{(ij)} = W_k^{(ij)} + \alpha(r_{k+1}^{(ij)} - W_k^{(ij)})$$

The factor $\alpha \in [0;1]$ decides how the importance of the events is rated:

- Large α (e.g. 0.9) $\rightarrow$ The last recent event is most important
- Small α (e.g. 0.1) $\rightarrow$ The initial weight is most important

CCs with a weight less than some threshold ε are not considered for contraction.

Contraction

Requirements for contraction operator:

Contraction

Requirements for contraction operator:

- Does not drop solutions
- Detects if no solution is contained in the current box

Contraction

Requirements for contraction operator:

- Does not drop solutions
- Detects if no solution is contained in the current box

The second usually cannot be guaranteed.

Contraction

Requirements for contraction operator:

- Does not drop solutions
- Detects if no solution is contained in the current box

The second usually cannot be guaranteed.

Usual contraction operators:

- Interval constraint propagation (original contractor)
- Interval Newton method

Contraction

Requirements for contraction operator:

- Does not drop solutions
- Detects if no solution is contained in the current box

The second usually cannot be guaranteed.

Usual contraction operators:

- Interval constraint propagation (original contractor)
- Interval Newton method

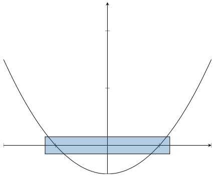
General approach: Contract as long as contraction gain is large enough.

Splitting

When the weight of all CCs is below the threshold we do not make progress
$\rightarrow$ split manually (autonomous split).

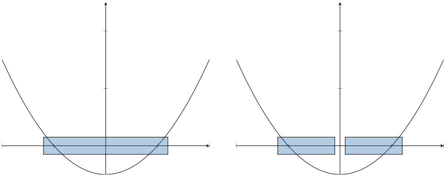
Splitting

When the weight of all CCs is below the threshold we do not make progress
$\rightarrow$ split manually (autonomous split).



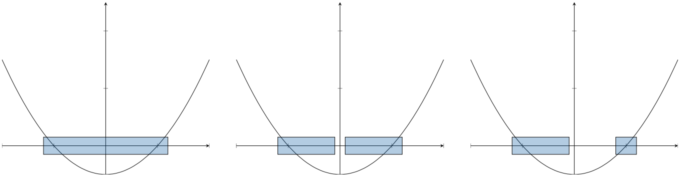
Splitting

When the weight of all CCs is below the threshold we do not make progress
$\rightarrow$ split manually (autonomous split).



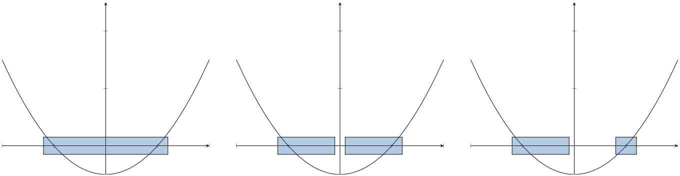
Splitting

When the weight of all CCs is below the threshold we do not make progress $\rightarrow$ split manually (autonomous split).



Splitting

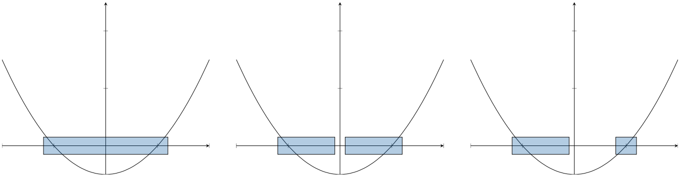
When the weight of all CCs is below the threshold we do not make progress
$\rightarrow$ split manually (autonomous split).



Problems:

Splitting

When the weight of all CCs is below the threshold we do not make progress
$\rightarrow$ split manually (autonomous split).

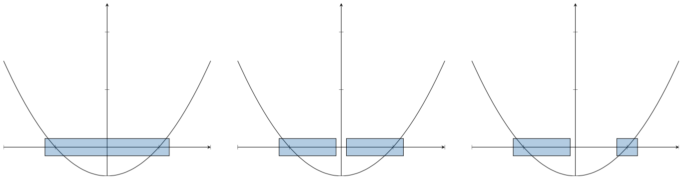


Problems:

- How to store boxes

Splitting

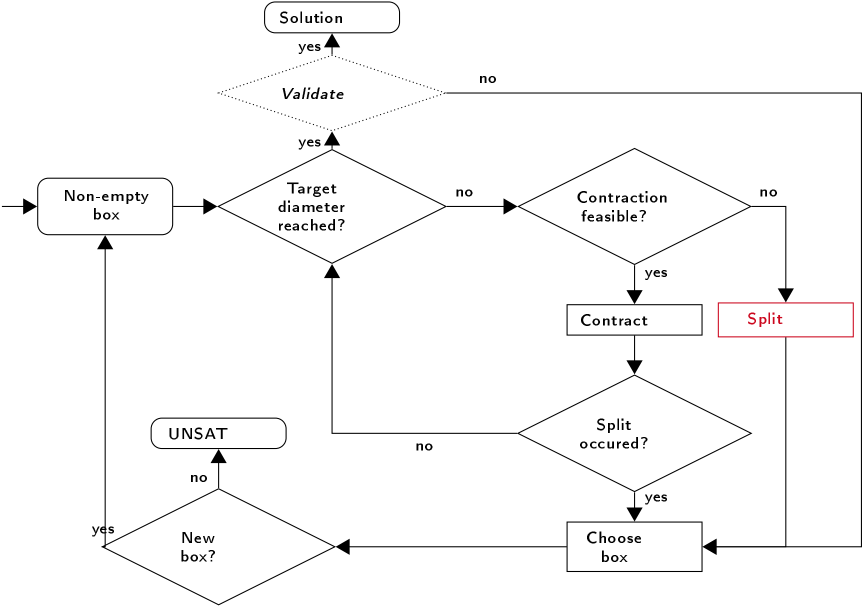
When the weight of all CCs is below the threshold we do not make progress $\rightarrow$ split manually (autonomous split).



Problems:

- How to store boxes
- How to select the next box

Algorithm overview



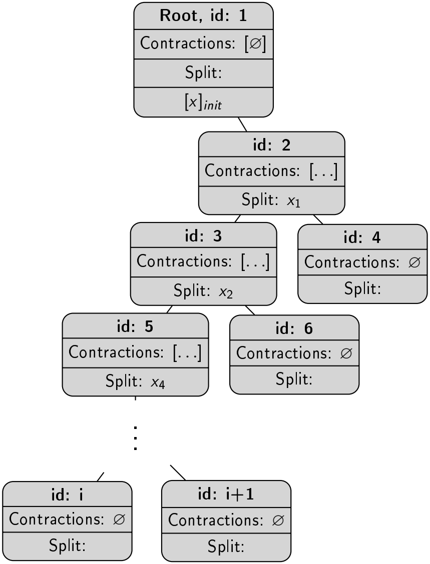
History tree

To keep track of current status we utilize a tree-structure, which holds solver states:

- Search box
- Applied contractions
- Splitting dimension

To keep track of current status we utilize a tree-structure, which holds solver states:

- Search box
- Applied contractions
- Splitting dimension

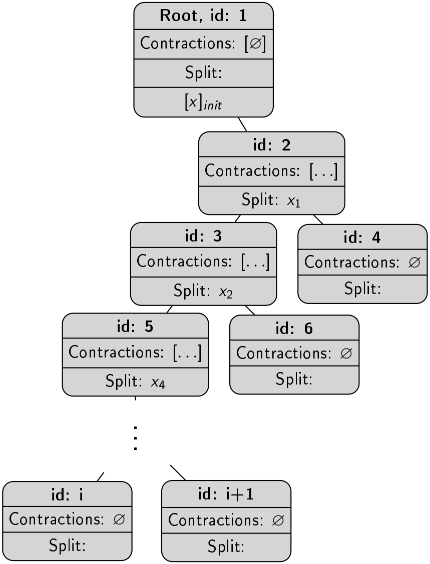


History tree

Additionally we can use the tree to collect infeasible subsets:

1. Infeasible box → propagate reasons to parent
2. (optional) skip boxes

→ We generate a set of constraints which includes the infeasible subset.

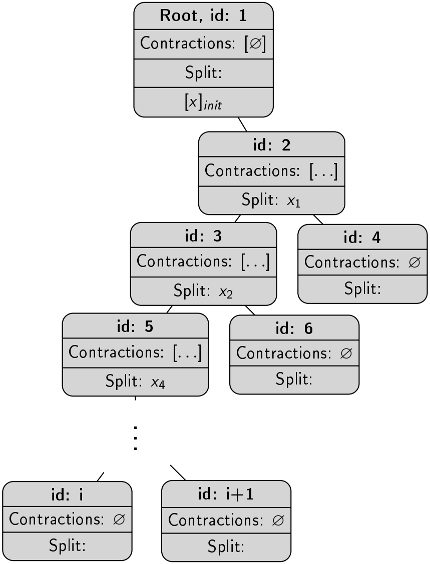


History tree

Additionally we can use the tree to collect infeasible subsets:

1. Infeasible box → propagate reasons to parent
2. (optional) skip boxes

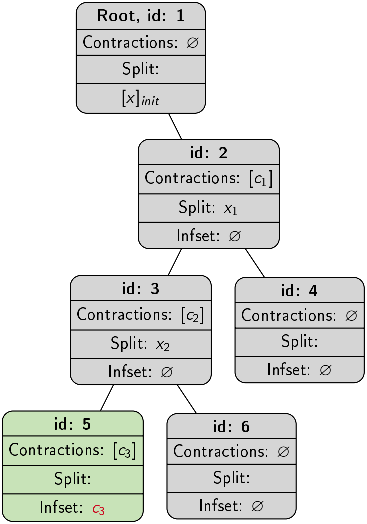
→ We generate a set of constraints which includes the infeasible subset.



If no further heuristics is applied we traverse the tree pre-order.

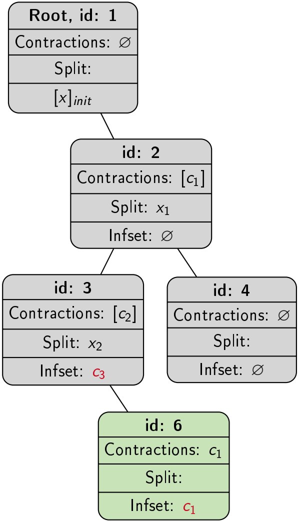
History tree example

Setup: constraints $c_1, \ldots, c_4$, variables $x_1, \ldots, x_3$



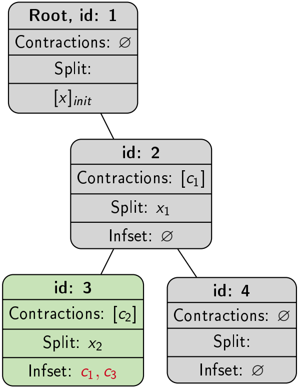
History tree example

Setup: constraints $c_1, \ldots, c_4$, variables $x_1, \ldots, x_3$



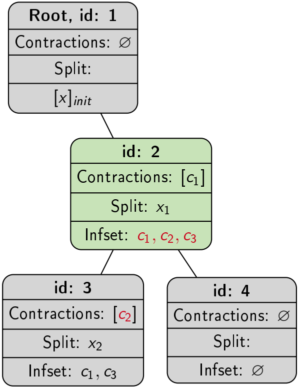
History tree example

Setup: constraints $c_1, \ldots, c_4$, variables $x_1, \ldots, x_3$



History tree example

Setup: constraints $c_1, \ldots, c_4$, variables $x_1, \ldots, x_3$

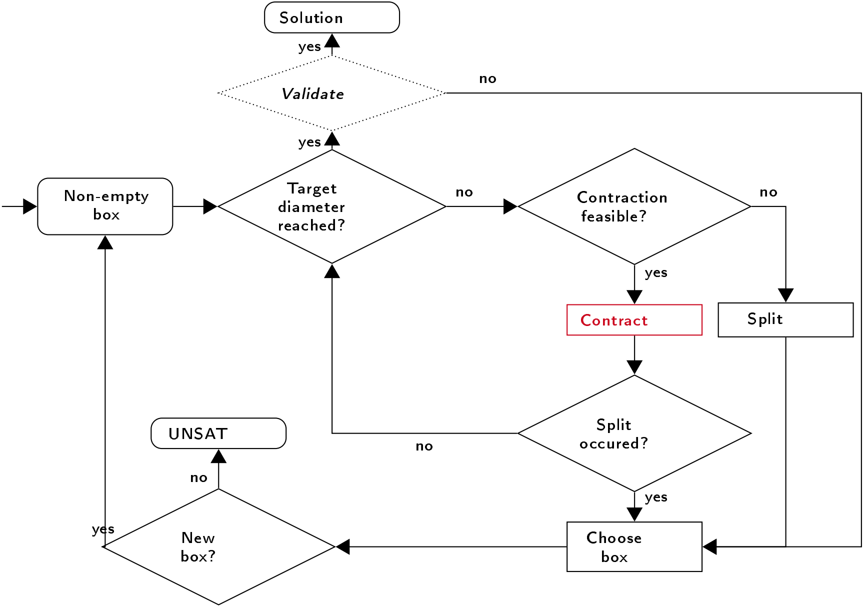


Improvements

Additional improvements:

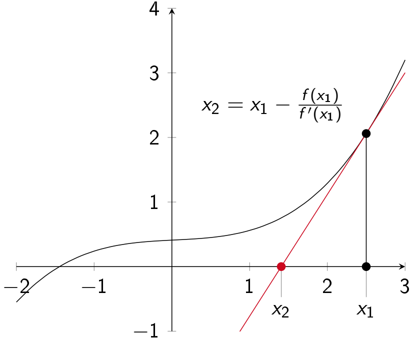
- More sophisticated contraction (Interval Newton method)
- Use linear (LRA) solver for linear constraints
- Introduce box validation
- Involve SAT-Solver for box-choice
- Introduce splitting heuristics

Algorithm overview



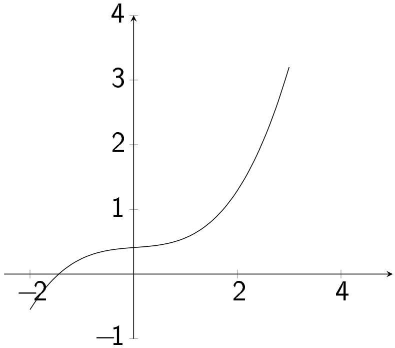
Interval Newton method

Reminder: Newton method for root finding (univariate polynomials):



Interval Newton method

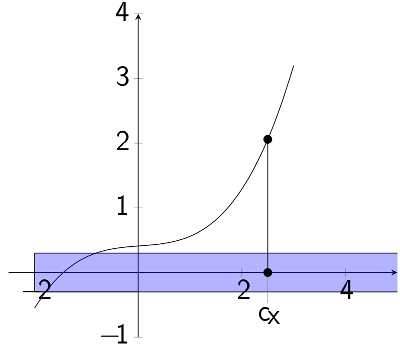
Interval extension of Newton's method:



$f(x) = 0.1 \cdot x^3 - 0.01 \cdot (x-3)^2 + 0.5$
$f'(x) = 0.2x^2 - 0.02x + 0.56$

Interval Newton method

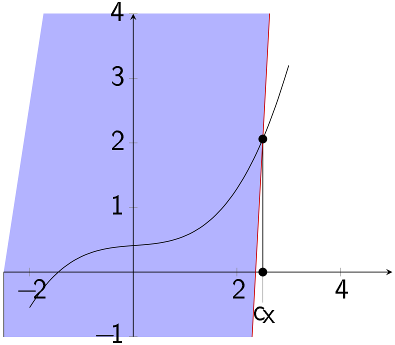
Interval extension of Newton's method:



$f(x) = 0.1 \cdot x^3 - 0.01 \cdot (x-3)^2 + 0.5$
$f'(x) = 0.2x^2 - 0.02x + 0.56$
Starting interval: $x \in I = [-2; 7]$, sampling point: $c_x = 2.5$

Interval Newton method

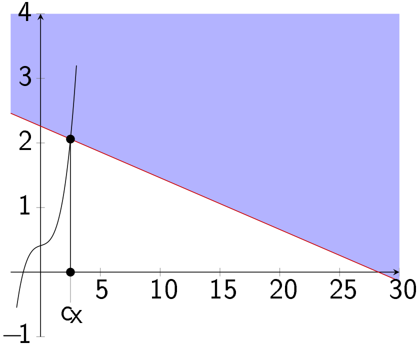
Interval extension of Newton's method:



$f(x) = 0.1 \cdot x^3 - 0.01 \cdot (x-3)^2 + 0.5$
$f'(x) = 0.2x^2 - 0.02x + 0.56$
Tangent: $[2.5] - \frac{f(c_x)}{f'(x)} = (-\infty; 2.36018] \cup [28.25; +\infty)$

Interval Newton method

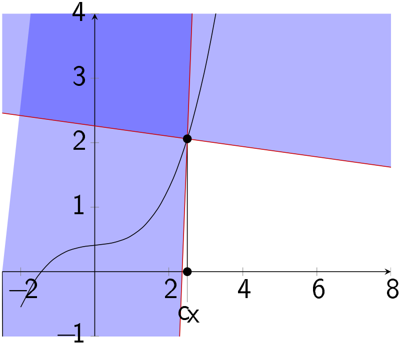
Interval extension of Newton's method:



$f(x) = 0.1 \cdot x^3 - 0.01 \cdot (x-3)^2 + 0.5$
$f'(x) = 0.2x^2 - 0.02x + 0.56$
Tangent: $[2.5] - \frac{f(c_x)}{f'(x)} = (-\infty; 2.36018] \cup [28.25; +\infty)$

Interval Newton method

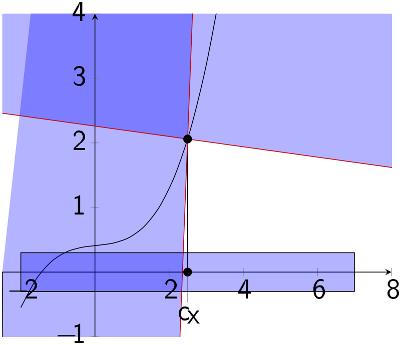
Interval extension of Newton's method:



$f(x) = 0.1 \cdot x^3 - 0.01 \cdot (x-3)^2 + 0.5$
$f'(x) = 0.2x^2 - 0.02x + 0.56$
Tangent: $[2.5] - \frac{f(c_x)}{f'(x)} = (-\infty; 2.36018] \cup [28.25; +\infty)$

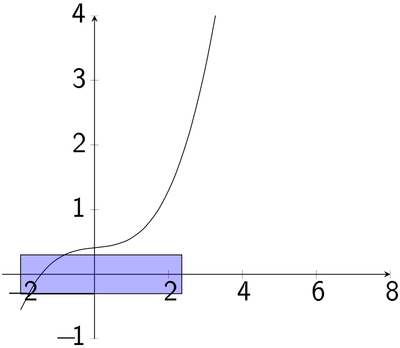
Interval Newton method

Interval extension of Newton's method:



$f(x) = 0.1 \cdot x^3 - 0.01 \cdot (x-3)^2 + 0.5$
$f'(x) = 0.2x^2 - 0.02x + 0.56$
New interval: $x \in [-2; 7] \cap (-\infty; 2.36018] \cup [28.25; +\infty) = [-2; 2.36018]$

Interval extension of Newton's method:



$f(x) = 0.1 \cdot x^3 - 0.01 \cdot (x-3)^2 + 0.5$
$f'(x) = 0.2x^2 - 0.02x + 0.56$
New interval: $x \in [-2;7] \cap (-\infty; 2.36018] \cup [28.25; +\infty) = [-2; 2.36018]$

Interval Newton method

Componentwise Newton operator

$$N_{cmp}(\overbrace{[x]}^{box}, \overbrace{f_i(x_1, \ldots, x_n), j}^{CC}) :=$$
$$c([x]_{x_j}) - \frac{f_i([x]_{x_1}, \ldots, [x]_{x_{j-1}}, c([x]_{x_j}), [x]_{x_{j+1}}, \ldots, [x]_{x_n})}{\frac{\partial f_i}{\partial x_j}([x]_{x_1}, \ldots, [x]_{x_n})}$$

Reminder(Multivariate Newton):

$$N_{cmp}(x, f_i(x_1, \ldots, x_n), j) = x - \frac{f_i(x)}{\frac{\partial f_i}{\partial x_j}(x)}$$

Interval Newton method

Componentwise Newton operator

$$N_{cmp}(\overbrace{[x]}^{box}, \overbrace{f_i(x_1, \ldots, x_n), j}^{CC}) :=$$

$$c([x]_{x_j}) - \frac{f_i([x]_{x_1}, \ldots, [x]_{x_{j-1}}, c([x]_{x_j}), [x]_{x_{j+1}}, \ldots, [x]_{x_n})}{\frac{\partial f_i}{\partial x_j}([x]_{x_1}, \ldots, [x]_{x_n})}$$

The operator N_{cmp} has two important properties:

- If x^* is a solution and $x^* \in [x]$, then $x^* \in N_{cmp}([x], i, j)$
- If $[x] \cap N_{cmp}([x], i, j) = \varnothing$, there is no solution in $[x]$

Interval Newton method

Componentwise Newton operator

$$N_{cmp}(\overbrace{[x]}^{box}, \overbrace{f_i(x_1, \ldots, x_n), j}^{CC}) :=$$

$$c([x]_{x_j}) - \frac{f_i([x]_{x_1}, \ldots, [x]_{x_{j-1}}, c([x]_{x_j}), [x]_{x_{j+1}}, \ldots, [x]_{x_n})}{\frac{\partial f_i}{\partial x_j}([x]_{x_1}, \ldots, [x]_{x_n})}$$

The operator N_{cmp} has two important properties:

- If x^* is a solution and $x^* \in [x]$, then $x^* \in N_{cmp}([x], i, j)$
- If $[x] \cap N_{cmp}([x], i, j) = \varnothing$, there is no solution in $[x]$

$\rightarrow$ Advantage: No diverging behavior like the original Newton method due to interval arithmetic.

$\rightarrow$ We can drop boxes when they contract to empty.

Introduce linear solver

ICP is not well-suited for linear problems
($\rightarrow$ slow convergence phenomenon).

Introduce linear solver

ICP is not well-suited for linear problems
($\rightarrow$ slow convergence phenomenon).

Make use of linear solvers for linear constraints:

- Pre-process to separate linear and nonlinear constraints

Introduce linear solver

ICP is not well-suited for linear problems
($\rightarrow$ slow convergence phenomenon).

Make use of linear solvers for linear constraints:

- Pre-process to separate linear and nonlinear constraints
- Determine linear feasible region

Introduce linear solver

ICP is not well-suited for linear problems
($\rightarrow$ slow convergence phenomenon).

Make use of linear solvers for linear constraints:

- Pre-process to separate linear and nonlinear constraints
- Determine linear feasible region
- Use nonlinear constraints for contraction

Introduce linear solver

ICP is not well-suited for linear problems
($\rightarrow$ slow convergence phenomenon).

Make use of linear solvers for linear constraints:

- Pre-process to separate linear and nonlinear constraints
- Determine linear feasible region
- Use nonlinear constraints for contraction
- Validate resulting boxes against linear feasible region

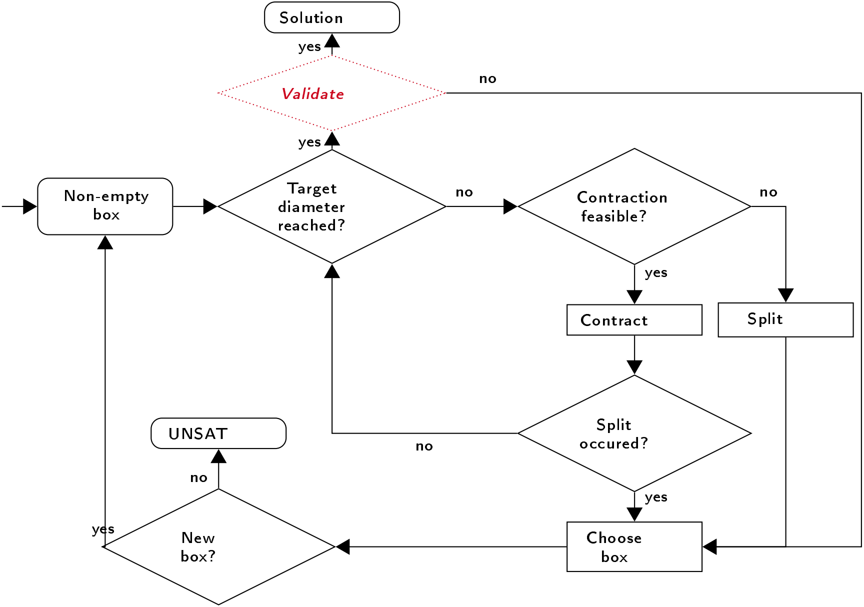
Introduce linear solver

ICP is not well-suited for linear problems
($\rightarrow$ slow convergence phenomenon).

Make use of linear solvers for linear constraints:

- Pre-process to separate linear and nonlinear constraints
- Determine linear feasible region
- Use nonlinear constraints for contraction
- Validate resulting boxes against linear feasible region
- In case box is linear infeasible: Add violated linear constraint for contraction

Algorithm overview



Validation

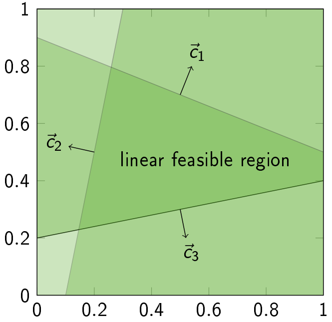
- Resulting intervals represent n–dimensional hyperboxes

Validation

- Resulting intervals represent n–dimensional hyperboxes
- Boxes may violate linear constraints

Validation

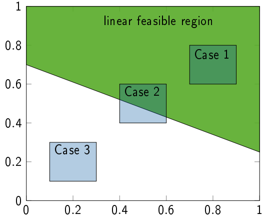
- Resulting intervals represent n–dimensional hyperboxes
- Boxes may violate linear constraints
- Check resulting boxes against the linear feasible region



Case distinction

We have to distinct 3 cases:

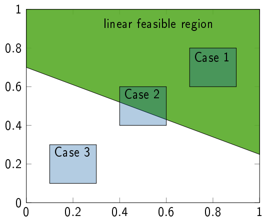
- Case 1: The resulting interval lies completely inside the linear feasible region



Case distinction

We have to distinct 3 cases:

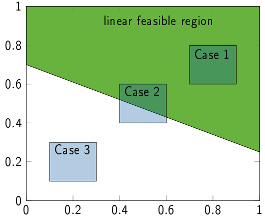
- Case 1: The resulting interval lies completely inside the linear feasible region
- Case 2: The hyperbox partly covers the linear feasible region



Case distinction

We have to distinct 3 cases:

- Case 1: The resulting interval lies completely inside the linear feasible region
- Case 2: The hyperbox partly covers the linear feasible region
- Case 3: The solution interval lies outside the linear feasible region

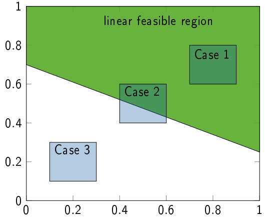


Case distinction

We have to distinct 3 cases:

- Case 1: The resulting interval lies completely inside the linear feasible region
- Case 2: The hyperbox partly covers the linear feasible region
- Case 3: The solution interval lies outside the linear feasible region

The problem lies in separating the 1st case from the 2nd one.

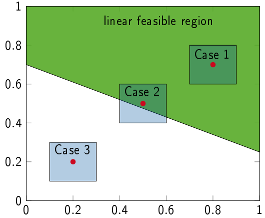


Case distinction

We have to distinct 3 cases:

- Case 1: The resulting interval lies completely inside the linear feasible region
- Case 2: The hyperbox partly covers the linear feasible region
- Case 3: The solution interval lies outside the linear feasible region

The problem lies in separating the 1st case from the 2nd one.

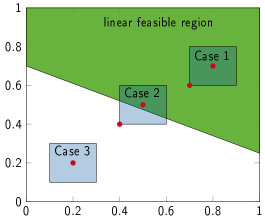


Case distinction

We have to distinct 3 cases:

- Case 1: The resulting interval lies completely inside the linear feasible region
- Case 2: The hyperbox partly covers the linear feasible region
- Case 3: The solution interval lies outside the linear feasible region

The problem lies in separating the 1st case from the 2nd one.

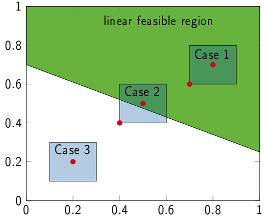


Case distinction

We have to distinct 3 cases:

- Case 1: The resulting interval lies completely inside the linear feasible region
- Case 2: The hyperbox partly covers the linear feasible region
- Case 3: The solution interval lies outside the linear feasible region

The problem lies in separating the 1st case from the 2nd one.



If a checked point is valid $\rightarrow$ return SAT

Raise splitting

Instead of having an internal box management (e.g. via a tree) we can raise a split to the SAT solver

Raise splitting

Instead of having an internal box management (e.g. via a tree) we can raise a split to the SAT solver

- Create deductions (tautologies)

Raise splitting

Instead of having an internal box management (e.g. via a tree) we can raise a split to the SAT solver

- Create deductions (tautologies)
- Deduction is added as additional information to the SAT solver

Raise splitting

Instead of having an internal box management (e.g. via a tree) we can raise a split to the SAT solver

- Create deductions (tautologies)
- Deduction is added as additional information to the SAT solver
- SAT solver decides based on its information which split to take

Raise splitting

Instead of having an internal box management (e.g. via a tree) we can raise a split to the SAT solver

- Create deductions (tautologies)
- Deduction is added as additional information to the SAT solver
- SAT solver decides based on its information which split to take

ICP deductions include:

Raise splitting

Instead of having an internal box management (e.g. via a tree) we can raise a split to the SAT solver

- Create deductions (tautologies)
- Deduction is added as additional information to the SAT solver
- SAT solver decides based on its information which split to take

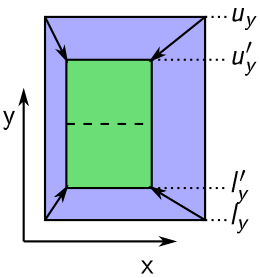
ICP deductions include:

- The split
- (optional) A premise in form of previous contractions

Raise splitting

Setup:

- Original box (blue) contracted to new box (green)
- Contraction via constraints $c_i, i = 1, \ldots, n$
- Demanded split at $c(l_y)$

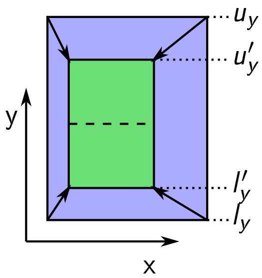


We create a splitting deduction:

Raise splitting

Setup:

- Original box (blue) contracted to new box (green)
- Contraction via constraints $c_i, i = 1, \ldots, n$
- Demanded split at $c(l_y)$



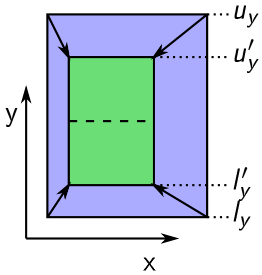
We create a splitting deduction:

$$\underbrace{(y \in [l'_y; c(l_y)] \oplus y \in (c(l_y); u'_y])}_{split}$$

Raise splitting

Setup:

- Original box (blue) contracted to new box (green)
- Contraction via constraints $c_i, i = 1, \ldots, n$
- Demanded split at $c(l_y)$



We create a splitting deduction:

$$\underbrace{x \in [I_x] \wedge y \in [I_y] \wedge (c_1 \wedge \ldots \wedge c_n)}_{premise} \rightarrow \underbrace{(y \in [l'_y; c(l_y)] \oplus y \in (c(l_y); u'_y])}_{split}$$

Summary

We've learned about:

- Basic interval arithmetic (no division by intervals containing 0)
- Interval propagation
- Problems during contraction
- Splitting
- Basic algorithm (see Slide 20)
- Possible extensions (not in detail)

Bibliography I