Numerical Constraint Solving Based on Linear Relaxations

Stefan Ratschan

Institute of Computer Science
Czech Academy of Sciences

April 3, 2011

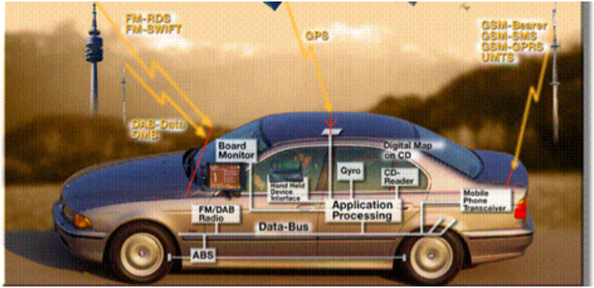
Disclaimer

Talk more of survey type

Hardly any original results

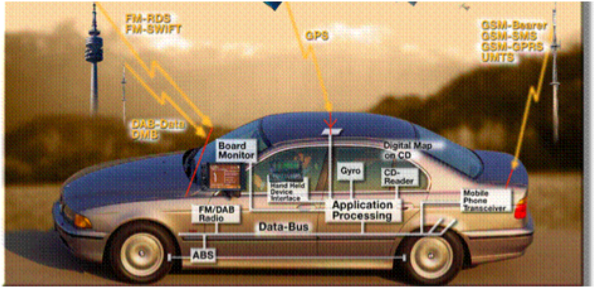
Numerical Constraints: Motivation I

By far most micro-processors nowadays do not occur in desktop PC's but embedded in technical systems (trains, cars, robots, your washing machine etc.)



Numerical Constraints: Motivation I

By far most micro-processors nowadays do not occur in desktop PC's but embedded in technical systems (trains, cars, robots, your washing machine etc.)



Models of technical systems usually in numerical domains.

Numerical Constraints: Motivation II

Continuous is simpler then discrete!

Numerical Constraints: Motivation II

Continuous is simpler then discrete!

	integers	reals
sat. of linear constraints	NP-hard	polynomial time
sat. of polynomial constraints	undecidable	decidable

Numerical Constraints: Motivation II

Continuous is simpler then discrete!

	integers	reals
sat. of linear constraints	NP-hard	polynomial time
sat. of polynomial constraints	undecidable	decidable

So: to solve discrete problem,

exploit corresponding continuous problem ("relaxation").

Numerical Constraints: Motivation II

Continuous is simpler then discrete!

	integers	reals
sat. of linear constraints	NP-hard	polynomial time
sat. of polynomial constraints	undecidable	decidable

So: to solve discrete problem,
exploit corresponding continuous problem ("relaxation").

Prototypical example: MILP

Numerical Constraints: Motivation III

Sometimes continuous reasoning can help
in analyzing discrete systems.

Numerical Constraints: Motivation III

Sometimes continuous reasoning can help
in analyzing discrete systems.

For example:

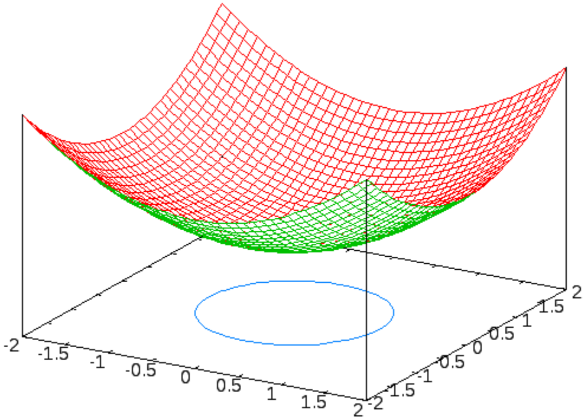
Verification of programs with integer variables based on
invariants and ranking functions with rational/real coefficients

Example

$$x^2 + y^2 - 1 = 0 \wedge y - x^2 = 0$$

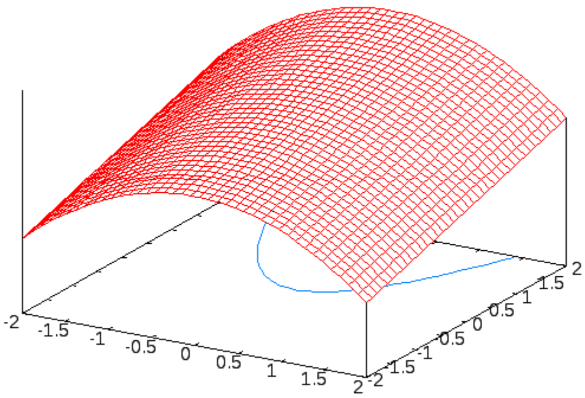
Example

$$x^2 + y^2 - 1 = 0 \wedge y - x^2 = 0$$



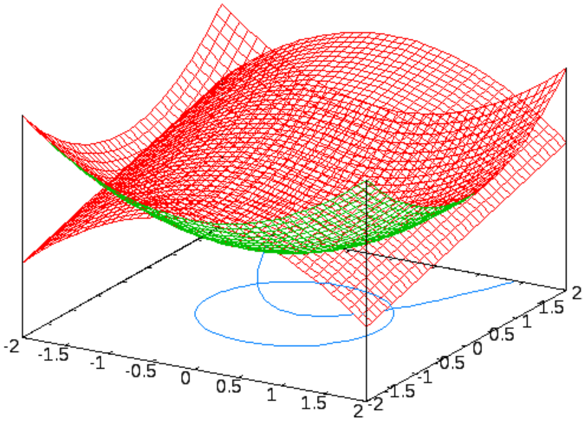
Example

$$x^2 + y^2 - 1 = 0 \wedge y - x^2 = 0$$

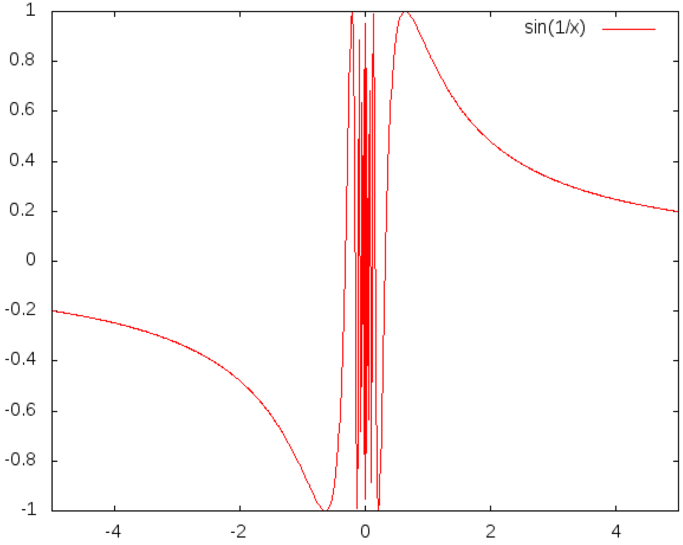


Example

$$x^2 + y^2 - 1 = 0 \wedge y - x^2 = 0$$



Example



"Solving Numerical Constraints"

Given: conjunction of equalities and inequalities,
function symbols in $\{+, \times, \sin, \exp, \ldots\}$ (*constraint*)

"Solving Numerical Constraints"

Given: conjunction of equalities and inequalities,
function symbols in $\{+, \times, \mathsf{sin}, \mathsf{exp}, \dots\}$ (*constraint*)

Find: "Nice" over-approximation of set of real solutions

"Solving Numerical Constraints"

Given: conjunction of equalities and inequalities,
function symbols in $\{+, \times, \sin, \exp, \dots\}$ (*constraint*)

Find: "Nice" over-approximation of set of real solutions

Note: Since sin can encode the integers,
we are in the land of the undecidable.

"Solving Numerical Constraints"

Given: conjunction of equalities and inequalities,
function symbols in $\{+, \times, \sin, \exp, \dots\}$ (*constraint*)

Find: "Nice" over-approximation of set of real solutions

Note: Since sin can encode the integers,
we are in the land of the undecidable.

But: We head for *quasi-decidability*:
algorithm that can detect (un)satisfiability for all *robust* inputs
(does not change (un)satisfiability under perturbations)
[Franek et al., 2010, Ratschan, 2010]

"Solving Numerical Constraints"

Given: conjunction of equalities and inequalities,
function symbols in $\{+, \times, \sin, \exp, \dots\}$ (*constraint*)

Find: "Nice" over-approximation of set of real solutions

Note: Since sin can encode the integers,
we are in the land of the undecidable.

But: We head for *quasi-decidability*:
algorithm that can detect (un)satisfiability for all *robust* inputs
(does not change (un)satisfiability under perturbations)
[Franek et al., 2010, Ratschan, 2010]

From now on, we assume that we search for solutions
in an n-dimensional hyper-rectangle (*box*).

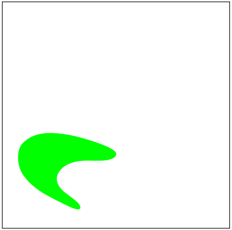
Basic Operation: Pruning

For

- a constraint ϕ in n variables
- an n-dimensional box B.

prune(ϕ, B) is a box B', such that

- $B' \subseteq B$,
- B' contains all solutions of ϕ in B:



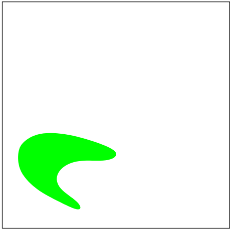
Basic Operation: Pruning

For

- a constraint ϕ in n variables
- an n-dimensional box B.

prune(ϕ, B) is a box B', such that

- $B' \subseteq B$,
- B' contains all solutions of ϕ in B:



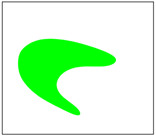
Basic Operation: Pruning

For

- a constraint ϕ in n variables
- an n-dimensional box B.

prune(ϕ, B) is a box B', such that

- $B' \subseteq B$,
- B' contains all solutions of ϕ in B:



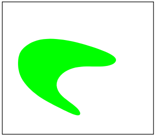
Basic Operation: Pruning

For

- a constraint ϕ in n variables
- an n-dimensional box B.

prune(ϕ, B) is a box B', such that

- $B' \subseteq B$,
- B' contains all solutions of ϕ in B:



This already is an over-approximation of the solution set of ϕ in B.

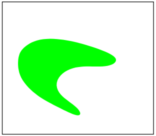
Basic Operation: Pruning

For

- a constraint ϕ in n variables
- an n-dimensional box B.

prune(ϕ, B) is a box B', such that

- $B' \subseteq B$,
- B' contains all solutions of ϕ in B:



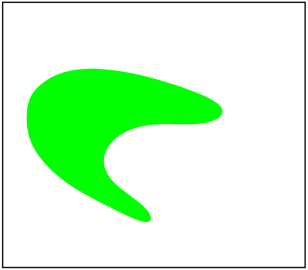
This already is an over-approximation of the solution set of ϕ in B.

But: Usually (by design) efficient, but crude.

Branch and Prune Algorithm

Algorithm $BP(\phi, B)$:

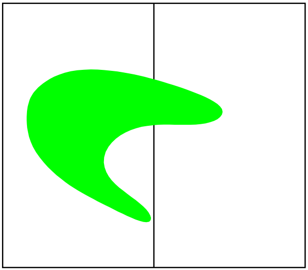
$S \leftarrow prune(\phi, B)$
if S good enough **then** S
else
  let B_1, B_2 be s. t. $S = B_1 \cup B_2$,
    non-overlapping
  return $BP(\phi, B_1) \cup BP(\phi, B_2)$



Branch and Prune Algorithm

Algorithm $BP(\phi, B)$:

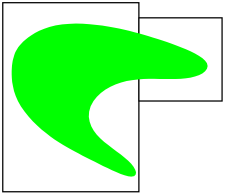
$S \leftarrow prune(\phi, B)$
if S good enough **then** S
else
  let B_1, B_2 be s. t. $S = B_1 \cup B_2$, non-overlapping
  return $BP(\phi, B_1) \cup BP(\phi, B_2)$



Branch and Prune Algorithm

Algorithm $BP(\phi, B)$:

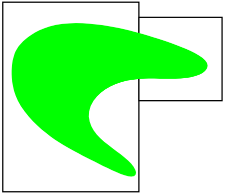
$S \leftarrow prune(\phi, B)$
if S good enough **then** S
else
  let B_1, B_2 be s. t. $S = B_1 \cup B_2$,
  non-overlapping
  return $BP(\phi, B_1) \cup BP(\phi, B_2)$



Branch and Prune Algorithm

Algorithm $BP(\phi, B)$:

$S \leftarrow \mathit{prune}(\phi, B)$
if S good enough **then** S
else
  let B_1, B_2 be s. t. $S = B_1 \cup B_2$,
  non-overlapping
  return $BP(\phi, B_1) \cup BP(\phi, B_2)$



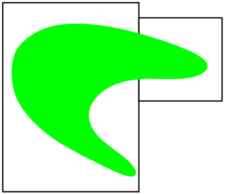
"good enough" can be:

- $= \emptyset$: try to prove unsatisfiability at all costs,
  algorithm may run forever, but terminates for robust inputs

Branch and Prune Algorithm

Algorithm $BP(\phi, B)$:

$S \leftarrow prune(\phi, B)$
if S good enough **then** S
else
  let B_1, B_2 be s. t. $S = B_1 \cup B_2$,
  non-overlapping
  return $BP(\phi, B_1) \cup BP(\phi, B_2)$



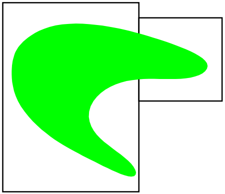
"good enough" can be:

- $= \emptyset$: try to prove unsatisfiability at all costs,
  algorithm may run forever, but terminates for robust inputs
- box small enough (size: volume, maximal side-length)

Branch and Prune Algorithm

Algorithm $BP(\phi, B)$:

$S \leftarrow prune(\phi, B)$
if S good enough **then** S
else
  let B_1, B_2 be s. t. $S = B_1 \cup B_2$,
  non-overlapping
  return $BP(\phi, B_1) \cup BP(\phi, B_2)$



"good enough" can be:

- $= \emptyset$: try to prove unsatisfiability at all costs, algorithm may run forever, but terminates for robust inputs
- box small enough (size: volume, maximal side-length)
- time bound exceeded

Usage

If over-approximation is empty,
    we know that input is unsatisfiable, otherwise

Usage

If over-approximation is empty,
we know that input is unsatisfiable, otherwise

it can be used for searching for

- real solutions
- integer solution.

Usage

If over-approximation is empty,
we know that input is unsatisfiable, otherwise

it can be used for searching for

- real solutions
- integer solution.

This search can even be built into the branch-and-prune algorithm.

Pruning Based on Interval Arithmetic

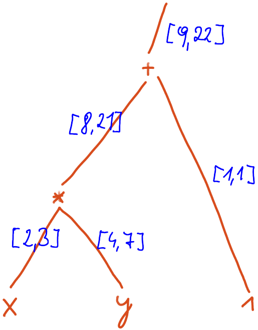
Special case: one single equality

Pruning Based on Interval Arithmetic

Special case: one single equality

Input: $f = 0$, box B
Example:

$xy + 1 = 0, x \in [2, 3], y \in [4, 7]$

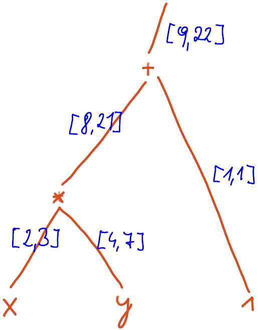


Pruning Based on Interval Arithmetic

Special case: one single equality

Input: $f = 0$, box B
Example:

$xy + 1 = 0, x \in [2,3], y \in [4,7]$



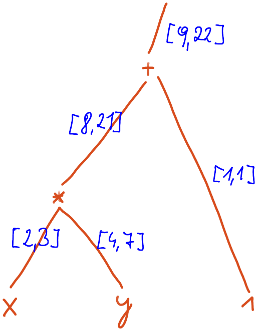
Compute interval $[f](B)$ such that $\{f(\vec{x}) \mid \vec{x} \in B\} \subseteq [f](B)$

Pruning Based on Interval Arithmetic

Special case: one single equality

Input: $f = 0$, box B
Example:

$xy + 1 = 0, x \in [2, 3], y \in [4, 7]$



Compute interval $[f](B)$ such that $\{f(\vec{x}) \mid \vec{x} \in B\} \subseteq [f](B)$

if $0 \notin [f](I_1, \ldots, I_n)$ **then** $\emptyset$ **else** B

Comparison with Symbolic Computation

- Interval based methods:
  - Usually require a-priori bounds
  - Often do not exploit (partial) linearity well

Comparison with Symbolic Computation

- Interval based methods:
  - Usually require a-priori bounds
  - Often do not exploit (partial) linearity well
- Symbolic computation:
  - Mostly polynomial case only
  - Usually does not produce useful partial results under limited time (no anytime algorithms)

Comparison with Symbolic Computation

- Interval based methods:
  - Usually require a-priori bounds
  - Often do not exploit (partial) linearity well
- Symbolic computation:
  - Mostly polynomial case only
  - Usually does not produce useful partial results under limited time (no anytime algorithms)

Both: limited scalability

Comparison with Symbolic Computation

- Interval based methods:
  - Usually require a-priori bounds
  - Often do not exploit (partial) linearity well
- Symbolic computation:
  - Mostly polynomial case only
  - Usually does not produce useful partial results under limited time (no anytime algorithms)

Both: limited scalability

Now: Extension of interval approach [Lebbah et al., 2005], applying ideas from global optimization

Comparison with Symbolic Computation

- Interval based methods:
  - Usually require a-priori bounds
  - Often do not exploit (partial) linearity well
- Symbolic computation:
  - Mostly polynomial case only
  - Usually does not produce useful partial results under limited time (no anytime algorithms)

Both: limited scalability

Now: Extension of interval approach [Lebbah et al., 2005], applying ideas from global optimization

- more often can live without a-priori bounds
- efficient handling of linearity
- partial results under limited time
- more scalable

Primitive Constraint Decomposition

$$\sin xy + z = 1 \land x - y = 7$$

Primitive Constraint Decomposition

$$\sin xy + z = 1 \wedge x - y = 7$$

$$xy = t_1 \wedge \sin t_1 = t_2 \wedge t_2 + z = t_3 \wedge t_3 = 1 \wedge x - y = t_4 \wedge t_4 = 7$$

Primitive Constraint Decomposition

$$\sin xy + z = 1 \wedge x - y = 7$$

$$xy = t_1 \wedge \sin t_1 = t_2 \wedge t_2 + z = t_3 \wedge t_3 = 1 \wedge x - y = t_4 \wedge t_4 = 7$$

Now: for each primitive constraint,
produce implied linear inequalities (*linear relaxation*)

Primitive Constraint Decomposition

$$\sin xy + z = 1 \land x - y = 7$$

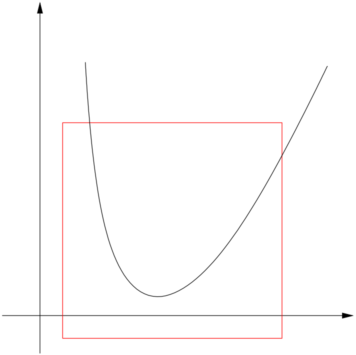
$$xy = t_1 \land \sin t_1 = t_2 \land t_2 + z = t_3 \land t_3 = 1 \land x - y = t_4 \land t_4 = 7$$

Now: for each primitive constraint,
produce implied linear inequalities (*linear relaxation*)

Result: linear program whose
solution set over-approximates original solution set

Linear Relaxations

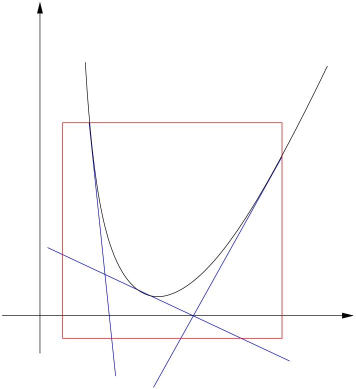
$z = x + y$: already linear



$z = f(x)$, where f convex:

Linear Relaxations

$z = x + y$: already linear

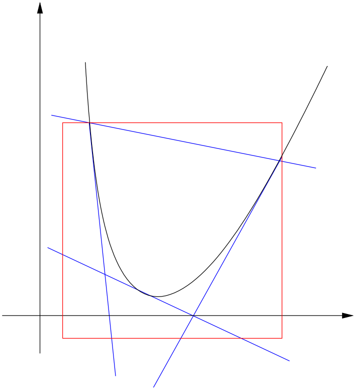


$z = f(x)$, where f convex:

- underestimate: tangent at any point,

Linear Relaxations

$z = x + y$: already linear

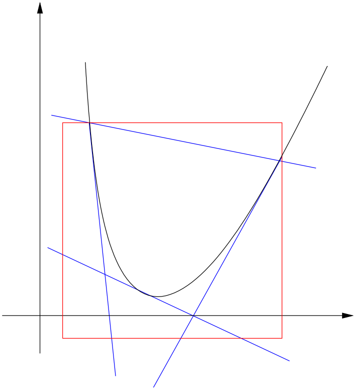


$z = f(x)$, where f convex:

- underestimate: tangent at any point,
- overestimate: secant at endpoints

Linear Relaxations

$z = x + y$: already linear



$z = f(x)$, where f convex:

- underestimate: tangent at any point,
- overestimate: secant at endpoints

if not convex: treat convex/concave segments separately

Linear Relaxation of Multiplication

Theorem: [McCormick, 1976]

$z = xy, x \in [\underline{x}, \overline{x}], y \in [\underline{y}, \overline{y}]$ implies

- $\underline{y}x + \underline{x}y - \underline{x}\underline{y} \leq z$
- $z \leq \underline{y}x + \overline{x}y - \overline{x}\underline{y}$
- $\overline{y}x + \overline{x}y - \overline{xy} \leq z$
- $z \leq \overline{y}x + \underline{x}y - \underline{x}\overline{y}$

Linear Relaxation of Multiplication

Theorem: [McCormick, 1976]
$z = xy$, $x \in [\underline{x}, \overline{x}], y \in [\underline{y}, \overline{y}]$ implies

- $\underline{y}x + \underline{x}y - \underline{x}\underline{y} \leq z$
- $z \leq \underline{y}x + \overline{x}y - \overline{x}\underline{y}$
- $\overline{y}x + \overline{x}y - \overline{xy} \leq z$
- $z \leq \overline{y}x + \underline{x}y - \underline{x}\overline{y}$

Moreover:

- optimal (in general, no further implied inequalities) [Al-Khayyal and Falk, 1983]

Linear Relaxation of Multiplication

Theorem: [McCormick, 1976]
$z = xy, x \in [\underline{x}, \overline{x}], y \in [\underline{y}, \overline{y}]$ implies

- $\underline{y}x + \underline{x}y - \underline{x}\underline{y} \le z$
- $z \le \underline{y}x + \overline{x}y - \overline{x}\underline{y}$
- $\overline{y}x + \overline{x}y - \overline{x}\overline{y} \le z$
- $z \le \overline{y}x + \underline{x}y - \underline{x}\overline{y}$

Moreover:

- optimal (in general, no further implied inequalities) [Al-Khayyal and Falk, 1983]
- always at least as tight as interval arithmetic

In practice

Attention! careless rounding might cut of solutions!

In practice

Attention! careless rounding might cut of solutions!

Now: Over-approximating LP can be used
for pruning within branch-and-prune algorithm

In practice

Attention! careless rounding might cut of solutions!

Now: Over-approximating LP can be used
for pruning within branch-and-prune algorithm

Here one can
just test whether the resulting LP is satisfiable,

In practice

Attention! careless rounding might cut of solutions!

Now: Over-approximating LP can be used
for pruning within branch-and-prune algorithm

Here one can
just test whether the resulting LP is satisfiable,

or

try to infer new variable bounds from it by solving $2n$ LPs

Implementation

We have a (very prototypical) implementation: RSolver

Implementation

We have a (very prototypical) implementation: RSolver

`http://rsolver.sourceforge.net`

Implementation

We have a (very prototypical) implementation: RSOLVER

http://rsolver.sourceforge.net

Can handle quantifiers [Ratschan, 2006]
as long as this does not need
positive information about equalities

For example, it cannot (yet) prove

$$\exists x \,.\, f(x) = 0$$

Implementation

We have a (very prototypical) implementation: RSolver

`http://rsolver.sourceforge.net`

Can handle quantifiers [Ratschan, 2006]
as long as this does not need
positive information about equalities

For example, it cannot (yet) prove

$$\exists x \, . \, f(x) = 0$$

But, in theory [Franek et al., 2010] we can already do this, too.

Conclusion

Constraint solvers for the real numbers,
can be useful for analyzing discrete problem.

Conclusion

Constraint solvers for the real numbers,
can be useful for analyzing discrete problem.

If you want to try, contact us.

Literature I

Faiz A. Al-Khayyal and James E. Falk. Jointly constrained biconvex programming. *Mathematics of Operations Research*, 8 (2):273–286, 1983.

Peter Franek, Stefan Ratschan, and Piotr Zgliczynski. Satisfiability of systems of equations of real analytic functions is quasi-decidable. `http://www.cs.cas.cz/~ratschan/preprints.html`, 2010.

Yahia Lebbah, Claude Michel, Michel Rueher, David Daney, and Jean-Pierre Merlet. Efficient and safe global constraints for handling numerical constraint systems. *SIAM Journal on Numerical Analysis*, 42(5):2076–2097, 2005.

Garth P. McCormick. Computability of global solutions to factorable nonconvex programs: Part I convex underestimating problems. *Mathematical Programming*, 10(1):147–175, 1976.

Literature II

Stefan Ratschan. Efficient solving of quantified inequality constraints over the real numbers. *ACM Transactions on Computational Logic*, 7(4):723–748, 2006.

Stefan Ratschan. Safety verification of non-linear hybrid systems is quasi-decidable. `http://www2.cs.cas.cz/~ratschan/papers/quasidecidable.pdf`, 2010. Extended journal version, to be submitted.