Program Invariants

James Worrell

Department of Computer Science, Oxford University

(joint work with Ehud Hrushovski, Joël Ouaknine, Amaury Pouly)

WST 2018, Oxford
July 2018
destination (or origin) is \( v \). An interpretation \( I \) of a flowchart is a mapping of its edges on propositions. Some, but not necessarily all, of the free variables of these propositions may be variables manipulated by the

**Figure 1.** Flowchart of program to compute \( S = \sum_{j=1}^{n} a_j \) (\( n \geq 0 \))
The classical approach to the verification of temporal safety properties of programs requires the construction of **inductive invariants** at each program point, that is, assertions that are true on every program execution reaching that point, and moreover, that are closed under the strongest postcondition operator. **Automation of this construction is the main challenge in program verification.**

D. Beyer, T. Henzinger, R. Majumdar, and A. Rybelchenko
*Invariant Synthesis for Combined Theories*, 2007
$x := 3; \\
y := 2; \\
while 2y - x \geq -2 \ do \\
\begin{pmatrix} x \\ y \end{pmatrix} := \begin{pmatrix} 10 & -8 \\ 6 & -4 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} ;
Does This Loop Terminate?

\[
x := 3;
y := 2;
\text{while } 2y - x \geq -2 \text{ do }
\begin{pmatrix} x \\ y \end{pmatrix} := \begin{pmatrix} 10 & -8 \\ 6 & -4 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix};
\]
Does This Loop Terminate?

\[
x := 3; \\
y := 2; \\
\text{while } 2y - x \geq -2 \text{ do} \\
\begin{pmatrix} x \\ y \end{pmatrix} := \begin{pmatrix} 10 & -8 \\ 6 & -4 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix};
\]
x := 3;
y := 2;
while 2y − x ≥ −2 do
    \[
    \begin{pmatrix}
    x \\
y
    \end{pmatrix}
    :=
    \begin{pmatrix}
    10 & -8 \\
    6 & -4
    \end{pmatrix}
    \begin{pmatrix}
    x \\
y
    \end{pmatrix};
\]
Does This Loop Terminate?

\[
x := 3; \\
y := 2; \]

while \(2y - x \geq -2\) do

\[
\begin{pmatrix}
x \\
y
\end{pmatrix} := \begin{pmatrix} 10 & -8 \\ 6 & -4 \end{pmatrix} \begin{pmatrix} x \\
y
\end{pmatrix};
\]

Polynomial invariant:
\[
x - 9x^2 - y + 24xy - 16y^2 = 0
\]
Does This Loop Terminate?

\begin{align*}
x & := 3; \\
y & := 2; \\
\text{while } 2y - x \geq -2 \text{ do} \\
\begin{pmatrix}
x \\
y
\end{pmatrix} & := 
\begin{pmatrix}
10 & -8 \\
6 & -4
\end{pmatrix} 
\begin{pmatrix}
x \\
y
\end{pmatrix};
\end{align*}

Polynomial invariant: \( x - 9x^2 - y + 24xy - 16y^2 = 0 \)
Termination of Linear Loops

Linear-Loop Termination

Given a vector $\mathbf{x} \in \mathbb{Q}^d$, a halfspace $F \subseteq \mathbb{Q}^d$ and a matrix $A \in \mathbb{Q}^{d \times d}$, does there exist $n \in \mathbb{N}$ such that $A^n \mathbf{x} \in F$?
Termination of Linear Loops

Linear-Loop Termination

Given a vector \( x \in \mathbb{Q}^d \), a halfspace \( F \subseteq \mathbb{Q}^d \) and a matrix \( A \in \mathbb{Q}^{d \times d} \), does there exist \( n \in \mathbb{N} \) such that \( A^n x \in F \)?

- Open for many decades!
Termination of Linear Loops

Linear-Loop Termination

Given a vector \( \mathbf{x} \in \mathbb{Q}^d \), a halfspace \( F \subseteq \mathbb{Q}^d \) and a matrix \( A \in \mathbb{Q}^{d \times d} \), does there exist \( n \in \mathbb{N} \) such that \( A^n \mathbf{x} \in F \)?

Open for many decades!

"It is faintly outrageous that this problem is still open; it is saying that we do not know how to decide the Halting Problem even for 'linear' automata!"

Terence Tao
Better termination proving through cooperation

Marc Brockschmidt\textsuperscript{1}, Byron Cook\textsuperscript{2,3}, and Carsten Fuhs\textsuperscript{3}

\textsuperscript{1} RWTH Aachen University
\textsuperscript{2} Microsoft Research Cambridge
\textsuperscript{3} University College London

\textbf{Abstract.} One of the difficulties of proving program termination is managing the subtle interplay between the finding of a termination argument and the finding of the argument’s supporting invariant. In this
Better termination proving through cooperation

Marc Brockschmidt\textsuperscript{1}, Byron Cook\textsuperscript{2,3}, and Carsten Fuhs\textsuperscript{3}

\textsuperscript{1} RWTH Aachen University
\textsuperscript{2} Microsoft Research Cambridge
\textsuperscript{3} University College London

\textbf{Abstract.} One of the difficulties of proving program termination is managing the subtle interplay between the finding of a termination argument and the finding of the argument’s supporting invariant. In this

1 Introduction

When proving program termination we are simultaneously solving two problems: the search for a termination argument, and the search for a supporting invariant. Consider the following example:

\begin{verbatim}
y := 1;
while x > 0 do
   x := x - y;
   y := y + 1;
done
\end{verbatim}
Only 'nondeterministic' branching (no conditionals)

All assignments are affine

Also allow nondeterministic assignments

Affine programs:
can overapproximate more complex programs

already cover a range of existing formalisms, e.g. probabilistic and quantum automata,

...
Only ‘nondeterministic’ branching (no conditionals)
- Only ‘nondeterministic’ branching (no conditionals)
- All assignments are affine
From Flowcharts to Affine Programs

- Only ‘nondeterministic’ branching (no conditionals)
- All assignments are affine

\[ x := 7y - 3z + 2 \]
From Flowcharts to Affine Programs

- Only ‘nondeterministic’ branching (no conditionals)
- All assignments are affine
- Also allow nondeterministic assignments $x := ?$

Affine programs can overapproximate more complex programs already cover a range of existing formalisms, e.g. probabilistic and quantum automata.
From Flowcharts to Affine Programs

- Only ‘nondeterministic’ branching (no conditionals)
- All assignments are affine
- Also allow nondeterministic assignments $x := ?$
From Flowcharts to Affine Programs

- Only ‘nondeterministic’ branching (no conditionals)
- All assignments are affine
- Also allow nondeterministic assignments $x := ?$

Affine programs:
- can overapproximate more complex programs
Affine programs:

- can **overapproximate** more complex programs
- already cover a range of existing formalisms, e.g. **probabilistic** and **quantum automata**, ...
\textbf{invariant} = \textit{overapproximation} (of the reachable states)
Inductive Invariants

**invariant** = overapproximation (of the reachable states)

**inductive invariant** = \{ overapproximation preserved by the transition relation \}
Inductive Invariants

\[ x, y, z \text{ range over } Q \]

\[ \langle I_1, I_2, I_3 \rangle \] is an invariant \( I_1, I_2, I_3 \subseteq R^3 \)
\( x, y, z \) range over \( \mathbb{Z} \) (or \( \mathbb{Q} \))
$x, y, z$ range over $\mathbb{Z}$ (or $\mathbb{Q}$)
$x, y, z$ range over $\mathbb{Z}$ (or $\mathbb{Q}$)
Inductive Invariants

$x, y, z$ range over $\mathbb{Z}$ (or $\mathbb{Q}$)
$x, y, z$ range over $\mathbb{Z}$ (or $\mathbb{Q}$)
Inductive Invariants

$x, y, z$ range over $\mathbb{Z}$ (or $\mathbb{Q}$)
Inductive Invariants

\( x, y, z \) range over \( \mathbb{Z} \) (or \( \mathbb{Q} \))
Inductive Invariants

$x, y, z$ range over $\mathbb{Z}$ (or $\mathbb{Q}$)
Inductive Invariants

\[ x, y, z \text{ range over } \mathbb{Z} \text{ (or } \mathbb{Q}) \]
Inductive Invariants

\[ x, y, z \text{ range over } \mathbb{Z} \text{ (or } \mathbb{Q} \text{)} \]
$x, y, z$ range over $\mathbb{Z}$ (or $\mathbb{Q}$)
Inductive Invariants

$x, y, z$ range over $\mathbb{Z}$ (or $\mathbb{Q}$)
\( x, y, z \) range over \( \mathbb{Z} \) (or \( \mathbb{Q} \))
$x, y, z$ range over $\mathbb{Z}$ (or $\mathbb{Q}$)
$x, y, z$ range over $\mathbb{Z}$ (or $\mathbb{Q}$)
x, y, z range over $\mathbb{Z}$ (or $\mathbb{Q}$)
Inductive Invariants

$x, y, z$ range over $\mathbb{Z}$ (or $\mathbb{Q}$)
Inductive Invariants

$x, y, z$ range over $\mathbb{Z}$ (or $\mathbb{Q}$)
$x, y, z$ range over $\mathbb{Z}$ (or $\mathbb{Q}$)

$\langle l_1, l_2, l_3 \rangle$ is an invariant ($l_1, l_2, l_3 \subseteq \mathbb{R}^3$)
\[ x, y, z \text{ range over } \mathbb{Z} \text{ (or } \mathbb{Q} \text{)} \]

\[ \langle l_1, l_2, l_3 \rangle \text{ is an invariant } (l_1, l_2, l_3 \subseteq \mathbb{R}^3) \]
Inductive Invariants

$x, y, z$ range over $\mathbb{Z}$ (or $\mathbb{Q}$)

$\langle I_1, I_2, I_3 \rangle$ is an **inductive invariant** ($I_1, I_2, I_3 \subseteq \mathbb{R}^3$)
Inductive Invariants

$x, y, z$ range over $\mathbb{Z}$ (or $\mathbb{Q}$)

$\langle l_1, l_2, l_3 \rangle$ is an \textbf{inductive invariant} ($l_1, l_2, l_3 \subseteq \mathbb{R}^3$)
Inductive Invariants

$x, y, z$ range over $\mathbb{Z}$ (or $\mathbb{Q}$)

$\langle l_1, l_2, l_3 \rangle$ is an inductive invariant ($l_1, l_2, l_3 \subseteq \mathbb{R}^3$)
Inductive Invariants

$x, y, z$ range over $\mathbb{Z}$ (or $\mathbb{Q}$)

$\langle S_1, S_2, S_3 \rangle$ is always an inductive invariant
$x, y, z$ range over $\mathbb{Z}$ (or $\mathbb{Q}$)

$\langle R^3, R^3, R^3 \rangle$ is also always an inductive invariant
Inductive Invariants

$x, y, z$ range over $\mathbb{Z}$ (or $\mathbb{Q}$)

A good invariant is worth a thousand reachability queries!
Choose the right abstract domain

- Some domains always have ‘best’ (strongest, smallest) invariants, others not
Choose the right abstract domain

- Some domains always have ‘best’ (strongest, smallest) invariants, others not

Compute an invariant!

- Many eclectic methods: fixed-point computations, constraint solving, interpolation, abduction, machine learning, ...
- Some approaches require ‘widening’ to ensure termination
- Other techniques invoke e.g. dimension or algebraic arguments
- Trade-off between precision and tractability ...
Affine Relationships Among Variables of a Program*

Michael Karr

Received May 8, 1974

Summary. Several optimizations of programs can be performed when in certain regions of a program equality relationships hold between a linear combination of the variables of the program and a constant. This paper presents a practical approach to detecting these relationships by considering the problem from the viewpoint of linear algebra. Key to the practicality of this approach is an algorithm for the calculation of the “sum” of linear subspaces.

Theorem (Karr 76)

There is an algorithm which computes, for any given affine program over $\mathbb{Q}$, its strongest affine inductive invariant.
AUTOMATIC DISCOVERY OF LINEAR RESTRAINTS AMONG VARIABLES OF A PROGRAM

Patrick Cousot* and Nicolas Halbwachs**
Laboratoire d'Informatique, U.S.M.G., BP. 53
38041 Grenoble cedex, France

1. INTRODUCTION

The model of abstract interpretation of programs developed by Cousot [1976], Cousot [1977] is applied
to the static determination of linear equality or
inequality relations among variables of programs.

For example, consider the following sorting
procedure (Knuth [1973], p.107):

procedure BUBBLESORT(integer value N;
integer array[1:N] K):

A certain number of classical data flow analysis
techniques are included in or generalized by the
determination of linear equality relations among
program variables. For example constant propagation
can be understood as the discovery of very simple
linear equality relations among variables (such as
X=1, Y=5). However the resolution of the more
general problem of determining linear equality
relations among variables allows the discovery of
A Note on Karr’s Algorithm

Markus Müller-Olm¹ * and Helmut Seidl²

Abstract. We give a simple formulation of Karr’s algorithm for computing all affine relationships in affine programs. This simplified algorithm runs in time $O(nk^3)$ where $n$ is the program size and $k$ is the number of program variables assuming unit cost for arithmetic operations. This improves upon the original formulation by a factor of $k$. Moreover, our re-formulation avoids exponential growth of the lengths of intermediately occurring numbers (in binary representation) and uses less complicated elementary operations. We also describe a generalization that determines all polynomial relations up to degree $d$ in time $O(nk^{3d})$. 
Why Linear Invariants Are Not Enough

\[
s := 0; \\
x := 0; \\
\text{while } \langle \ldots \rangle \text{ do} \\
\qquad x := x + 1; \\
\qquad s := s + x;
\]

The loop invariant is:

\[
s = x(x+1)/2
\]

Or equivalently:

\[
p(s, x) = 2s - x^2 - x = 0
\]
\begin{verbatim}
  s := 0;
  x := 0;
  while ⟨...⟩ do
    x := x + 1;
    s := s + x;
\end{verbatim}

The loop invariant is:
\[ s = \frac{x(x + 1)}{2} \]
Why Linear Invariants Are Not Enough

The loop invariant is:

\[ s = \frac{x(x + 1)}{2} \]

Or equivalently:

\[ p(s, x) = 2s - x^2 - x = 0 \]
More on Polynomial Invariants

[Sankaranarayanan, Sipma, and Manna 04],
[Ildikó, Kovács, and T. Jebelean 05],
[Müller-Olm, Potter, Seidl 06],
[Rodríguez-Carbonell and Kapur 07a],
[Rodríguez-Carbonell and Kapur 07b],
[Colón 07],
[Kovács 08],
[Kapur 13],
[Cachera, Jensen, Jobin, and Kirchner 14],
[de Oliveira, Bensalem, and Prevosto 16],
[Humenberger, Jaroschek, L. Kovács 18],
...
More on Polynomial Invariants

[Sankaranarayanan, Sipma, and Manna 04],
[Ildikó, Kovács, and T. Jebelean 05],
[Müller-Olm, Potter, Seidl 06],
[Rodríguez-Carbonell and Kapur 07a],
[Rodríguez-Carbonell and Kapur 07b],
[Colón 07],
[Kovács 08],
[Kapur 13],
[Cachera, Jensen, Jobin, and Kirchner 14],
[de Oliveira, Bensalem, and Prevosto 16],
[Humenberger, Jaroschek, L. Kovács 18],

...}

Tools:

- **Aligator**: Gröbner bases
- **Fastind**: constraints-based; avoids Gröbner bases
- **Pilat**: linearisation
Non-linear Reasoning for Invariant Synthesis

ZACHARY KINCAID, Princeton University, USA
JOHN CYPHERT and JASON BRECK, University of Wisconsin, USA
THOMAS REPS, University of Wisconsin, USA and GrammaTech, Inc., USA

Automatic generation of non-linear loop invariants is a long-standing challenge in program analysis, with many applications. For instance, reasoning about exponentials provides a way to find invariants of digital-filter programs, and reasoning about polynomials and/or logarithms is needed for establishing invariants that describe the time or memory usage of many well-known algorithms. An appealing approach to this challenge is to exploit the powerful recurrence-solving techniques that have been developed in the field of computer algebra, which can compute exact characterizations of non-linear repetitive behavior. However, there is a gap...
Computing polynomial program invariants

Markus Müller-Olm\textsuperscript{a,*,1}, Helmut Seidl\textsuperscript{b}

\textsuperscript{a} FernUniversität Hagen, LG Praktische Informatik 5, 58084 Hagen, Germany
\textsuperscript{b} TU München, Informatik, 12, 85748 München, Germany

Received 16 October 2003; received in revised form 20 April 2004
Available online 19 June 2004
Computing polynomial program invariants

Markus Müller-Olm a,*, 1, Helmut Seidl b

a FernUniversität Hagen, LG Praktische Informatik 5, 58084 Hagen, Germany
b TU München, Informatik, 12, 85748 München, Germany

Received 16 October 2003; received in revised form 20 April 2004
Available online 19 June 2004

It is a challenging open problem whether or not the set of all valid polynomial relations can be computed not just the ones of some given form. It is not
Theorem (Hrushovski, Ouaknine, Pouly, W. 18)

There is an algorithm which computes, for any given affine program over $\mathbb{Q}$, its strongest polynomial inductive invariant.
There is an algorithm which computes, for any given affine program over \( \mathbb{Q} \), its strongest polynomial inductive invariant.

- **strongest polynomial invariant** ⇐⇒ **smallest algebraic set**
Theorem (Hrushovski, Ouaknine, Pouly, W. 18)

There is an algorithm which computes, for any given affine program over $\mathbb{Q}$, its strongest polynomial inductive invariant.

- strongest polynomial invariant $\iff$ smallest algebraic set
  - algebraic sets are defined by conjunctions of polynomial equalities
Theorem (Hrushovski, Ouaknine, Pouly, W. 18)

There is an algorithm which computes, for any given affine program over $\mathbb{Q}$, its strongest polynomial inductive invariant.

- **strongest polynomial invariant** $\iff$ **smallest algebraic set**
  - **algebraic sets** are defined by conjunctions of polynomial equalities
  - Algorithm computes for each program location the set of **all** polynomial relations that hold among program variables at whenever control reaches that location
Theorem (Hrushovski, Ouaknine, Pouly, W. 18)

There is an algorithm which computes, for any given affine program over $\mathbb{Q}$, its strongest polynomial inductive invariant.

- **strongest polynomial invariant** $\iff$ **smallest algebraic set**
  - **algebraic sets** are defined by conjunctions of polynomial equalities

- Algorithm computes for each program location the set of **all** polynomial relations that hold among program variables at whenever control reaches that location

- We represent this set of relations using a **finite basis** of polynomial equalities
Because affine functions are Zariski-continuous.
Computing Strongest Inductive Polynomial Invariants

\( S_1 \subseteq S_2 = \Rightarrow f_1(\overline{S}_1) \subseteq S_2 \). . . because affine functions are Zariski-continuous.
$f_1(S_1) \subseteq S_2$
$f_1(S_1) \subseteq S_2 \implies f_1(S_1^c) \subseteq S_2^c$
$f_1(S_1) \subseteq S_2 \implies f_1(S_1^c) \subseteq S_2^c$

... because affine functions are Zariski-continuous
Algebraic sets are closed under:

- finite unions (by taking products of polynomials)
Algebraic sets are closed under:

- finite unions (by taking products of polynomials)
- finite intersections (by definition)
Algebraic sets are closed under:

- finite unions (by taking products of polynomials)
- finite intersections (by definition)
- arbitrary intersections (by Hilbert Basis Theorem)

\[ V(\mathcal{P}) = V(p_1, \ldots, p_k) \text{ for some } p_1, \ldots, p_k \in \mathcal{P} \]
Algebraic sets are closed under:

- finite unions (by taking products of polynomials)
- finite intersections (by definition)
- arbitrary intersections (by Hilbert Basis Theorem)

\[
V(\mathcal{P}) = V(p_1, \ldots, p_k) \text{ for some } p_1, \ldots, p_k \in \mathcal{P}
\]

\[\implies\]

*algebraic sets* \(\equiv\) *closed sets* in the **Zariski topology**
Algebraic sets are closed under:

- finite unions (by taking products of polynomials)
- finite intersections (by definition)
- arbitrary intersections (by Hilbert Basis Theorem)

\[ V(\mathcal{P}) = V(p_1, \ldots, p_k) \text{ for some } p_1, \ldots, p_k \in \mathcal{P} \]

\[ \Rightarrow \]

*algebraic sets* \( \equiv \) *closed sets* in the **Zariski topology**

\[ \bar{S} = \text{‘smallest algebraic set containing } S\text{’} \]
Zariski Closure for a Single Loop

\[
\begin{pmatrix}
2 \cos \theta & -2 \sin \theta & 0 \\
2 \sin \theta & 2 \cos \theta & 0 \\
0 & 0 & 4
\end{pmatrix}
\]
Zariski Closure for a Single Loop

\[
\begin{pmatrix}
2 \cos \theta & -2 \sin \theta & 0 \\
2 \sin \theta & 2 \cos \theta & 0 \\
0 & 0 & 4
\end{pmatrix}
\]
Zariski Closure for a Single Loop

\[
\begin{pmatrix}
2 \cos \theta & -2 \sin \theta & 0 \\
2 \sin \theta & 2 \cos \theta & 0 \\
0 & 0 & 4
\end{pmatrix}
\]
Zariski Closure for a Single Loop

\[
\begin{pmatrix}
2 \cos \theta & -2 \sin \theta & 0 \\
2 \sin \theta & 2 \cos \theta & 0 \\
0 & 0 & 4
\end{pmatrix}
\]
Zariski Closure for a Single Loop

\[
\begin{pmatrix}
2 \cos \theta & -2 \sin \theta & 0 \\
2 \sin \theta & 2 \cos \theta & 0 \\
0 & 0 & 4
\end{pmatrix}
\]
Zariski Closure for a Single Loop

\[
\begin{pmatrix}
2 \cos \theta & -2 \sin \theta & 0 \\
2 \sin \theta & 2 \cos \theta & 0 \\
0 & 0 & 4
\end{pmatrix}
\]
From Affine Programs to Linear Semigroups

Each $M_i \in \mathbb{Q}^{d \times d}$
From Affine Programs to Linear Semigroups

Each $M_i \in \mathbb{Q}^{d \times d}$
From Affine Programs to Linear Semigroups

Each $M_i \in \mathbb{Q}^{d \times d}$
Zariski Closure of Linear Semigroups

\( M_1, \ldots, M_k \in \mathbb{Q}^{d \times d} \)

Theorem (Hrushovski, Ouaknine, Pouly, W. 18)

There is an algorithm which computes \( \langle M_1, \ldots, M_k \rangle \).

Outputs a finite list of polynomials \( p_1, \ldots, p_m \in \mathbb{Z}[x_1, \ldots, x_{2^d}] \) such that:

\[ \langle M_1, \ldots, M_k \rangle = V(p_1, \ldots, p_m) \]
Zariski Closure of Linear Semigroups

- $M_1, \ldots, M_k \in \mathbb{Q}^{d\times d}$

- Linear semigroup $\langle M_1, \ldots, M_k \rangle \subseteq \mathbb{Q}^{d\times d}$
Zariski Closure of Linear Semigroups

- $M_1, \ldots, M_k \in \mathbb{Q}^{d \times d}$
- Linear semigroup $\langle M_1, \ldots, M_k \rangle \subseteq \mathbb{Q}^{d \times d}$
- Zariski closure $\overline{\langle M_1, \ldots, M_k \rangle} \subseteq \mathbb{R}^{d \times d}$
Zariski Closure of Linear Semigroups

- \( M_1, \ldots, M_k \in \mathbb{Q}^{d \times d} \)
- Linear semigroup \( \langle M_1, \ldots, M_k \rangle \subseteq \mathbb{Q}^{d \times d} \)
- Zariski closure \( \overline{\langle M_1, \ldots, M_k \rangle} \subseteq \mathbb{R}^{d \times d} \)

**Theorem (Hrushovski, Ouaknine, Pouly, W. 18)**

There is an algorithm which computes \( \overline{\langle M_1, \ldots, M_k \rangle} \).
Zariski Closure of Linear Semigroups

- \( M_1, \ldots, M_k \in \mathbb{Q}^{d \times d} \)
- Linear semigroup \( \langle M_1, \ldots, M_k \rangle \subseteq \mathbb{Q}^{d \times d} \)
- Zariski closure \( \overline{\langle M_1, \ldots, M_k \rangle} \subseteq \mathbb{R}^{d \times d} \)

**Theorem (Hrushovski, Ouaknine, Pouly, W. 18)**

There is an algorithm which computes \( \overline{\langle M_1, \ldots, M_k \rangle} \).

Outputs a finite list of polynomials \( p_1, \ldots, p_m \in \mathbb{Z}[x_1, \ldots, x_{d^2}] \) such that:

\[
\overline{\langle M_1, \ldots, M_k \rangle} = \mathbf{V}(p_1, \ldots, p_m)
\]
ON FINITE SEMIGROUPS OF MATRICES*

Arnaldo MANDEL¹ and Imre SIMON²

Instituto de Matemática e Estatística, Universidade de São Paulo, 05508 São Paulo, SP, Brasil

Communicated by M. Nivat
Received February 1977

Abstract. Finite semigroups of $n$ by $n$ matrices over the naturals are characterized both by algebraic and combinatorial methods. Next we show that the cardinality of a finite semigroup $S$ of $n$ by $n$ matrices over a field is bounded by a function depending only on $n$, the number of generators of $S$ and the maximum cardinality of its subgroups. As a consequence, given $n$ and $k$, there exist, up to isomorphism, only a finite number of finite semigroups of $n$ by $n$ matrices over the rationals, generated by at most $k$ elements. Among other applications to Automaton Theory, we show that it is decidable whether the behavior of a given $N - \Sigma$ automaton is bounded.

1. Introduction

The results in this paper originated from the investigation of the following question in Automaton Theory: Is it decidable whether the behavior of a given $N - \Sigma$ automaton is bounded? This is answered affirmatively and it leads to the study of finite semigroups of matrices over the naturals. After obtaining effective characterizations of these semigroups, we investigate finite semigroups of matrices over a field. This enables us to generalize, to matrices over the rationals, one of the results obtained earlier.
ON FINITE SEMIGROUPS OF MATRICES*

Arnaldo MANDEL¹ and Imre SIMON²

Instituto de Matemática e Estatística, Universidade de São Paulo, 05508 São Paulo, SP, Brasil

Communicated by M. Nivat
Received February 1977

Abstract. Finite semigroups of $n$ by $n$ matrices over the naturals are characterized both by algebraic and combinatorial methods. Next we show that the cardinality of a finite semigroup $S$ of $n$ by $n$ matrices over a field is bounded by a function depending only on $n$, the number of generators of $S$ and the maximum cardinality of its subgroups. As a consequence, given $n$ and $k$, there exist, up to isomorphism, only a finite number of finite semigroups of $n$ by $n$ matrices over the rationals, generated by at most $k$ elements. Among other applications to Automaton Theory, we show that it is decidable whether the behaviour of a given $N - \Sigma$ automaton is bounded.

$\langle M_1, \ldots, M_k \rangle$ is finite

$\iff$

$\langle \bar{M}_1, \ldots, \bar{M}_k \rangle$ is finite!
Some Hard Problems for Linear Semigroups

Theorem (Markov 1947)

There is a fixed set of $6 \times 6$ integer matrices $M_1, \ldots, M_k$ such that the membership problem

"$M \in \langle M_1, \ldots, M_k \rangle$?" is undecidable.
Some Hard Problems for Linear Semigroups

Theorem (Markov 1947)

There is a fixed set of $6 \times 6$ integer matrices $M_1, \ldots, M_k$ such that the membership problem “$M \in \langle M_1, \ldots, M_k \rangle$?” is undecidable.

Mortality: Is the zero matrix contained in the semigroup generated by a given set of $n \times n$ matrices with integer entries?
Theorem (Markov 1947)

There is a fixed set of $6 \times 6$ integer matrices $M_1, \ldots, M_k$ such that the membership problem “$M \in \langle M_1, \ldots, M_k \rangle$?” is undecidable.

Mortality: Is the zero matrix contained in the semigroup generated by a given set of $n \times n$ matrices with integer entries?

Theorem (Paterson 1970)

The mortality problem is undecidable for $3 \times 3$ matrices.
Quantum automata and algebraic groups

Harm Derksen\textsuperscript{a}, Emmanuel Jeandel\textsuperscript{b}, Pascal Koiran\textsuperscript{b,*}

\textsuperscript{a}Department of Mathematics, University of Michigan, Ann Arbor, MI 48109, United States
\textsuperscript{b}Laboratoire de l’Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, 69364, France

Received 15 September 2003; accepted 1 November 2004

Abstract

We show that several problems which are known to be undecidable for probabilistic automata become decidable for quantum finite automata. Our main tool is an algebraic result of independent interest: we give an algorithm which, given a finite number of invertible matrices, computes the Zariski closure of the group generated by these matrices.

© 2005 Elsevier Ltd. All rights reserved.

Keywords: Quantum automata; Probabilistic automata; Undecidability; Algebraic groups; Algebraic geometry
Theorem (Masser 1988)

*Given algebraic numbers* $\lambda_1, \ldots, \lambda_k$, *there is a procedure to compute the set of multiplicative relations*

$$\{(n_1, \ldots, n_k) \in \mathbb{Z}^k : \lambda_1^{n_1} \cdots \lambda_k^{n_k} = 1\}.$$
Theorem (Masser 1988)

*Given algebraic numbers* $\lambda_1, \ldots, \lambda_k$, *there is a procedure to compute the set of multiplicative relations*

$$\{(n_1, \ldots, n_k) \in \mathbb{Z}^k : \lambda_1^{n_1} \cdots \lambda_k^{n_k} = 1\}.$$ 

Theorem (Schur 1911)

*Every finitely generated periodic subgroup of* $\text{GL}_n(\mathbb{C})$ *is finite.*
Define $G := \langle S, T, E \rangle$, where

$$S := \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}, \quad T := \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \quad E := \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}.$$
Define $G := \langle S, T, E \rangle$, where

$$S := \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}, \quad T := \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \quad E := \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}.\]$$

Then $G = \{ M \in M_2(\mathbb{R}) : \det(M) \in \{0, 1\} \}$. 
Define $G := \langle S, T, E \rangle$, where

$$S := \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}, \quad T := \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \quad E := \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}.$$ 

Then $G = \{ M \in M_2(\mathbb{R}) : \det(M) \in \{0, 1\} \}$.

Indeed, since

$$\{ M \in G : \text{rank}(M) = 2 \} = \langle S, T \rangle = \overline{\text{SL}_2(\mathbb{Z})} = \text{SL}_2(\mathbb{R}),$$
Define $G := \langle S, T, E \rangle$, where

\[
S := \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}, \quad T := \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \quad E := \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}.
\]

Then $G = \{ M \in M_2(\mathbb{R}) : \det(M) \in \{0, 1\} \}$. Indeed, since

\[
\{ M \in G : \text{rank}(M) = 2 \} = \langle S, T \rangle = \overline{\text{SL}_2(\mathbb{Z})} = \text{SL}_2(\mathbb{R}),
\]

we have that $\{ M \in G : \text{rank}(M) < 2 \}$ is generated by the definable set

\[
\{ MEM', ME, EM : M, M' \in \text{SL}_2(\mathbb{R}) \}.
\]
Given a definable set $A \subseteq M_n(\mathbb{C})$ of rank-$r$ matrices, there is a procedure to compute

$$S = \{ A \in \langle A \rangle : \text{rank}(A) = r \}.$$
Want to compute

\[ S = \{ A \in \langle A \rangle : \text{rank}(A) = r \} . \]
Want to compute

\[ S = \{ A \in \langle A \rangle : \text{rank}(A) = r \} . \]

Consider \( S \) as a category \( C(S) \), rather than a semigroup.
Want to compute

\[ S = \{ A \in \langle A \rangle : \text{rank}(A) = r \} \].

Consider \( S \) as a category \( C(S) \), rather than a semigroup.
Want to compute

\[ S = \{ A \in \langle A \rangle : \text{rank}(A) = r \} . \]

Consider \( S \) as a category \( C(S) \), rather than a semigroup.

Object \((U, V)\) s.t. \( U, V \subseteq \mathbb{C}^n \)
- \( U \cap V = 0 \)
- \( \dim(U) = n - r, \dim(V) = r \)
Want to compute

\[ S = \{ A \in \langle A \rangle : \text{rank}(A) = r \} . \]

Consider \( S \) as a category \( C(S) \), rather than a semigroup.

Object \( (U, V) \) s.t. \( U, V \subseteq \mathbb{C}^n \)
- \( U \cap V = 0 \)
- \( \dim(U) = n - r, \dim(V) = r \)

Arrow \( (U, V) \to (U', V') \):
- \( A \in S \) s.t. \( \ker(A) = U \), \( \text{Im}(A) = V' \)
Properties of $\mathcal{C}(S)$:

- Each non-trivial SCC is a groupoid.
Properties of $\mathcal{C}(S)$:

- Each non-trivial SCC is a groupoid.
- The number of non-trivial SCCs is at most $\binom{n}{r}$.
Properties of $\mathcal{C}(S)$:

- Each non-trivial SCC is a groupoid.
- The number of non-trivial SCCs is at most $\binom{n}{r}$.
- The depth of the dag of SCCs is at most $2\binom{n}{r}^2$. 

To obtain $S$ we generalise the algorithm Derken, Jeandel, and Koiran from finitely generated groups to constructibly generated groupoids.
Properties of \( \mathcal{C}(S) \):

- Each non-trivial SCC is a groupoid.
- The number of non-trivial SCCs is at most \( \binom{n}{r} \).
- The depth of the dag of SCCs is at most \( 2 \binom{n}{r}^2 \).
- Every two objects are connected by a path of arrows from \( A \) of length at most \( \binom{n}{r} + 1 \).
Properties of $\mathcal{C}(S)$:

- Each non-trivial SCC is a groupoid.
- The number of non-trivial SCCs is at most $\binom{n}{r}$.
- The depth of the dag of SCCs is at most $2\left(\binom{n}{r}\right)^2$.
- Every two objects are connected by a path of arrows from $A$ of length at most $\binom{n}{r} + 1$.

To obtain $S$ we generalise the algorithm Derken, Jeandel, and Koiran from finitely generated groups to constructibly generated groupoids.
Theorem

Given a finite set of rational square matrices of the same dimension, we can compute the Zariski closure of the semigroup that they generate.

Corollary

Given an affine program, we can compute for each location the ideal of all polynomial relations that hold at that location.