The fundamental limitation: undecidability

- In 1921, David Hilbert put forward the so-called Hilbert’s Program, calling for a formalization of all of mathematics in axiomatic form, together with a proof that this axiomatization of mathematics is consistent, to be carried out using only “finitary” methods.
- Kurt Gödel’s incompleteness theorems [1] essentially show that Hilbert’s Program cannot be carried out.

Reference

Decision Problems

- A **decision problem** is a computational problem where the answer is always YES/NO:
  
  \[ \text{solve\_problem(data)} \mapsto \{\text{YES, NO}\} \]

- The **complement** \(\neg P\) of a decision problem \(P\) is one where all the YES and NO answers are exchanged.

Decidability/Semidecidable/Undecidability

A decision problem is:

- **Decidable** if and only if there exists an **algorithm** to solve the problem in finite time;
- **Undecidable** if and only if there exists no **algorithm** to solve the problem in finite time;
- **Semidecidable** if and only if there exists an **algorithm** to solve the problem in finite time when the answer is YES but which **may not terminate** when the answer is NO;

The termination problem

- **Termination problem**: given a sequential program and its input data, will execution of this program on these data ever terminate?
- The complement is the **Nontermination problem**: given a sequential program and its input data, will execution of this program on these data never terminate?
- Termination is undecidable (but semidecidable);
- Nontermination is undecidable (and not semidecidable).

Interpretor

- We let \(Ff\) be the text file (of type text) containing the text of a program encoding a function \(f\) of type text \(\rightarrow\) bool;
- We let \(Fd\) be the text file (of type text) containing the text encoding the data \(d\)
- An interpreter \(I : \text{text} \times \text{text} \rightarrow \text{bool}\) is a program which execution \(I(Ff, Fd)\) is the result \(f(d)\) of the evaluation of function \(f\) on the data \(d\).
Termination is semidecidable

Termination\((F_f, F_d) = \text{if } I(\text{F}_f, \text{F}_d) \text{ then YES else YES}\)
- Will answer YES if and only if \(I(\text{F}_f, \text{F}_d)\) that is \(f(d)\) does terminate
- Will not terminate if and only if \(I(\text{F}_f, \text{F}_d)\) that is \(f(d)\) does not terminate

Termination is undecidable


Reference


Proof that termination is undecidable

It can be schematized as follows:
- Assume, by reductio ad absurdum, that we can design a termination algorithm \(T : \text{text} \times \text{text} \rightarrow \text{bool}\), which execution is assumed to always terminate, and returns \(T(\text{F}_f, \text{F}_d) = \text{YES}\) if and only if \(I(\text{F}_f, \text{F}_d)\) terminates and \(T(\text{F}_f, \text{F}_d) = \text{NO}\) otherwise.
Let \( F_c \) be the text of a function \( c : \text{text} \to \text{bool} \) defined by:

\[
c(F) = \text{if } T(F, F) \text{ then } \neg I(F, F) \text{ else YES}
\]

Observe that execution of \( c \) always terminate, whence

\[
T(F_c, F_c) = \text{true}
\]

It follows that:

\[
I(F_c, F_c) = \text{if } T(F_c, F_c) \text{ then } \neg I(F_c, F_c) \text{ else YES}
\]

is a contradiction.

Nontermination is not semidecidable

a) \( \text{Decidable}(P) \Leftrightarrow [\text{Semidecidable}(P) \land \text{Semidecidable}(\neg P)] \)

\( \Rightarrow \) obvious (by defining \( \text{Semidecision}(P) \equiv \text{Decision}(P) \)
and \( \text{Semidecision}(\neg P) \equiv \neg \text{Decision}(P) \));

\( \Leftarrow \) alternatively execute one step of the semidecision algorithms of \( P \) and \( \neg P \). Stop as soon as the first answer is returned.

b) \( \neg \text{Semidecidable}(\neg \text{Terminaison}) \)

By reductio ad absurdum,

Semidecidable(Terminaison) and Semidecidable(\neg Terminaison) would imply Decidable(Terminaison).

Problem Reduction

- To prove that a problem \( P \) is undecidable, prove, by reductio ad absurdum, that if it were decidable, then the termination problem would be decidable.

Constant propagation is undecidable

- The constant propagation problem: determines whether, after initialization, a variable is constant (is never assigned a different value);

- The program \( P \) does not terminate if and only if the variable \( X \) has a constant value after initialization in the program:

\[
\text{var } X : \text{boolean}; (* \text{new variable not in } P *)
\]

\[
X := \text{true};
\]

\[
P;
\]

\[
X := \text{false};
\]
Absence of runtime errors is undecidable

The absence of runtime errors is not semidecidable:
– If absence of runtime errors in a program $P$ where semidecidable then the nontermination of $P$ would be semidecidable, by answering the question to know if $P; 1/0$ has no runtime error.

What to do about undecidable problems?

Beyond simply abandoning:
– Consider decidable subcases only (but computational complexity strikes!)
– Ask for correct, intelligent, interactive human help
– Accept nontermination
– Accept approximations (I don’t know)

Example of approximation: false alarms

When verifying the absence of runtime errors in a program, it may be the case that the automatic verifier is enabled to establish statically that some error can be raised at runtime although this will never happen during execution.

For soundness, it must report a possibility of runtime error, which is impossible. This is called a false alarm.

Example of false alarms in ASTRÉE

```bash
% cat -n falsealarm.c
1 /* falsealarm.c */
2 void main()
3 {
4   int x, y;
5   if ((-4681 < y) && (y < 4681) && (x < 32767) && (-32767 < x)
6      && ((7*y*y - 1) == x*x)) {
7     y = 1 / x;
8   }
%
% astree -exec-fn main falsealarm.c | grep WARN
falsealarm.c:6:9-6:14:[call#main@2]: WARN: integer division by zero [-32766, 32766]
%
```
Computational Complexity

- Decidable problems can have a very high computational complexity;
- The time complexity of a problem is the number of steps that it takes to solve an instance of the problem, as a function of the size of the input, (usually measured in bits) using the most efficient algorithm;
- For example sorting an $n$-elements array is $\mathcal{O}(n \log n)$;

---

Complexity Classes

- the class $P$ consists of all those decision problems that can be solved in polynomial time in the size of the input on a deterministic sequential machine;
- the class $NP$ consists of all those decision problems whose positive solutions can be verified in polynomial time given the right information\(^1\);
- the class co-$NP$ consists of all those decision problems whose complement is in $P$.

---

Problem Reduction

- A problem decision $A$ is reducible \(^5\) to a decision problem $B$

\[ A \leq B \]

- there exists a deterministic polynomial-time algorithm which transforms instances $a$ of $A$ into instances $b$ of $B$, such that the answer to $b$ is YES if and only if the answer to $a$ is YES.

---

Reference

\(^1\) Recall the big O notation: if $f(x)$ and $g(x)$ are two real functions then $f(x) = \mathcal{O}(g(x))$ as $x \to +\infty$ if and only if there exist numbers $x_0$ and $M$ such that $|f(x)| \leq M|g(x)|$ for $x > x_0$.

NP-hardness

A decision problem is **NP-hard** if and only if
– every other problem in NP is reducible to it.

If we can find a polynomial algorithm to solve a NP-hard problem, then $P = NP$ (?).

NP-completeness

A decision problem is **NP-complete** [6] if
– it is in NP, and
– every other problem in NP is reducible to it.

Reference

The boolean satisfiability problem (SAT)

– An instance of SAT is defined by a Boolean expression written using only AND, OR, NOT, variables, and parentheses.
– The question is: given the expression, is there some assignment of TRUE and FALSE values to the variables that will make the entire expression true?
– For $n$ variables, there are $2^n$ possible truth assignments to be checked.

Complexity of the boolean satisfiability problem

– The boolean satisfiability problem is NP-complete [7]

Reference
What to do about NP-complete problems?

- **Small is beautiful**: Consider only problems of very small size.
- **Special cases**: An algorithm that is provably fast if the problem instances belong to a certain special case. Fixed-parameter algorithms can be seen as an implementation of this approach.
- **Probabilistic**: An algorithm that provably yields good average runtime behavior for a given distribution of the problem instances—ideally, one that assigns low probability to "hard" inputs.

**SAT solvers**

- modern variants of the Davis-Logemann-Loveland algorithm [8] (depth first search with backtracking), such as zchaff\(^2\) [9];
- stochastic local search algorithms, e.g. WalkSAT [10].

**Polynomial Time Complexity**

- **Heuristic**: An algorithm that works "reasonably well" on many cases, but for which there is no proof that it is always fast (a rule of thumb, intuition).
- **Approximation**: An algorithm that quickly finds a suboptimal solution that is within a certain (known) range of the optimal one.

---

**SAT solvers**

- modern variants of the Davis-Logemann-Loveland algorithm [8] (depth first search with backtracking), such as zchaff\(^2\) [9];
- stochastic local search algorithms, e.g. WalkSAT [10].

**Reference**


\(^2\) A few thousands variables, reported to solve a problem with a million variables and 10 million clauses.

---

**Polynomial Time Complexity**

- Polynomial-time computability is identified with the intuitive notion of algorithmic efficiency;
- Intuitively valid only for small powers:

<table>
<thead>
<tr>
<th>(n)</th>
<th>(O(n))</th>
<th>(O(n \log(n)))</th>
<th>(O(n^2))</th>
<th>(O(n^3))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(\epsilon)</td>
<td>(\epsilon)</td>
<td>(\epsilon)</td>
<td>(\epsilon)</td>
</tr>
<tr>
<td>10</td>
<td>(\epsilon)</td>
<td>(\epsilon)</td>
<td>0.1(\mu)s</td>
<td>1(\mu)s</td>
</tr>
<tr>
<td>(10^3)</td>
<td>1(\mu)s</td>
<td>6(\mu)s</td>
<td>1ms</td>
<td>1s</td>
</tr>
<tr>
<td>(10^6)</td>
<td>1ms</td>
<td>13ms</td>
<td>16(\mu)n</td>
<td>32 years</td>
</tr>
<tr>
<td>(10^9)</td>
<td>1s</td>
<td>20s</td>
<td>32 years</td>
<td>300 000 000 centuries</td>
</tr>
<tr>
<td>(10^{12})</td>
<td>16(\mu)n</td>
<td>7.7h</td>
<td>300 000 centuries</td>
<td>—</td>
</tr>
<tr>
<td>(10^{15})</td>
<td>11.6 days</td>
<td>1 year</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>
Large problems

A computer program with
- 10,000 global variables
- represented on 32 bits
- and 500,000 lines of code
has about $2.10^{19} = 2^{64}$ states!

The following slides are based on:

Patrick Cousot.
Proving Program Invariance and Termination by Parametric Abstraction, Lagrangian Relaxation and Semidefinite Programming (Invited paper).
In *Sixth International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI’05)*.
Overview of the Termination Analysis Method

Proving Termination of a Loop

1. Perform an iterated forward/backward relational static analysis of the loop with termination hypothesis to determine a necessary proper termination precondition
2. Assuming the termination precondition, perform a forward relational static analysis of the loop to determine the loop invariant
3. Assuming the loop invariant, perform a forward relational static analysis of the loop body to determine the loop abstract operational semantics
4. Assuming the loop semantics, use an abstraction of Floyd’s ranking function method to prove termination of the loop

The main point in this talk is (4).

Proving Termination of a Loop

1. Termination precondition
2. Loop invariant
3. Loop operational semantics
4. Ranking function

Arithmetic Mean Example

while (x <> y) do
    x := x - 1;
    y := y + 1
od

The polyhedral abstraction used for the static analysis of the examples is implemented using Bertrand Jeannet’s NewPolka library.
Arithmetic Mean Example

1. Perform an iterated forward/backward relational static analysis of the loop with termination hypothesis to determine a necessary proper termination precondition
2. Assuming the termination precondition, perform an forward relational static analysis of the loop to determine the loop invariant
3. Assuming the loop invariant, perform an forward relational static analysis of the loop body to determine the loop abstract operational semantics
4. Assuming the loop semantics, use an abstraction of Floyd’s ranking function method to prove termination of the loop

Forward/reachability properties

Example: partial correctness (must stay into safe states)

Backward/ancestry properties

Example: termination (must reach final states)

Forward/backward properties

Example: total correctness (stay safe while reaching final states)
Arithmetic Mean Example: Termination Precondition (1)

\{x\geq y\}
while (x \neq y) do
\{x\geq y + 2\}
\quad x := x - 1;
\{x\geq y + 1\}
y := y + 1
\{x\geq y\}
od
\{x=y\}

Idea 1

The auxiliary termination counter method

Add an auxiliary termination counter to enforce (bounded) termination in the backward analysis!

Arithmetic Mean Example: Termination Precondition (2)

\{x=y+2k,x\geq y\}
while (x \neq y) do
\{x=y+2k,x\geq y+2\}
k := k - 1;
\{x=y+2k+2,x\geq y+2\}
x := x - 1;
\{x=y+2k+1,x\geq y+1\}
y := y + 1
\{x=y+2k,x\geq y\}
od
\{x=y,k=0\}
assume (k = 0)
\{x=y,k=0\}

1. Perform an iterated forward/backward relational static analysis of the loop with termination hypothesis to determine a necessary proper termination precondition
2. Assuming the termination precondition, perform an forward relational static analysis of the loop to determine the loop invariant
3. Assuming the loop invariant, perform an forward relational static analysis of the loop body to determine the loop abstract operational semantics
4. Assuming the loop semantics, use an abstraction of Floyd’s ranking function method to prove termination of the loop
Arithmetic Mean Example: Loop Invariant

\[
\begin{align*}
\text{assume } & (x = y + 2 \cdot k) \land (x \geq y); \\
& \{x = y + 2 \cdot k, x \geq y\} \\
\text{while } & (x \neq y) \text{ do} \\
& \{x = y + 2 \cdot k, x \geq y + 2\} \\
& k := k - 1; \\
& \{x = y + 2 \cdot k + 2, x \geq y + 2\} \\
& x := x - 1; \\
& \{x = y + 2 \cdot k + 1, x \geq y + 1\} \\
& y := y + 1 \\
& \{x = y + 2 \cdot k, x \geq y\} \\
\text{od} \\
& \{k = 0, x = y\}
\end{align*}
\]

1. Perform an iterated forward/backward relational static analysis of the loop with termination hypothesis to determine a necessary proper termination precondition
2. Assuming the termination precondition, perform an forward relational static analysis of the loop to determine the loop invariant
3. Assuming the loop invariant, perform an forward relational static analysis of the loop body to determine the loop abstract operational semantics
4. Assuming the loop semantics, use an abstraction of Floyd’s ranking function method to prove termination of the loop

Arithmetic Mean Example: Body Relational Semantics

\[
\begin{align*}
\text{Case } x < y: \\
\text{assume } & (x = y + 2 \cdot k) \land (x \geq y + 2); \\
& \{x = y + 2 \cdot k, x \geq y + 2\} \\
\text{assume } & (x < y); \\
& \text{empty(6)} \\
\text{assume } & (x_0 = x) \land (y_0 = y) \land (k_0 = k); \\
& \text{empty(6)} \\
& k := k - 1; \\
& x := x - 1; \\
& y := y + 1 \\
& \text{empty(6)}
\end{align*}
\]

\[
\begin{align*}
\text{Case } x > y: \\
\text{assume } & (x = y + 2 \cdot k) \land (x \geq y + 2); \\
& \{x = y + 2 \cdot k, x \geq y + 2\} \\
\text{assume } & (x > y); \\
& \text{empty(6)} \\
\text{assume } & (x_0 = x) \land (y_0 = y) \land (k_0 = k); \\
& \text{empty(6)} \\
& k := k - 1; \\
& x := x - 1; \\
& y := y + 1 \\
& \{x + 2 = y + 2 \cdot k, y = y_0 + 1, x + 1 = x_0, y = y + 1\}
\end{align*}
\]

1. Perform an iterated forward/backward relational static analysis of the loop with termination hypothesis to determine a necessary proper termination precondition
2. Assuming the termination precondition, perform an forward relational static analysis of the loop to determine the loop invariant
3. Assuming the loop invariant, perform an forward relational static analysis of the loop body to determine the loop abstract operational semantics
4. Assuming the loop semantics, use an abstraction of Floyd’s ranking function method to prove termination of the loop
Floyd’s method for termination of while B do C

Given a loop invariant $I$, find an $\mathbb{R}/\mathbb{Q}/\mathbb{Z}$-valued unknown rank function $r$ such that:

- The rank is nonnegative:
  $$\forall x_0, x : I(x_0) \land [B; C](x_0, x) \Rightarrow r(x_0) \geq 0$$

- The rank is strictly decreasing:
  $$\forall x_0, x : I(x_0) \land [B; C](x_0, x) \Rightarrow r(x) \leq r(x_0) - \eta$$

$\eta \geq 1$ for $\mathbb{Z}$, $\eta > 0$ for $\mathbb{R}/\mathbb{Q}$ to avoid Zeno $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots$

Arithmetic Mean Example: Ranking Function

Input the loop abstract semantics

Proving Termination by Parametric Abstraction, Lagrangian Relaxation and Semidefinite Programming
Idea 2

Express the loop invariant and relational semantics as numerical positivity constraints

Relational semantics of while B do C od loops

- $x_0 \in \mathbb{R}/\mathbb{Q}/\mathbb{Z}$: values of the loop variables before a loop iteration
- $x \in \mathbb{R}/\mathbb{Q}/\mathbb{Z}$: values of the loop variables after a loop iteration
- $I(x_0)$: loop invariant, $[[B;C](x_0, x)$: relational semantics of one iteration of the loop body
- $I(x_0) \land [B;C](x_0, x) = \bigwedge_{i=1}^{N} \sigma_i(x_0, x) \geq 0$ ($\geq \in \{>, \ge, =\}$)
- not a restriction for numerical programs

Example of linear program (Arithmetic mean)

$[A A'] [x_0 x] \geq b$

{x=y+2k, x>y}
while (x <> y) do
  k := k - 1;
  x := x - 1;
  y := y + 1
od

Example of quadratic form program (factorial)

$[x x'] A [x x] + 2[x x'] q + r \geq 0$

n := 0;
f := 1;
while (f <= N) do
  n := n + 1;
f := n * f
od
Example of semialgebraic program (logistic map)

\[
\text{eps} = 1.0 \times 10^{-9};
\text{while (0 <= a) & (a <= 1 - eps) & (eps <= x) & (x <= 1) do}
\text{x := a*x*(1-x)}
\text{od}
\]

Floyd’s method for termination of while B do C

Find an \(\mathbb{R}/\mathbb{Q}/\mathbb{Z}\)-valued unknown rank function \(r\) and \(\eta > 0\) such that:

– The rank is nonnegative:

\[
\forall x_0, x : \bigwedge_{i=1}^{N} \sigma_i(x_0, x) \geq 0 \Rightarrow r(x_0) \geq 0
\]

– The rank is strictly decreasing:

\[
\forall x_0, x : \bigwedge_{i=1}^{N} \sigma_i(x_0, x) \geq 0 \Rightarrow r(x_0) - r(x) - \eta \geq 0
\]

Idea 3

Eliminate the conjunction \(\land\) and implication \(\Rightarrow\) by Lagrangian relaxation

Implication (general case)

\[
A \Rightarrow B \iff \forall x \in A : x \in B
\]
Implication (linear case)

\[ A \Rightarrow B \quad (\text{assuming } A \neq \emptyset) \]

\( \iff \) (soundness)

\( \Rightarrow \) (completeness)

border of \( A \) parallel to border of \( B \)

Lagrangian relaxation (linear case)

Lagrangian relaxation, formally

Let \( \mathbb{V} \) be a finite dimensional linear vector space, \( N > 0 \) and \( \forall k \in [0, N] : \sigma_k \in \mathbb{V} \mapsto \mathbb{R} \).

\[ \forall x \in \mathbb{V} : \left( \bigwedge_{k=1}^{N} \sigma_k(x) \geq 0 \right) \Rightarrow (\sigma_0(x) \geq 0) \]

\( \iff \) soundness (Lagrange)

\( \Rightarrow \) completeness (lossless)

\( \not\Rightarrow \) incompleteness (lossy)

\( \exists \lambda \in [1, N] \mapsto \mathbb{R}^+ : \forall x \in \mathbb{V} : \sigma_0(x) - \sum_{k=1}^{N} \lambda_k \sigma_k(x) \geq 0 \)

relaxation = approximation, \( \lambda_i = \text{Lagrange coefficients} \)

Lagrangian relaxation, equality constraints

\[ \forall x \in \mathbb{V} : \left( \bigwedge_{k=1}^{N} \sigma_k(x) = 0 \right) \Rightarrow (\sigma_0(x) \geq 0) \]

\( \iff \) soundness (Lagrange)

\[ \exists \lambda \in [1, N] \mapsto \mathbb{R}^+ : \forall x \in \mathbb{V} : \sigma_0(x) - \sum_{k=1}^{N} \lambda_k \sigma_k(x) \geq 0 \]

\[ \land \exists \lambda' \in [1, N] \mapsto \mathbb{R}^+ : \forall x \in \mathbb{V} : \sigma_0(x) + \sum_{k=1}^{N} \lambda'_k \sigma_k(x) \geq 0 \]

\( \iff \) (\( \lambda'' = \frac{\lambda' - \lambda}{2} \))

\[ \exists \lambda'' \in [1, N] \mapsto \mathbb{R} : \forall x \in \mathbb{V} : \sigma_0(x) - \sum_{k=1}^{N} \lambda''_k \sigma_k(x) \geq 0 \]
Example: affine Farkas’ lemma, informally

- An application of Lagrangian relaxation to the case when $A$ is a polyhedron

Example: affine Farkas’ lemma, formally

- Formally, if the system $Ax + b \geq 0$ is feasible then
  \[ \forall x : Ax + b \geq 0 \Rightarrow cx + d \geq 0 \]
  \[ \Leftrightarrow \text{(soundness, Lagrange)} \]
  \[ \Rightarrow \text{(completeness, Farkas)} \]
  \[ \exists \lambda \geq 0 : \forall x : cx + d - \lambda(Ax + b) \geq 0 . \]

Yakubovich’s S-procedure, informally

- An application of Lagrangian relaxation to the case when $A$ is a quadratic form

Incompleteness (convex case)
Yakubovich’s S-procedure, completeness cases

- The constraint \( \sigma(x) \geq 0 \) is regular if and only if \( \exists \xi \in \mathbb{V} : \sigma(\xi) > 0 \).
- The S-procedure is lossless in the case of one regular quadratic constraint:

\[
\forall x \in \mathbb{R}^n : x^\top P_1 x + 2 q_1^\top x + r_1 \geq 0 \implies x^\top P_0 x + 2 q_0^\top x + r_0 \geq 0
\]

(Lagrange)

\[
\exists \lambda \geq 0 : \forall x \in \mathbb{R}^n : x^\top \left( \begin{bmatrix} P_0 & q_0 \\ q_0 & r_0 \end{bmatrix} - \lambda \begin{bmatrix} P_1 & q_1 \\ q_1 & r_1 \end{bmatrix} \right) x \geq 0.
\]

Floyd’s method for termination of while B do C

Find an \( \mathbb{R}/\mathbb{Q}/\mathbb{Z} \)-valued unknown rank function \( r \) which is:

- **Nonnegative**: \( \exists \lambda \in [1, N] : \mapsto \mathbb{R}^+: \)

\[
\forall x_0, x : r(x_0) - \sum_{i=1}^{N} \lambda_i \sigma_i(x_0, x) \geq 0
\]

- **Strictly decreasing**: \( \exists \eta > 0 : \exists \lambda' \in [1, N] : \mapsto \mathbb{R}^+: \)

\[
\forall x_0, x : (r(x_0) - r(x) - \eta) - \sum_{i=1}^{N} \lambda'_i \sigma_i(x_0, x) \geq 0
\]

Idea 4

Parametric abstraction of the ranking function \( r \)

- How can we compute the ranking function \( r \)?

→ parametric abstraction:

1. Fix the form \( r_a \) of the function \( r \) a priori, in term of unknown parameters \( a \)
2. Compute the parameters \( a \) numerically

- Examples:

\[
\begin{align*}
    r_a(x) &= a.x^\top & \text{linear} \\
    r_a(x) &= a.(x 1)^\top & \text{affine} \\
    r_a(x) &= (x 1).a.(x 1)^\top & \text{quadratic}
\end{align*}
\]
Floyd’s method for termination of while B do C

Find $\mathbb{R}/\mathbb{Q}/\mathbb{Z}$-valued unknown parameters $a$, such that:

- **Nonnegative**: $\exists \lambda \in [1, N] \mapsto \mathbb{R}^+ :$
  $$\forall x_0, x : r_a(x_0) - \sum_{i=1}^{N} \lambda_i \sigma_i(x_0, x) \geq 0$$

- **Strictly decreasing**: $\exists \eta > 0 : \exists \lambda' \in [1, N] \mapsto \mathbb{R}^+ :$
  $$\forall x_0, x : (r_a(x_0) - r_a(x) - \eta) - \sum_{i=1}^{N} \lambda'_i \sigma_i(x_0, x) \geq 0$$

---

**Mathematical programming**

$$\exists x \in \mathbb{R}^n : \bigwedge_{i=1}^{N} g_i(x) \geq 0$$

[Minimizing $f(x)$]

**Feasibility problem**: find a solution to the constraints

**Optimization problem**: find a solution, minimizing $f(x)$

Example: Linear programming

$$\exists x \in \mathbb{R}^n : A x \geq b$$

[Minimizing $cx$]

---

**Feasibility**

- **feasibility problem**: find a solution $s \in \mathbb{R}^n$ to the optimization program, such that $\bigwedge_{i=1}^{N} g_i(s) \geq 0$, or to determine that the problem is **infeasible**
- **feasible set**: $\{x | \bigwedge_{i=1}^{N} g_i(x) \geq 0\}$
- A feasibility problem can be converted into the optimization program

$$\min\{y \in \mathbb{R} | \bigwedge_{i=1}^{N} g_i(x) - y \geq 0\}$$
Semidefinite programming

\[
\exists x \in \mathbb{R}^n: \quad M(x) \succ 0
\]

[Minimizing \( cx \)]

Where the linear matrix inequality (LMI) is

\[
M(x) = M_0 + \sum_{k=1}^{n} x_k M_k
\]

with symmetric matrices \((M_k = M_k^T)\) and the positive semidefiniteness is

\[
M(x) \succ 0 = \forall X \in \mathbb{R}^N : X^\top M(x) X \geq 0
\]

Semidefinite programming, once again

Feasibility is:

\[
\exists x \in \mathbb{R}^n: \quad \forall X \in \mathbb{R}^N : X^\top \left( M_0 + \sum_{k=1}^{n} x_k M_k \right) X \geq 0
\]

of the form of the formulae we are interested in for programs which semantics can be expressed as LMIs:

\[
\bigwedge_{i=1}^{N} \sigma_i(x_0, x) \geq_i 0 = \bigwedge_{i=1}^{N} (x_0 x 1) M_i(x_0 x 1)^\top \geq_i 0
\]

Floyd’s method for termination of while B do C

Find \(\mathbb{R}/\mathbb{Q}/\mathbb{Z}\)-valued unknown parameters \(a\), such that:

- Nonnegative: \(\exists \lambda \in [1, N] \mapsto \mathbb{R}^+\) :

\[
\forall x_0, x : r_a(x_0) - \sum_{i=1}^{N} \lambda_i(x_0, x 1) M_i(x_0, x 1)^\top \geq 0
\]

- Strictly decreasing: \(\exists \eta > 0 : \exists \lambda' \in [1, N] \mapsto \mathbb{R}^+\) :

\[
\forall x_0, x : (r_a(x_0) - r_a(x) - \eta) - \sum_{i=1}^{N} \lambda'_i(x_0, x 1) M_i(x_0, x 1)^\top \geq 0
\]

Main steps in a typical soundness/completeness proof

\[
\exists r : \forall x, x' : [B;C](x, x') \Rightarrow r(x, x') \geq 0
\]

\[\iff\]

\[
\exists r : \forall x, x' : \bigwedge_{k=1}^{N} \sigma_k(x, x') \geq 0 \Rightarrow r(x, x') \geq 0
\]

\[\iff\]\(\exists r : \forall x, x' : \bigwedge_{k=1}^{N} \sigma_k(x, x') \geq 0 \Rightarrow r(x, x') \geq 0\)

\[\iff\]\(\exists r : \forall x, x' : \bigwedge_{k=1}^{N} \sigma_k(x, x') \geq 0 \Rightarrow r(x, x') \geq 0\)

\[\iff\]\(\exists r : \forall x, x' : \bigwedge_{k=1}^{N} \sigma_k(x, x') \geq 0 \Rightarrow r(x, x') \geq 0\)

\[\iff\]\(\exists r : \forall x, x' : \bigwedge_{k=1}^{N} \sigma_k(x, x') \geq 0 \Rightarrow r(x, x') \geq 0\)
\[ \exists r : \exists \lambda \in [1, N] \mapsto \mathbb{R}_* : \forall x, x' \in \mathbb{D}^n : r(x, x') - \sum_{k=1}^{N} \lambda_k (x \cdot x' 1) M_k (x \cdot x' 1)^T \geq 0 \]

\[ \Longleftrightarrow \exists M_0 : \exists \lambda \in [1, N] \mapsto \mathbb{R}_* : \forall x, x' \in \mathbb{D}^n : (x \cdot x' 1) M_0 (x \cdot x' 1)^T - \sum_{k=1}^{N} \lambda_k (x \cdot x' 1) M_k (x \cdot x' 1)^T \geq 0 \]

\[ \Longleftrightarrow \exists M_0 : \exists \lambda \in [1, N] \mapsto \mathbb{R}_* : \forall x, x' \in \mathbb{D}^n : (x \cdot x' 1) M_0 (x \cdot x' 1)^T \geq 0 \]

\[ \Longleftrightarrow \exists M_0 : \exists \lambda \in [1, N] \mapsto \mathbb{R}_* : \forall x, x' \in \mathbb{D}^{(n \times 1)} : \begin{bmatrix} x \\ x' \\ 1 \end{bmatrix}^T \begin{bmatrix} M_0 - \sum_{k=1}^{N} \lambda_k M_k \end{bmatrix} \begin{bmatrix} x \\ x' \\ 1 \end{bmatrix} \geq 0 \]

\[ \Longleftrightarrow \text{if } (x \cdot 1) A (x \cdot 1)^T \geq 0 \text{ for all } x, \text{ this is the same as } (y \cdot t) A (y \cdot t)^T \geq 0 \text{ for all } y \text{ and all } t \neq 0 \text{ (multiply the original inequality by } t^2 \text{ and call } x t = y). \text{ Since the latter inequality holds true for all } x \text{ and all } t \neq 0, \text{ by continuity it holds true for all } x, t, \text{ that is, the original inequality is equivalent to positive semidefiniteness of } A \]

Idea 6

Solve the convex constraints by semidefinite programming
The simplex for linear programming

\[ \begin{align*}
  y \\
  AX \geq b
\end{align*} \]

Dantzig 1948, exponential in worst case, good in practice

Polynomial methods

**Ellipsoid method**: Khachian 1979 [11], polynomial in worst case but not good in practice

**Interior point method**: Narendra Karmarkar 1984 [12], polynomial in worst case and good in practice (hundreds of thousands of variables)

Reference


Interior point method for semidefinite programming

- Nesterov & Nemirovskii 1988 [13], polynomial in worst case and good in practice (thousands of variables)

- Various path strategies e.g. “stay in the middle”

Semidefinite programming solvers

Numerous solvers available under MATHLAB®, a.o.:

- lmiLab: P. Gahinet, A. Nemirovskii, A. J. Laub, M. Chilali
- Sdp1r: S. Burer, R. Monteiro, C. Choi
- Sdpt3: R. Tütüncü, K. Toh, M. Todd
- SeDuMi: J. Sturm
- bnb: J. Lőfberg (integer semidefinite programming)

Common interfaces to these solvers, a.o.:

- Yalmip: J. Lőfberg

Sometime need some help (feasibility radius, shift, ...)

Linear program: termination of Euclidean division

```matlab
> clear all
% linear inequalities
% y0 q1 r0
A = [0 0 0; 0 0 0; 0 0 0];
% y q r
A1 = [1 0 0; % y-1 >=0 0 1 0; % q-1 >=0 0 0 1; % r >=0
bi = [-1; -1; 0];% y0 q1 r0
% linear equalities
% y0 q1 r0
Ae = [0 -1 0; % y0 + q + r =0 -1 0 0; % y0 + y =0 0 0 -1; % r0 + y + r =0
be = [-1; 0; 0];
```

Iterated forward/backward polyhedral analysis:

```matlab
{y>=1} q := 0; {q=0,y>=1}
{x=r,q=0,y>=1} r := x;
{y<=r} while (y <= r) do
{y<=r,q>=0} r := r - y;
{r>=q0,y>=0} q := q + 1
{r>=q0,y>=1} od
{q>=0,y>=r+1}
```

Reference

\[ [\text{InToMk}(A_i, A_i, b_i)] = \text{linToMk}(A_i, A_i, b_i);\]
\[ [\text{InToMk}(A_e, A_e, b_e)] = \text{linToMk}(A_e, A_e, b_e);\]
\[ [v_0, v] = \text{variables}(‘y’, ‘q’, ‘r’);\]
\[ \text{display}_Mk(M_k, N, v_0, v);\]
\[ +1.y -1 >= 0\]
\[ +1.q -1 >= 0\]
\[ +1.r >= 0\]
\[ -1.q0 +1.q -1 <= 0\]
\[ -1.y0 +1.y = 0\]
\[ -1.r0 +1.y +1.r = 0\]
\[ [\text{diagnostic}, R] = \text{termination}(v_0, v, M_k, N, ‘integer’, ‘quadratic’);\]
\[ \text{disp}(\text{diagnostic})\]
\[ \text{intrank}(R, v)\]
\[ r(y, q, r) = -2.y +2.q +6.r\]

Floyd’s proposal \( r(x, y, q, r) = x - q \) is more intuitive but requires to discover the nonlinear loop invariant \( x = r + q.y \).

---

**Idea 7**

Convex abstraction of non-convex constraints
Semidefinite programming relaxation for polynomial programs

\[ \text{eps} = 1.0e-9; \]
\[ \text{while } (0 \leq a) \& (a \leq 1 - \text{eps}) \]
\[ \& (\text{eps} \leq x) \& (x \leq 1) \text{ do} \]
\[ x := a \times x \times (1-x) \]
\[ \text{od} \]

Write the verification conditions in polynomial form, use SOS solver to relax in semidefinite programming form.

SOS tool + SeDuMi:
\[ r(x) = 1.222356e-13 \times x + 1.406392e+00 \]

Principle
- Show \( \forall x : p(x) \geq 0 \) by \( \forall x : p(x) = \sum_{i=1}^{k} q_i(x)^2 \)
- Hilbert’s 17th problem (sum of squares)
- Undecidable (but for monovariable or low degrees)
- Look for an approximation (relaxation) by semidefinite programming

General relaxation/approximation idea
- Write the polynomials in quadratic form with monomials as variables: \( p(x, y, \ldots) = z^\top Q z \) where \( Q \succ 0 \) is a semidefinite positive matrix of unknowns and \( z = [\ldots x^2, xy, y^2, \ldots x, y, \ldots] \) is a monomial basis
- If such a \( Q \) does exist then \( p(x, y, \ldots) \) is a sum of squares³
- The equality \( p(x, y, \ldots) = z^\top Q z \) yields LMI constrains on the unkown \( Q : z^\top M(Q) z \succ 0 \)

³ Since \( Q \succ 0 \), \( Q \) has a Cholesky decomposition \( L \) which is an upper triangular matrix \( L \) such that \( Q = L^\top L \).
It follows that \( p(x) = z^\top Q z = z^\top L^\top L z = (Lz)^\top L z = [L_{x_1} \ldots z] [L_{x_2} \ldots z] = \sum (L_{x_i} \cdot z)^2 \) (where \( \cdot \) is the vector dot product \( x \cdot y = \sum x_{ij} y_{ij} \)), proving that \( p(x) \) is a sum of squares whence \( \forall x : p(x) \geq 0 \), which eliminates the universal quantification on \( x \).
Considering More General Forms of Programs

Handling disjunctive loop tests and tests in loop body

- By case analysis
- and "conditional Lagrangian relaxation" (Lagrangian relaxation in each of the cases)

Loop body with tests

```
while (x < y) do
  if (i >= 0) then
    x := x+i+1
  else
    y := y+i
  fi
od
```

```
lmilab:
r(i,x,y) = -2.252791e-09.i -4.355697e+07.x +4.355697e+07.y +5.502903e+08
```

Quadratic termination of linear loop

```
{n>=0}
i := n; j := n;
while (i <> 0) do
  if (j > 0) then
    j := j - 1
  else
    j := n; i := i - 1
  fi
od
```

termination precondition determined by iterated forward/backward polyhedral analysis
sdplr (with feasibility radius of 1.0e+3):

\[ r(n, i, j) = +7.024176e-04 \cdot n^2 + 4.394909e-05 \cdot n \cdot i \ldots 
-2.809222e-03 \cdot n \cdot j + 1.533829e-02 \cdot n \ldots 
+1.569773e-03 \cdot i^2 + 7.077127e-05 \cdot i \cdot j \ldots 
+3.093629e+01 \cdot i - 7.021870e-04 \cdot j^2 \ldots 
+9.940151e-01 \cdot j + 4.237694e+00 \]

Successive values of \( r(n, i, j) \) for \( n = 10 \) on loop entry

Example of termination of nested loops:

**Bubblesort inner loop**

Iterated forward/backward polyhedral analysis followed by forward analysis of the body:

\[
\begin{align*}
+1 \cdot i' - 1 &> 0 \\
+1 \cdot j' - 1 &> 0 \\
+1 \cdot n0' - 1 \cdot i' &> 0 \\
-1 \cdot j &+ 1 \cdot j' - 1 = 0 \\
-1 \cdot i &+ 1 \cdot i' = 0 \\
-1 \cdot n &+ 1 \cdot n0' = 0 \\
+1 \cdot n0' - 1 \cdot n' &> 0 \\
\end{align*}
\]

termination (lmlab)

\[ r(n0, n, i, j) = +434297566 \cdot n0 + 226687644 \cdot n - 72551842 \cdot i \\
-2 \cdot j + 2147483647 \]

Example of termination of nested loops:

**Bubblesort outer loop**

Iterated forward/backward polyhedral analysis followed by forward analysis of the body:

\[
\begin{align*}
+1 \cdot i'' &> 0 \\
+1 \cdot n0'' - 1 \cdot i'' &> 0 \\
+1 \cdot i' - 1 \cdot j' &< 1 = 0 \\
-1 \cdot i &+ 1 \cdot i'' + 1 = 0 \\
-1 \cdot n &+ 1 \cdot n0'' = 0 \\
+1 \cdot n0' - 1 \cdot n0'' &> 0 \\
+1 \cdot n0' - 1 \cdot n' &> 0 \\
\end{align*}
\]

termination (lmlab)

\[ r(n0, n, i, j) = +24348786 \cdot n0 + 16834142 \cdot n + 100314562 \cdot i + 65646865 \]

Handling nested loops

- **by induction on the loop depth**
- use an iterated forward/backward symbolic analysis to get a necessary **termination precondition**
- use a forward symbolic symbolic analysis to get the semantics of a loop body
- use Lagrangian relaxation and semidefinite programming to get the **ranking function**
Handling nondeterminacy

- By case analysis
- Same for concurrency by interleaving
- Same with fairness by nondeterministic interleaving with encoding of an explicit scheduler

Termination of a concurrent program

[1] 1: while (x+2 < y) do while (x+2 < y) do
 2: [x := x + 1] if ?=0 then
       od x := x + 1
       else if ?=0 then
           interleaving
       else
       y := y - 1

[2] 1: while (x+2 < y) do
 2: [y := y - 1] x := x + 1;
       od y := y - 1

3: fi
fi

penbmi: r(x,y) = 2.537395e+00.x+-2.537395e+00.y+-2.046610e-01

Termination of a fair parallel program

[[ while [(x>0)|(y>0) do x := x - 1] od ||
   while [(x>0)|(y>0) do y := y - 1] od ]] interleave

{m>=1} termination precondition determined by iterated
forward/backward polyhedral analysis

t := ?; if (s = 0) then
assume (0 <= t & t <= 1);
t := 0
else
s := ?;
t := 1
fi; assume ((1 <= s) & (s <= m));
while ((x > 0) | (y > 0)) do
if (t = 1) then
x := x - 1
else
y := y - 1
fi;
s := s - 1;
else
skip
fi

fi fi

if (s = 0) then

if (t = 1) then

else


s := s - 1;
fi

skip
fi

penbmi: r(x,y,m,s,t) = +1.000468e+00.x +1.000611e+00.y +2.855769e-02.m -3.929197e-07.s +6.588027e-06.t +9.998392e+03

Relaxed Parametric Invariance Proof Method
Floyd’s method for invariance

Given a loop precondition $P$, find an unknown loop invariant $I$ such that:

- The invariant is \( \textit{initial} \):
  \[
  \forall x : P(x) \Rightarrow I(x)
  \]

- The invariant is \( \textit{inductive} \):
  \[
  \forall x, x' : I(x) \land [B; C](x, x') \Rightarrow I(x')
  \]

Abstraction

- Express loop semantics as a conjunction of \( \text{LMI constraints} \) (by relaxation for polynomial semantics)
- Eliminate the conjunction and implication by \text{Lagrangian relaxation}
- Fix the form of the unknown invariant by \text{parametric abstraction}

... we get ...

Floyd’s method for numerical programs

Find $\mathbb{R}/\mathbb{Q}/\mathbb{Z}$-valued unknown parameters $a$, such that:

- The invariant is \( \textit{initial} \): \( \exists \mu \in \mathbb{R}^+ : \)
  \[
  \forall x : I_a(x) - \mu P(x) \geq 0
  \]

- The invariant is \( \textit{inductive} \): \( \exists \lambda \in [0, N] \longrightarrow \mathbb{R}^+ : \)
  \[
  \forall x, x' : I_a(x') - \lambda_0 I_a(x) - \sum_{k=1}^{N} \lambda_k \sigma_k(x, x') \geq 0
  \]
  \[
  \text{bilinear in } \lambda_0 \text{ and } a
  \]

Idea 8

Solve the bilinear matrix inequality (BMI) by semidefinite programming
Bilinear matrix inequality (BMI) solvers

\[ \exists x \in \mathbb{R}^n : \bigwedge_{i=1}^m \left( M_0^i + \sum_{k=1}^n x_k M_k^i + \sum_{k=1}^n \sum_{\ell=1}^n x_k x_\ell N_{k\ell}^i \geq 0 \right) \]

Two solvers available under MATLAB®:
- PenBMI: M. Kočvara, M. Stingl
- bminibnb: J. Löfberg

Common interfaces to these solvers:
- Yalmip: J. Löfberg

Example: linear invariant

Program:
- Invariant:
  \[ +2.14678e-12i -3.12793e-10j +0.486712 \geq 0 \]
- Less natural than \( i - 2j - 2 \geq 0 \)
- Alternative:
  - Determine parameters \((a)\) by other methods (e.g. random interpretation)
  - Use BMI solvers to check for invariance

Example: linear invariant

Program:
- Invariant:
  \[ +2.14678e-12i -3.12793e-10j +0.486712 \geq 0 \]
- Less natural than \( i - 2j - 2 \geq 0 \)
- Alternative:
  - Determine parameters \((a)\) by other methods (e.g. random interpretation)
  - Use BMI solvers to check for invariance

Constraint resolution failure

- infeasibility of the constraints does not mean “non termination” or “non invariance” but simply failure
- inherent to abstraction!
Numerical errors

- LMI/BMI solvers do numerical computations with rounding errors, shifts, etc
- ranking function is subject to numerical errors
- the hard point is to discover a candidate for the ranking function
- much less difficult, when the ranking function is known, to re-check for satisfaction (e.g. by static analysis)
- not very satisfactory for invariance (checking only ???)

Related work

- Linear case (Farkas lemma):
  - Invariants: Sankaranarayanan, Spima, Manna (CAV’03, SAS’04, heuristic solver)
  - Termination: Podelski & Rybalchenko (VMCAI’03, Lagrange coefficients eliminated by hand to reduce to linear programming so no disjunctions, no tests, etc)
  - Parallelization & scheduling: Feautrier, easily generalizable to nonlinear case

Seminal work

- LMI case, Lyapunov 1890, “an invariant set of a differential equation is stable in the sense that it attracts all solutions if one can find a function that is bounded from below and decreases along all solutions outside the invariant set”.

THE END, THANK YOU