Instituto de Computação
Universidade Estadual de Campinas

Métodos Formais Algébricos para Geração de Invariantes

Rachid Rebiha

Este exemplar corresponde à redação final da Tese devidamente corrigida e defendida por Rachid Rebiha e aprovada pela Banca Examinadora.

Campinas, 12 de agosto de 2011.

[illegible]

Prof. Dr. Arnaldo Vieira Moura
Instituto de Computação UNICAMP,
Universidade Estadual de Campinas
(Orientador)

Tese apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação.

FICHA CATALOGRÁFICA ELABORADA POR
MARIA FABIANA BEZERRA MÜLLER - CRB8/6162
BIBLIOTECA DO INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E
COMPUTAÇÃO CIENTÍFICA - UNICAMP

Rebiha, Rachid, 1977-
R242m Métodos formais algébricos para geração de invariantes / Rachid Rebiha. – Campinas, SP: [s.n.], 2011.

Orientador: Arnaldo Vieira Moura.
Tese (doutorado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Métodos formais (Computação). 2. Sistemas híbridos. 3. Computadores – Medidas de segurança. 4. Vírus de computador. 5. Crime por computador. I. Moura, Arnaldo Vieira, 1950-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações para Biblioteca Digital

Título em inglês: Algebraic formal methods for invariant generation
Palavras-chave em Inglês:
Formal methods (Computer science)
Hybrid systems
Computer security
Computer viruses
Computer crimes
Área de concentração: Ciência da Computação
Titulação: Doutor em Ciência da Computação
Banca examinadora:
Arnaldo Vieira Moura [Orientador]
Ricardo Dahab
Julio César López Hernández
Arnaldo Mandel
Edward Hermann Haeusler
Data da defesa: 12-08-2011
Programa de Pós Graduação: Ciência da Computação

TERMO DE APROVAÇÃO

Tese Defendida e Aprovada em 12 de agosto de 2011, pela Banca examinadora composta pelos Professores Doutores:

[illegible]

Prof. Dr. Arnaldo Mandel
IME / USP

[illegible]

Prof. Dr. Edward Hermann Haeusler
Departamento de Informática / PUC-RIO

[illegible]

Prof. Dr. Julio César López Hernández
IC / UNICAMP

[illegible]

Prof. Dr. Ricardo Dahab
IC / UNICAMP

[illegible]

Prof. Dr. Arnaldo Vieira Moura
IC / UNICAMP

Instituto de Computação
Universidade Estadual de Campinas

Métodos Formais Algébricos para Geração de Invariantes

Rachid Rebiha

Agosto de 2011

Resumo

É bem sabido que a automação e a eficácia de métodos de verificação formal de softwares, sistemas embarcados ou sistemas híbridos, depende da facilidade com que invariantes precisas possam ser geradas automaticamente a partir do código fonte.

Uma invariante é uma propriedade, especificada sobre um local específico do código fonte, e que sempre se verifica a cada execução de um sistema. Apesar dos progressos enormes ao longo dos anos, o problema da geração de invariantes ainda está em aberto para tanto programas não-lineares discretos, como para sistemas não-lineares híbridos.

Nesta tese, primeiramente, apresentamos novos métodos computacionais que podem automatizar a descoberta e o fortalecimento de relações não-lineares entre as variáveis de um programa que contém laços não-lineares, ou seja, programas que exibem relações polinomiais multivariadas e manipulações fracionárias. Além disso, a maioria dos sistemas de segurança críticos, tais como aviões, automóveis, produtos químicos, usinas de energia e sistemas biológicos, operam semanticamente como sistemas híbridos não-lineares. Nesse trabalho, apresentamos poderosos métodos computacionais que são capazes de gerar bases de ideais polinomiais de invariantes não-lineares para sistemas híbridos não-lineares.

Em segundo lugar, apresentamos métodos pioneiros de verificação que automaticamente gerem bases de invariantes expressas por séries de potências multi-variáveis e por funções transcendentais. Discutimos, também, a sua convergência em sistemas híbridos que exibem modelos não lineares. Verificamos que as séries de potência geradas para invariantes são, muitas vezes, compostas pela expansão de algumas funções transcendentais bem conhecidas, tais como “log” e “exp”. Assim, apresentam uma forma analisável fechada que facilita o uso de invariantes na verificação de propriedades de segurança.

Para cada problema de geração de invariantes estabelecemos condições suficientes, muito gerais, que garantem a existência e permitem o cálculo dos ideais polinomiais para situações que não podem ser tratadas pelas abordagens de geração invariantes hoje conhecidas.

Finalmente, estendemos o domínio de aplicações, acessíveis através de métodos de geração de invariantes, para a área de segurança. Mais precisamente, fornecemos uma plataforma extensível baseada em invariantes pré-computadas que seriam usadas como assinaturas semânticas para análise de intrusos ("malwares") e deteção dos ataques de intrusões mais virulentos. Seguindo a concepção de tais plataformas, propomos sistemas de detecção de intrusão, usando modelos gerados automaticamente, onde as chamadas de sistema e de funções são vigiados pela avaliação de invariantes, pré-calculadas para denunciar qualquer desvio observado durante a execução da aplicação.

De modo abrangente, nesta tese, propomos a redução de problemas de geração de invariantes para problemas algébricos lineares. Ao reduzir os problemas de geração de invariante não-triviais de sistemas híbridos não-lineares para problemas algébricos lineares relacionados, somos capazes de ultrapassar as deficiências dos mais modernos métodos de geração de invariante hoje conhecidos permitindo, assim, a geração automática e eficiente de invariantes para programas e sistemas híbridos não lineares complexos. Tais métodos algébricos lineares apresentam complexidades computacionais significativamente inferiores àquelas exigidas pelos os fundamentos matemáticos das abordagens usadas hoje, tais como a computação de bases de Gröbner, a eliminação de quantificadores e decomposições cilíndricas algébricas.

Abstract

It is well-known that the automation and effectiveness of *formal software verification* of embedded or hybrid systems depends to the ease with which precise invariants can be automatically generated from source specifications.

An invariant is a property that holds true at a specific location in the specification code, whenever an execution reaches that location. Despite tremendous progress over the years, the problem of *invariant generation* remains very challenging for both *non-linear discrete programs*, as well as for *non-linear hybrid systems.*

In this thesis, we first present new computational methods that can automate the discovery and can strengthen interrelationships among the variables of a program that contains non-linear loops, that is, programs that display multivariate polynomial and fractional manipulations. Moreover, most of safety-critical systems such as aircraft, cars, chemicals, power plants and biological systems operate semantically as non-linear hybrid systems. In this work, we demonstrate powerful computational methods that can generate basis for non-linear invariant ideals of non-linear hybrid systems.

Secondly, we present the first verification methods that automatically generate basis for invariants expressed by *multivariate formal power series* and *transcendental functions.* We also discuss their convergence over hybrid systems that exhibit non linear models. The formal power series invariants generated are often composed by the expansion of some well-known transcendental functions *e.g. log* and *exp.* They also have an analysable closed-form which facilitates the use of the invariants when verifying safety properties.

For each invariant generation problem, we establish very general sufficient conditions that guarantee the existence and allow for the computation of invariant ideals for situations that can not be treated in the presently known invariant generation approaches.

Finally, we extend the domain of applications for invariant generation methods to encompass security problems. More precisely, we provide an extensible invariant-based platform for malware analysis and show how we can detect the most virulent intrusions attacks using these invariants. We propose to automatically generate invariants directly from the specified malware code in order to use them as semantic aware signatures, *i.e.* malware invariant, that would remain unchanged by most obfuscated techniques. Fol-

lowing the design of such platforms, we propose host-based intrusion detection systems, using automatically generated models where system calls are guarded by pre-computed invariants in order to report any deviation observed during the execution of the application.

In a broad sense, in this thesis, we propose to reduce the verification problem of invariant generation to algebraic problems. By reducing the problems of non-trivial non-linear invariant generation for programs and hybrid systems to related linear algebraic problems we are able to address various deficiencies of other state-of-the-art invariant generation methods, including the efficient treatment of complicated non-linear loop programs and non-linear hybrid systems. Such linear algebraic methods have much lower computational complexities than the mathematical foundations of previous approaches know today, which use techniques such as as Gröbner basis computation, quantifier elimination and cylindrical algebraic decomposition.

Agradecimentos

This thesis could not have been accomplished without the constant support of Prof. Arnaldo Vieira Moura and Prof. Fernando Pedone. I would like to express my deeply-felt thanks to Prof. Arnaldo Vieira Moura, for his enlightened advices and confidence in me. I could not have imagined having a better mentor for my Ph.D study. Under his thoughtful guidance and immense knowledge, he has taught me how good theoretical or applied scientific research are done. He provided me with a great research environment where we were free to create concepts, solve problems that matters in the fields we focused in, while developing interdisciplinary skills. I could count on his unflinching support at each steps of the way. I am grateful in every possible way and hope to keep up our collaboration in the future.

I am heartily thankful to Prof. Fernando Pedone, whose warm encouragement, advises and support from the initial to the final level of my Ph.D study enabled me to accomplished this thesis. Prof. Fernando Pedone is not only a Top Researcher in his fields, he is also a great research advisor who care about his students. I would like to show my gratitude to him for the encouragement and help he gave to me to enter in the joint doctoral school program for USI Faculty of Informatics University of Lugano and the Institute of Computing UNICAMP University of Campinas.

Besides my advisors, I would like to thank the rest of my thesis committee: Prof. Ricardo Dahab, Prof. Julio Cesar López Hernández, Prof. Arnaldo Mandel and Prof. Edward Hermann Haeusler. I am honored by their presence in the thesis committee.

Without any exception or omission, I would like to thank all professors from the Institute of Computing UNICAMP. Also, I would like to thank people working at the CPG secretariat.

At IC/UNICAMP I was surrounded by knowledgeable and very friendly fellow labmates. I would like to show my gratitude to all of them.

Few semesters after I started my Ph.D. study at USI, I became a Stanford Research International Fellow. I would like to thanks Dr. Steven Dawson, Dr. Hassen Saidi who also came visit me at USI after my stay at Stanford Research Institute, Dr. John Rushby and Dr. Ashish Tiwari.

Special thanks to Prof. Nadir Matringe who double-checked all the proofs in our papers.

I gratefully acknowledge the funding sources that made my Ph.D. work possible especially the "CNPq – Conselho Nacional de Desenvolvimento Científico e Tecnológico", a public organization of the Brazilian government under the Ministry of Science and Technology. Before I arrived at IC/UNICAMP, I was funded by USI during my stay in Lugano and by SRI International during my stay at Stanford Research Institute.

The *join doctoral school program for USI Faculty of Informatics University of Lugano Switzerland and the Institute of Computing UNICAMP University of Campinas Brazil* provide a productive and stimulating research context for Ph.D. candidates.

Before I started my Ph.D, I was assistant research at Research and Development - EDF, French Electricity and Energy - Ministry of Energy and Ministry of Research in formal methods, static and dynamic program analysis, verification and security. I would like to thank them as they let me led research Lab collaborations and real-world applications projects while I was first a post-graduate at Ecole Normal Superieure. I would like to thank Prof. Jean-Michel Ilie and Prof. Denis Poitrenaud from University of Paris 6 where I did one of my master thesis. I also thank all professors I had as undergraduate in the Department of Computer Science and in the Department of Mathematics, University of Orleans France and Imperial College Royal School of Science London especially Prof. Vo-Kac Kohan and Prof. Nirina Andrianarivelo.

Lastly, I would like to thank my familly. Special thanks to my daughter Sofia, the treasure of my life, who was born during the first year of my Ph.D. I hope that this Ph.D. would help me to take care of her. Many thanks to my wife Maria and to my mother for all the sacrifies she did for me. Thanks to my sisters and brother for their love and for supporting me spiritually throughout the years.

Sumário

Resumo **vii**

Abstract **ix**

Agradecimentos **xi**

1 Introduction **1**

1.1 Formal Methods 2

1.1.1 Model Checking 2

1.1.2 Abstraction 3

1.1.3 Theorem Proving 4

1.1.4 Invariant Generation 4

1.2 Contributions at a Glance 5

1.2.1 Contributions of Chapter 3 5

1.2.2 Contributions of Chapter 4 6

1.2.3 Contributions of Chapter 5 7

1.2.4 Contributions of Chapter 6 8

1.3 Chapter Outline 9

1.4 Publications and Seminars 9

1.4.1 Publications and Pre-prints 9

1.4.2 Invited Seminars, Seminars and Conference Talks 11

2 Preliminaries **13**

2.1 Algebra Definitions 14

2.2 Multivariate Polynomial Ring 14

2.3 Polynomial Ideals and Algebraic Varieties 16

2.4 Vector Space, Morphism and Eigenspace 18

3 Algebraic Invariant Generation **23**

3.1 Introduction 24

3.2 Inductive Algebraic Assertions 26
3.3 New Continuous consecution conditions 28
3.4 Discrete transition with affine systems 29
3.4.1 Generating λ-scale Invariants 30
3.4.2 Intersection with an initial hyperplane 34
3.4.3 Limit of constant-scale consecution 37
3.5 Handling algebraic discrete transition systems 37
3.5.1 T-scale invariant generation 37
3.5.2 General theory for discrete transition with polynomial systems . . 39
3.5.3 Generating invariant ideals with an initiation step 42
3.5.4 Running example 43
3.6 Obtaining optimal degree bounds for discrete transition systems 45
3.7 Invariant generation for discrete transitions with fractional systems 46
3.8 Branching conditions and nested loops 51
3.9 Chapter Discussions: Performances and Limitations 55

4 Generating Invariants for Non-linear Hybrid Systems 59
4.1 Introduction 59
4.2 Algebraic Hybrid Systems and Inductive Assertions 62
4.3 New continuous consecution conditions 65
4.4 Differential Invariant Generation 67
4.4.1 Morphisms for *strong*-scale differential consecution 67
4.4.2 Morphisms for *constant*-scale differential consecution 72
4.5 Handling non-linear differential systems 75
4.6 Obtaining optimal degree bounds 80
4.7 Examples and Experimental Results 83
4.8 Putting all together: Global invariants 87
4.9 Chapter Discussions: Performances and Limitations 89

5 Multivariate Formal Power Series 91
5.1 Introduction 92
5.2 Hybrid Systems and Multivariate Formal Power Series 93
5.2.1 Hybrid Systems 93
5.2.2 Multivariate Formal Power Series 94
5.3 New continuous consecutions 96
5.4 Reduction to linear algebra 96
5.5 Sufficient general existence conditions and the computation of invariants . 99
5.5.1 Sufficient general existence conditions 100
5.5.2 Inductive invariants and initial conditions 102

5.6 Triangularizable systems . . . 104
5.6.1 The matrices $M_{p-k,p}$ . . . 104
5.6.2 Resolution of the infinite system . . . 105
5.6.3 Convergence of the λ-invariant . . . 107
5.6.4 The case of eigenspaces with dimension 2 . . . 109
5.7 Running example . . . 110
5.7.1 The matrices $M_{p-k,p}$ . . . 110
5.7.2 Resolution of the infinite system . . . 111
5.7.3 Convergence of the invariant . . . 112
5.8 Transcendental invariants generation in closed-form . . . 113
5.9 Chapter Discussions: Performances and Limitations . . . 114

6 New domain of applications **117**
6.1 Introduction . . . 118
6.2 Malware Invariant Generation . . . 119
6.2.1 Malware and Virus Characterisation . . . 119
6.2.2 Identifying Malware Concealment Behaviours . . . 120
6.2.3 Obfuscation Strategies for Infection Mechanisms, Triggers and Payloads . . . 121
6.2.4 Malware Invariant as Semantic Aware Signature . . . 123
6.2.5 Automatic Generation of Malware Invariants . . . 124
6.3 Guarded Monitors For Host-based Intrusion Detection Systems . . . 128
6.3.1 Systems Calls Guarded by Invariants . . . 131
6.3.2 Construction of our model . . . 136
6.3.3 Guarded monitors precision and properties . . . 139
6.3.4 Vulnerability Auditing . . . 147
6.4 Chapter Discussions: Performances and Limitations . . . 148

7 Conclusions **149**
7.1 Final Conclusions . . . 149
7.2 Conclusões Finais . . . 151

A Proofs **153**
A.1 Proofs of Chapter 2 . . . 154
A.1.1 Proofs of Section 3.4.1 . . . 154
A.1.2 Proofs of Section 3.4.2 . . . 155
A.1.3 Proofs of Section 3.5.1 . . . 156
A.1.4 Proofs of Section 3.5.2 . . . 157
A.1.5 Proofs of Section 3.5.3 . . . 158

A.1.6 Proofs of Section 3.6 159
A.1.7 Proofs of Section 3.7 160
A.1.8 Proofs of Section 3.8 162
A.2 Proofs of Chapter 3 163
A.2.1 Proofs of Section 4.3 163
A.2.2 Proofs of Section 4.4.1 164
A.2.3 Proofs of Section 4.4.2 167
A.2.4 Proofs of Section 4.5 168
A.2.5 Proofs of Section 4.6 170
A.2.6 Proofs of Section 4.8 171
A.3 Proofs of Chapter 4 173
A.3.1 Proofs of Section 5.4 173
A.3.2 Proofs of Section 5.5 174
A.3.3 Proofs of Section 5.5.1 175
A.4 Proofs of Chapter 5 178
A.4.1 Proofs of Section 6.3.3 178
A.4.2 Proofs of Section 6.3.4 179

Lista de Tabelas

4.1 Linear algebraic problems and consecution approximations 87

4.2 Experimental results: Basis of invariant ideals obtained automatically by our prototype. All examples are treated in Section 4.4.1, Section 4.4.2, Section 4.5 and Section 4.7 88

6.1 Characterisation for malware types. 120

Lista de Figuras

3.1 Intersection between the conditional loop: $800 < (x-5)^2 + (y-5)^2 + (y_0 - 5)^2 < 1000$ and the invariant $y_0(1-y_0)x^2 + xy - x + y^2 - 2y + 1 = 0$ from the invariant ideal $(\{x^2, xy - x, y^2 - 2y + 1\})$ computed for the running example 10. 54

6.1 Quasi-static analysis framework 126
6.2 New formal methods to verify if the considered systems contains common behavior with another specified systems 127
6.3 Formal Signature: Malware Invariant and Malware PDS Generation. 128
6.4 Formal Malware Intrusion Detector System 129
6.5 Process for producing the final global monitor 140
6.6 Non-control flow attacks detected by our methods. 147

Capítulo 1

Introduction

The concepts exposed in this chapter are also introduced in our articles [80, 79, 83, 88, 84, 81, 85, 86, 93, 108].

Abstract: In this Chapter, we discuss key research areas that are at the heart of this dissertations and its contributions. The Chapter is organized as follows. We start with an overview of formal methods in Section 1.1. In Section 1.2.1, we summarize the original main contributions of this dissertation, and outline its organization in Section 1.3. Finally, we list our publications and the seminars associated to the results of this dissertation in Section 1.4.

1.1 Formal Methods

Formal methods aim at modeling and analysing systems using methods derived from, or defined by, underlying mathematically-precise concepts and their associated algorithmic foundations. By modeling we mean building specifications expressed in a particular logic, design or code. By analysing we mean the verification, or falsification, of system properties specified in an appropriate formal system.

Formal methods research aims at discovering mathematical techniques and developing their associated algorithms to establish the *correctness* of software, hardware, concurrent systems, embedded systems or hybrid systems, i.e. to prove that the considered systems are faithful to their specification. On large or infinite systems, or even systems with a huge or infinite numbers of reachable states, establishing *total* correctness is usually not practically possible. That is why we narrow our interest to safety and liveness properties that any well behaved engineered systems must guarantee. For instance, by using *static program analysis*, one could prove a software free of defects, such as buffer overflow or segmentation fault, which are safety properties, or non-termination, which is a liveness property.

In this sense, **static analysis** is used to generate *invariant properties*, which are assertions that hold true at a specific location on every possible run of the system. Thus static analysis can provide provable guarantees that even the most exhaustive and rigorous testing methods could not attain. Next, we discuss some proeminent techniques that proved useful in program verification.

1.1.1 Model Checking

Model checking [106, 29] is a verification technique for finite state systems. All states, together with all possible interactions in all possible runs of the system, are exhaustively enumerated. This could lead to huge structures. The main advantage of this method is that these structures can be build in memory *automatically*, from specifications in higher level languages. Several properties could then be algorithmically checked in this *state space*, by automatically checking correctness conditions at each state. In practice, we would *explicitly* explore all the state space to see if a specified unsafe property holds, that is, a "bad" reachable state exists. Such techniques have been used to prove correctness of hardware designs.

But this methods face the "state space explosion" problem, when it is not possible to hold all the state space even in a huge memory. As an alternative, one could use *symbolic* representations.

Symbolic model checking [91] comprise model checking techniques that use *symbolic* representations of sets of states. For instance, one could use logic formulae to represent a set of states. In this way, one could represent the set of reachable states in a very concise way. Moreover, binary decision diagrams (BDDs) [20] provide a very concise way to represent Boolean expressions and so can be used to economically represent sets of states in memory. We could also symbolically, represent infinite set of states by using, for example, semi-linear forms written as Presburger formulae. But even with these “nice” states space representations one could not completely deal with truly *infinite* systems.

Bounded model checking [13] is a model checking technique that exhaustively, or symbolically, analyses finite instances of infinite state systems. Any assertions that hold in a finite instance will hold on an infinite, i.e. more concrete instance. The basic idea in bounded model checking is to search for a counterexample in executions whose length is bounded by some integer k. If no defect is found then one increases k until either a defect is found, or the problem becomes intractable, or some pre-known upper bound is reached.

Model checking is often also used in falsification tests, i.e. for finding logical errors, rather than in verification, i.e. proving that errors do not exist. Here, in order to deal with infinite state systems, one needs to define a suitable *abstraction* of the systems regarding the properties one is looking for.

1.1.2 Abstraction

Abstraction [35] is commonly used in formal methods in order to achieve termination in the verification process for infinite systems, thus implying a trade off between termination of the verification process and completeness. Abstraction techniques first simplify the system in order to perform easier proof steps and later transfer the result back to the concrete system.

Abstract interpretation [35, 36] approaches depend on iteration and fixed point computations. Basically, it performs an approximate symbolic execution of a program or system until an assertion is reached that will remain unchanged along further executions of the program. However, in order to guarantee termination, the method introduces imprecision by the use of extrapolation operators called widening or narrowing. These operators often cause the technique to produce a too coarse abstraction that gives rise to weak invariants. Moreover, it requires manual intervention, as abstraction operations have to be provided and proved correct. Also, the right widening and narrowing operators have to be manually provided, which becomes a key challenge for abstract interpretation based techniques.

Predicate abstraction [59] requires a given set of abstraction predicates and an infinite state systems. It returns an “abstract state graph”, in the form of a conservative finite state abstraction. Every execution in the concrete system has a corresponding execution in the abstract system. First the abstract version of the safety property is model-checked in the abstract system. If the property holds in the abstract system, then it holds in the concrete system too. Otherwise, an abstract counter-example trace is generated [33]. But it is not guaranteed that there is a concrete counter-example associated with the abstract one. It is possible that this abstract counter-example is a spurious one due to a too coarse abstraction, because the set of abstraction predicates induced a too coarse abstraction. On the other hand, if there is a concrete counter-example corresponding to the abstract trace, then we have generated the trace of a real defect in the original design. One can then automatically analyze the counter-example to find a real defect in the system, or one can refine the abstraction by adding new discovered predicates [117, 28, 116]. These steps can be done using *theorem provers* [100, 101, 89] or other decision procedures such as those stemming from *sat modulo theory* [43, 44].

1.1.3 Theorem Proving

Theorem proving [89, 101] is an interactive verification technique where the correctness condition of the system is written as a theorem in a fixed theory. A theorem is then proved by interleaving automatic and manual interventions, using inference rules of the theory, its axiomatization, and other proved deductions, i.e. previous lemmas. Theorem proving has all the methods of logic and mathematics at its disposal, which makes theorem proving tools very powerful. However, the complexity of the problems that can be addressed by current theorem provers is limited by the fact that they require manual intervention. Theorem provers have seen trendemous progress, and some recent techniques propose model checking frameworks using automated deductions, as in sequent analysis [7].

1.1.4 Invariant Generation

Safety properties can be proved by *induction* techniques in infinite state systems [78]. An *inductive invariant* must hold in the initial state of the system and every possible transition must preserve it. The former is known as the initiation condition and the latter is called the consecution condition. Actually, the verification problem of safety properties can be reduced to the problem of invariant generation. In other words, if it holds in a given state then it continues to hold in all of its successor states. Let φ_P be the desired property and denote by φ_{Inv} the inductive invariant obtained. If $\varphi_{Inv} \Rightarrow \varphi_P$ holds then the proof of φ_P is complete.

Automated inductive invariant generation is the essential step in proving safety properties. For example when proving that an application is free of bugs like division by zero, outbounds of arrays, buffer overflows, null pointer de-referentiation, and many others. Or in proving liveness properties such as progress and termination.

We know that the weakest precondition method [41, 50] and the Floyd-Hoare [50, 66] inductive assertion technique require loop invariants to establish total correctness. In order to be completely automatic these methods require the use of invariant generation techniques. Also, ranking function techniques [78] depend on automated invariant generation methods in order to prove termination. We could list here many verification approaches that are only practical depending on the easy with which invariant can be automatically generated. Verification diagrams [18] is another example.

We can separate invariant generation methods in two main classes. We have methods that are *goal-oriented* [71, 54, 130, 113] and follow a *top-down* approach. These approaches start with a candidate potential invariant which can be seen as a target property that implies the properties we want to prove. On the other hand, we find methods which generate invariants directly from the program code. These approaches [70, 35, 37, 14, 126] are *bottom-up* techniques.

1.2 Contributions at a Glance

We look for invariants that strengthen what we wish to prove, and so allow us to establish the desired property. We can summarize our research achievements as follows.

1.2.1 Contributions of Chapter 3

We describe new computational methods [80, 79, 84] for generating basis of non-linear loop invariants ideals that can automate the discovery and strengthening of non-linear interrelationships among the variables of a program containing non-linear loops.

- We do not need to start with candidate invariants that generate intractable solving problems. Instead, we show that the preconditions for discrete transitions can be viewed as morphisms over a vector space of degree bounded by polynomials which can, thus, be suitably represented by matrices.

- We introduce a more general form for approximating consecution, called fraction and polynomial consecution. As far as we know, our methods are the first invariant generation methods that handle multivariate fractional system. The new relaxed consecution requirements are also encoded as morphisms, represented by matrices with terms that are the unknown coefficients used to approximate the consecution conditions.

- We succeeded in reducing the non-linear loop invariant generation problem to the computation of eigenspaces of specific morphisms.

- Such linear algebraic methods have lower complexities than the mathematical foundations of the previous approaches based on fixed point computations or the constraint-based approaches. Moreover, our methods do not require computation of Gröbner bases, quantifier elimination, cylindrical algebraic decomposition, or direct resolution of (semi-)algebraic systems, and do not depend on abstraction operators.

- Our methods generate the basis of an ideal where each of its elements is an inductive non-trivial invariant. In other words, instead of generating one invariant at a time, a huge (infinite) set of invariants can be generated using the computed basis.

- We provide general sufficient conditions guaranteeing the existence and allowing the computation of non-trivial non-linear loop invariant ideals.

- Also, we incorporate a strategy that attains optimal degree bounds for the candidate degree of invariants. We also note that our existence results and our methods can be reused in other approach in order to reduce their complexity, since they can reduce the number of Gröbner basis computations or quantifier eliminations, for example.

1.2.2 Contributions of Chapter 4

Hybrid systems [63, 4] exhibit both discrete and continuous behaviors, as one often finds when modeling digital system embedded in analog environments. The analysis of hybrid systems has been one of the main challenges for the formal verification community for several decades. In fact, most safety-critical systems like aircraft, automobiles, chemical and nuclear power plants, and biological systems operate semantically as non-linear hybrid systems. As such, they can only be adequately modeled by means of non linear arithmetic over the real numbers, involving multivariate polynomials and fractional or transcendental functions.

Regarding hybrid systems:

- We demonstrate powerful algorithms [83, 88, 84, 81], relying on linear algebraic methods, capable of computing basis for ideals of non-trivial invariants for non-linear hybrid systems.

- We reduce the non-trivial invariant generation problem to the computation of associated eigenspaces or nullspaces by encoding consecution requirements as specific morphisms represented by matrices.

- Our methods display lower complexities than the mathematical foundations of previous approaches based on fixed point computation, as well as the present constraint-based approaches and other approaches that use Gröbner basis calculations, Syzygy calculations or quantifier elimination.

- We handle non-linear hybrid systems, extended with parameters and variables that are functions of time. We note that the latter conditions are still not treated by other state-of-the-art invariant generation methods.

- We establish general sufficient conditions guaranteeing the existence and allowing the computation of invariant ideals for situations not treated by other modern invariant generation approaches.

- Our algorithm incorporates a strategy for estimating optimal degree bounds for candidate invariants, thus being able to compute basis for ideals of non-trivial non-linear invariants.

1.2.3 Contributions of Chapter 5

In order to verify safety properties expressed using transcendental functions, and to reason symbolically about formal power series, it is necessary to be able to first generate formal power series invariants.

We present the first verification methods [85, 87] that automatically generate basis of invariants expressed by *multivariate formal power series* and *transcendental functions*, while dealing with non linear continuous models present in many critical hybrid and embedded systems. The problem of synthesizing power series invariants and the results are clearly novel and significant.

In this Chapter:

- We introduce a more general approximation of consecution, dealing with assertions expressed by multivariate formal power series. We show that the preconditions for discrete transitions and the Lie-derivatives for continuous evolution can be viewed as morphisms and suitably represented by matrices.

- In this way, we reduce the invariant generation problem to linear algebraic matrix manipulations. We present an analysis of these matrices.

- The formal power series invariants generated are often composed by the expansion of some well-known transcendental functions, like *log* and *exp* and has an analysable closed-form, thus facilitating the use of the invariants to verify safety properties.

- We also discuss their convergence over hybrid systems that exhibit non linear models. To our knowledge, there are no other known methods that generate this type of invariants or that can deal with this type of systems. We provide resolution and convergence analysis for techniques that generate non trivial bases of provable multivariate formal power series and generate transcendental invariants for each local continuous evolution rules.

- Here again we succeed in obtaining very general sufficient conditions. Moreover, our methods are efficient since we use linear algebraic methods.

- The contribution is significant as it provides invariants that can be used to prove safety properties which also exhibit formal power series expressions or transcendental functions.

- Mathematically, we develop very general sufficient conditions allowing for the existence and computation of solutions defined by convergent formal power series for multivariate polynomial differential systems. In order to achieve this goal we develop new methods, in the spirit of Boularas et al. [17].

1.2.4 Contributions of Chapter 6

We present the *theoretical basis* [93, 108, 95] for the design of static and dynamic analysis platforms that can exhibit a suitable architecture for automatic in-depth malware analysis and detection. We provide an extensible invariant-based formal platform for malware analysis and detection. We show that invariant generation methods can be used to detect and identify malware.

More specifically, in this Chapter:

- We propose to automatically generate invariants directly from the specified malware code. We then show how to use it as *semantic aware signatures*, that we call *malware-invariant*, and we indicate how these invariants remain unchanged in most of the *obfuscated* version of malwares. These invariants could concisely capture the semantic of the malicious behavior of this family of viruses and could be, then, associated to few semantic aware signatures.

- Following the design of such platforms, we propose *host-based intrusion detection systems*, using automatically generated models where system calls are guarded by pre-computed invariants in order to report any deviation observed during the execution of the application.

- Given the precision of our generated models, we are able to detect the most virulent attacks such as mimicry attacks and non-data-control flow attacks. we also provide techniques for the detection of logic bugs and vulnerabilities in the application.

- Any intrusion or malware detection system analysis will be strongly re-enforced by the presence of these pre-computed invariants and will be weakened by their absence. Our platform is flexible, as any invariant generation method could be incorporated. In other words, it is an open architecture, where any invariant generation tool can be connected into a tool bus where invariants, expressed in different logics, will help in the identification of malicious behavior or in the construction of a precision intrusion detection system.

1.3 Chapter Outline

- In Chapter 1.1 we provide an overview of formal methods. We present standard definitions and the mathematical concepts that will used. Then we summarize, now more precisely, our advancements and achievements.
- In Chapter 2 we present a preliminaries chapter that introduces all the linear algebra definitions and needed notations that are used throughout the subsequent chapters.
- In Chapter 3 we develop our invariant generation methods for programs. We generates basis of non-trivial non-linear loop invariants.
- Then, in Chapter 4 we present our computational methods for invariant generation of non-linear hybrid systems using an extension of a linear algebraic approach.
- In Chapter 5 we present the first multivariate formal power series and transcendental invariants generation for non-linear hybrid systems.
- In Chapter 6 we discuss our extensible invariant-based platforms for malware analysis, and we show how we detect the most virulent intrusions attacks using these invariants.
- Finally, in Chapter 7 we present our conclusions.
- The Appendix contains a collection of proofs for all the theorems, lemmas and corollaries stated in this thesis.

Chapters 3, 4, and 5 address the problem of invariants generation for different systems. Chapter 6 discusses about new domains of applications, forming an independent part that does not necessarily use only the invariants generated in Chapters 3, 4, and 5, but any invariants generated by any other technique as well.

1.4 Publications and Seminars

1.4.1 Publications and Pre-prints

- [84] Generating Invariants for Non-linear Hybrid Systems by Linear Algebraic Methods. 17th Int. Static Analysis Symposium, SAS2010, Lecture Notes in Computer

Science (LNCS).

- [80] Endomorphisms for Non-trivial Non-Linear Loop Invariant Generation, 5th Int. Conf. Theoretical Aspects of Computing ICTAC2008, Lecture Notes in Computer Science (LNCS).

- [79] Endomorphism for Non-Trivial Semi-Algebraic Loop Invariant Generation, Technical Report, IC Unicamp, 2008.

- [88] Morphisms for Analysis of Hybrid Systems. ACM/IEEE Cyber-Physical Systems CPSWeek'09, Second International Workshop on Numerical Software Verification. NSV2009, Verification of Cyber-Physical Software Systems. San Francisco, CA, USA, 2009.

- [83] Morphisms for Non-trivial Non-linear Invariant Generation for Algebraic Hybrid Systems, 12th Int. Conf. Hybrid Systems: Computation and Control (HSCC2009), Lecture Notes in Computer Science (LNCS), 2010.

- [81] Morphisms for Non-trivial Non-Linear Invariant Generation for Algebraic Hybrid Systems. Technical Report, IC Unicamp, 2008.

- [82], Generating multivariate formal power series for hybrid systems. Technical Report, IC Unicamp, 2009.

- [85], Multivariate Formal Power Series and Transcendental Invariants Generation for Non-linear Differential and Hybrid Systems. Under submission.

- [94] Automated Malware Invariant Generation. Invited submission to the International Journal of Forensic Computer Science, 2010.

- [27] An Ant Colony Verification Algorithm. 7th. IEEE. International Conference on Intelligent Systems Design and Applications. ISDA 2007.

- Section 6.2 was published at [93] Formal Methods for Forensic Computer Science: Automated Malware Invariant Generation, 6th International Conference on Forensic Computer Science, ICoFSC'2009 and 7th International Conference on Cyber Computer Science, best paper award, ICCYBER'2009.

- [108] Quasi-static binary analysis: guarded model for intrusion detection, Technical Report, USI Lugano – SRI International, 2006.

1.4.2 Invited Seminars, Seminars and Conference Talks

- Institute of Computing UNICAMP, University of Campinas, São Paulo Brazil. "Algebraic Formal Methods I : Invariant Generation for Program Verification and Security".
- 5th Int. Conf. Theoretical Aspects of Computing ICTAC2008. "Endomorphisms for non-trivial non-linear loop invariant generation".
- "Morphisms for Analysis of Hybrid Systems". ACM/IEEE Cyber-Physical Systems CPSWeek'09, Second International Workshop on Numerical Software Verification. NSV2009, Verification of Cyber-Physical Software Systems.
- "Morphisms for Non-trivial Non-linear Invariant Generation for Algebraic Hybrid Systems", 12th Int. Conf. Hybrid Systems: Computation and Control, 2010.
- "Generating Invariants for Non-linear Hybrid Systems by Linear Algebraic Methods", 17th Int. Static Analysis Symposium, 2010.
- "Invariant Generation for Host-Based Intrusion Detection". USI Faculty of Informatics, University of Lugano, Switzerland, 2007.
- "Quasi-Static Binary analysis: Guarded Model for Host-Based Intrusion Detection". Stanford Research Institute, 2006.
- "Program and Memory Heap Verification by Infinite Model Checking: Segmentation Fault and Memory Leak Checking with Recursive Data Structures". Stanford Research Institute, 2006.
- "An Extensible LTL Model Checking Library and Transition-based Generalized Büchi Automata", Stanford Research Institute, 2006.
- "Automatic Memory Heap Verification", ETH, University of Zurich, Switzerland, 2006.
- "Formal Verification with CTL* and (Propositional-) Fixed point Theory", USI Faculty of Informatics, University of Lugano, Switzerland, 2006.
- "Automated Malware Invariant Generation", 6th International Conference on Forensic Computer Science, ICoFSC'2009 and 7th International Conference on Cyber Computer Science, ICCYBER'2009.

Capítulo 2

Preliminaries

The concepts and results exposed are also mentioned in our articles [80, 79, 83, 88, 84, 81, 85, 86, 93, 108].

Abstract: This is a preliminary chapter that introduces all algebra definitions that are used throughout the development of our formal methods and their associated algorithms. We provide the background material on abstract algebra, algebraic geometry and linear algebra that are at the heart of the main object of our considerations and at our theoretical and algorithmic approaches to it.

2.1 Algebra Definitions

In Section 2.2 we give some important elements of abstract algebra such as the definition of Multivariate Polynomial. In Section 2.3 we introduce important elements of algebraic geometry such as ideals. Finally, in Section 2.4 we provide notions of linear algebra that are central in the developments of our approaches.

2.2 Multivariate Polynomial Ring

Here we introduce the needed background material on abstract algebra.

A *field* is an algebraic structure, that is a set satisfying certain filed axioms for both addition and multiplication. It is also a commutative division algebra.

Definition 1. *Any set $\mathbb{K}$ of elements together with two binary operators $+$ and $*$ (commonly interpreted as addition and multiplication, respectively) is a* field *if and only if it satisfies the following field axioms for both $+$ and $*$:*

- $\mathbb{K}$ *is closed under addition and multiplication:*

$$\forall r, s \in \mathbb{K},\ r+s, r*s \in \mathbb{K}.$$

- *Associativity of addition and multiplication:*

$$\forall r, s, t \in \mathbb{K},\ r+(s+q) = (r+s)+q \text{ and } r*(s*q) = (r*s)*q.$$

- *Commutativity of addition and multiplication:*

$$\forall r, s \in \mathbb{K},\ r+s = s+r \text{ and } r*s = s*r.$$

- *Existence of additive and multiplicative identity and inverses:*
  - *There are $\alpha, \epsilon \in \mathbb{K}$ such that for all $r \in \mathbb{K}$, $\alpha + r = r$ and $\epsilon * r = r$.*
  - *For all $r \in \mathbb{K}$ there exists an element $-r \in \mathbb{K}$ such that $r+(-r) = \alpha$. Similarly, for all $r \in \mathbb{K}$ there exists an element $r^{-1} \in \mathbb{K}_{\backslash\{\epsilon\}}$ such that $r*(r^{-1}) = \epsilon$.*
  - *Distributivity of multiplication over addition:*

    For all $r, s, t \in \mathbb{K}$, $r*(s+q) = (r*s)+(r*q)$ and $(s+q)*r = (s*r)+(q*r)$.

□

Let $\{X_1, .., X_n\}$ be a set of variables. An expression of the from

$$X_1^{k_1} X_2^{k_2} \cdots X_n^{k_n},$$

where each k_i are positive or null integers, is called a *power-product* over the variables $\{X_1, .., X_n\}$. We denote by $\mathbb{P}_{X_1,...,X_n}$ the set of power-products over the variables $\{X_1, .., X_n\}$.

A *monomial* is an expression of the form $a \cdot m$ where a is a constant in a field $\mathbb{K}$ and $m \in \mathbb{P}_{X_1,...,X_n}$. We denote by $\mathbb{M}_{X_1,...,X_n}$ the set of all monomials over the variables $\{X_1, .., X_n\}$.

Definition 2. *A (multivariate) polynomial over $\{X_1, .., X_n\}$ is a finite sum of monomials in $\mathbb{M}_{X_1,...,X_n}$. In other words, an n multivariate polynomial Q over $\{X_1, .., X_n\}$ is an expression of the form*

$$Q = \sum_{i_1,...,i_n} a_{i_1,...,i_n} X_1^{i_1} X_2^{i_2} \cdots X_n^{i_n}.$$

□

In this thesis, we refer several times to the notions of *polynomial and fractional rings*. We first give the definition of a ring.

Definition 3. *Any set R of elements together with two binary operators $+$ and $*$ (commonly called addition and multiplication, respectively) is a* ring *if and only if the following conditions hold:*

- *R is closed under addition and multiplication: for all $r, s \in R$, $r + s, r * s \in R$.*
- *Associativity of addition and multiplication:*

  $$\text{For all } r, s, t \in R,\ r + (s + q) = (r + s) + q \text{ and } r * (s * q) = (r * s) * q.$$
- *Commutativity of addition: for all $r, s \in R$, $r + s = s + r$.*
- *Existence of additive and multiplicative identities: there are $\alpha, \epsilon \in R$ such that for all $r \in R$, $\alpha + r = r$ and $\epsilon * r = r$.*
- *Existence of additive inverse: for all $r \in \mathbb{K}$ there exists an element $-r \in \mathbb{K}$ such that $r + (-r) = \alpha$.*
- *Distributivity of multiplication over addition:*

  $$\forall r, s, t \in \mathbb{K},\ r * (s + q) = (r * s) + (r * q) \text{ and } (s + q) * r = (s * r) + (q * r).$$

□

The sum and product of polynomials are again polynomials, and it is easy to see in a purely algebraic setting that the collection of polynomials forms a commutative ring (a ring satisfying the multiplicative commutativity axiom). The set of polynomials on the variables $\{X_1, .., X_n\}$, whose coefficients are drawn from a field $\mathbb{K}$ is the ring of polynomials over the variables $\{X_1, .., X_n\}$ denoted $\mathbb{K}[X_1, .., X_n]$.

2.3 Polynomial Ideals and Algebraic Varieties

Let $\mathbb{K}[X_1, .., X_n]$ be a multivariate polynomial ring over the set of variables $\{X_1, .., X_n\}$. Specializing the definition of an ideal to $\mathbb{K}[X_1, .., X_n]$, we have the following.

Definition 4. *An ideal is any set $I \subseteq \mathbb{K}[X_1, .., X_n]$ such that*

- *it is closed under addition. In other words, if $P, Q \in I$ then $P + Q \in I$;*
- *it is closed under multiplication by any element in $\mathbb{K}[X_1, .., X_n]$, i.e.*

$$\textit{if } P \in I \textit{ and } Q \in K[X_1, .., X_n] \textit{ then } PQ \in I;$$

- *it includes the null polynomial,* i.e. $0_{\mathbb{K}[X_1,..,X_n]} \in I$.

□

In the following definition we propose a very important polynomial ideal for our purposes and the development of our methods.

Definition 5. *Let $E \subseteq \mathbb{K}[X_1, .., X_n]$ be a set of polynomials.*

- *The* ideal generated by E *is the set of finite sums*

$$(E) = \left\{ \sum_{i=1}^{k} P_i Q_i \mid P_i \in \mathbb{K}[X_1, \ldots, X_n], Q_i \in E, k \geq 1 \right\}.$$

- *A set of polynomials E is said to be a* basis *of an ideal I if $I = (E)$.*

□

We say that an ideal is finitely generated if it is generated by a finite basis. By the Hilbert basis theorem, we know that $\mathbb{K}[X_1, .., X_n]$ is a *Noetherian* ring, i.e, all ideals included in $\mathbb{K}[X_1, .., X_n]$ are finitely generated as they have a *finite basis*. In Chapter 3, 4, and 5 we generate basis of ideals where each of its elements has a very powerful property.

An algebraic variety is the set of all common zeroes of a finite collection of polynomials. In other words, it is the zero set of a finite set of polynomials.

Definition 6. *Let $P_1, ..., P_m$ be a finite set of polynomials in $\mathbb{K}[X_1, .., X_n]$.*

- *An* algebraic assertion *is an assertion $\phi(X_1, .., X_n)$ of the following form:*

$$\bigwedge_i P_i(X_1, ..., X_n) = 0.$$

- *The* variety *defined by $P_1, ..., P_m$ is the set $\mathbb{V}(P_1, ..., P_m)$ such that:*

$$\mathbb{V}(P_1, ..., P_m) = \{(r_1, ..., r_n) \in \mathbb{K} \mid (P_1(r_1, ..., r_n) = 0) \wedge ... \wedge (P_m(r_1, ..., r_n) = 0)\}.$$

□

In Chapter 3 algebraic assertions will be formed by (non-linear) loop instructions.

To start connecting ideals and varieties we need to introduce one more specific ideal. The set of polynomials in $\mathbb{K}[X_1, .., X_n]$ that vanish in a given set $G \subset \mathbb{K}^n$ is an ideal. This set, denoted as $\mathbb{I}(G)$, is called the *vanishing ideal* of G and it is defined as follow:

$$\mathbb{I}(G) = \{P \in \mathbb{K}[X_1, .., X_n] \mid P(r_1, ..., r_n) = 0 \ \forall (r_1, ...r_n) \in G\}.$$

Now, let $P_1, ..., P_m$ be a finite set of polynomials in $\mathbb{K}[X_1, .., X_n]$. Looking at Definition 5 we know how to generate the ideal $(P_1, ..., P_m)$. On the other hand, we could look at the corresponding variety $\mathbb{V}(P_1, ..., P_m) \subset \mathbb{K}^n$. Moreover, we could form the corresponding vanishing ideal $\mathbb{I}(\mathbb{V}(P_1, ..., P_m))$. Now, a famous result by Hilbert, the *Hilbert's Nullstellensatz* show how these two ideals, $(P_1, ..., P_m)$ and $\mathbb{I}(\mathbb{V}(P_1, ..., P_m))$ are related to each other and in which case $(P_1, ..., P_m) = \mathbb{I}(\mathbb{V}(P_1, ..., P_m))$ holds. Below we briefly state the relevant direction of the theorem.

Let $\phi(X_1, .., X_n) \equiv (P_1(X_1, ..., X_n) = 0 \wedge \cdots \wedge P_m(X_1, ..., X_n) = 0)$ be an algebraic assertion with $P_1, ..., P_m \in \mathbb{K}[X_1, .., X_n]$. If $Q \in (P_1, ..., P_m)$ then we could write Q in the following form $Q = H_1P_1 + ... + H_mP_m$ where the H_is are in $\mathbb{K}[X_1, .., X_n]$. Let $(r_1, ..., r_n) \in \mathbb{K}^n$ be such that $\phi(r_1, .., r_n)$ holds. Then

$$Q(r_1, ..., r_n) = H_1(r_1, ..., r_n)P_1(r_1, ..., r_n) + ... + H_m(r_1, ..., r_n)P_m(r_1, ..., r_n) = 0,$$

since we have $P_1(r_1, ..., r_n) = 0, \ldots, P_m(r_1, ..., r_n) = 0$. Therefore we just proved that $\forall (r_1, .., r_n) \in \mathbb{K}^n$, $\phi(X_1, .., X_n) \Rightarrow (Q(r_1, ..., r_n) = 0)$, i.e., $\phi(X_1, .., X_n) \models (Q(X_1, ..., X_n) =$

0). In other words, we have shown that $(P_1, ..., P_m) \subset \mathbb{I}(\mathbb{V}(P_1, ..., P_m))$. This correspondence between ideals and varieties form a dual viewpoint (algebraic with ideals and geometric with varieties) that is very powerful. An algebraic assertion can form the basis of an ideal and an ideal could define a variety, formed by the set of common zeros of all its polynomials.

2.4 Vector Space, Morphism and Eigenspace

Here we define key notions that are central in the theoretical and algorithmic development of our methods.

We first define the notion of *vector space* formed by a collections of vectors that could be added together and multiplied by a scalar in a field $\mathbb{K}$. A vector space is a set closed under finite vector addition and scalar multiplication. Its formal definition is given below.

Definition 7. *A set V is a vector space over a field $\mathbb{K}$ if and only if the following conditions holds:*

- *Associativity of vector addition: for all $X, Y, Z \in V$, $X + (Y + Z) = (X + Y) + Z$,*
- *Associativity of scalar multiplication: for all $X \in V$ and $a, b \in \mathbb{K}$, $a * (b * X) = (a * b) * X$,*
- *Additive vector identity: there is a vector $0 \in V$ such that for all$X \in V$, $X + 0 = 0 + X = X$,*
- *Multiplicative scalar identity: for all $X \in V$, $1 * X = X$, where 1 is the multiplicative unit in $\mathbb{K}$,*
- *Existence of additive inverse: for all $X \in V$ there exists an element $-X \in V$ such that $X + (-X) = 0$,*
- *Commutativity of vector addition: for all $X, Y, Z \in V$, $X + Y = Y + X$,*
- *Distributivity of scalar addition: for all $X \in V$ and $a, b \in \mathbb{K}$, $(a+b)*X = a*X + b*X$,*
- *Distributivity of vector addition: for all$X, Y \in V$ and $a \in \mathbb{K}$, $a * (X + Y) = a * X + a * Y$.*

$\square$

A *morphism* is a function φ between two vector spaces E and F, denoted by $\varphi : E \to F$, such that the addition and multiplication are respected.Such morphism between vector space could aslo be called linear transformation in the litterature. We give the formal definition below.

Definition 8. *Let E and F be two vector spaces over a field $\mathbb{K}$. A morphism is a function $\varphi : E \to F$ such that:*

- *for all $P, Q \in E$, $\varphi(P + Q) = \varphi(P) + \varphi(Q)$,*
- *for all $P \in E$ and for all $\lambda \in \mathbb{K}$, $\varphi(\lambda * P) = \lambda * \varphi(P)$.*

□

A morphism from a vectorial space E to the same vector space E is called an *endomorphism.*

In Chapter 3, 4, and 5, we build several very important morphisms between vectorial spaces. For instance, we define the morphism $\mathscr{L}$, in Section 3.5.2, between $\mathbb{R}_r[X_1, .., X_n]$ and $\mathbb{R}_{rd}[X_1, .., X_n]$ where r and d are two integers and where $\mathbb{R}_r[X_1, .., X_n]$ and $\mathbb{R}_{rd}[X_1, .., X_n]$ refer to the set of polynomials of degree at most r and rd, respectively, over the real field $\mathbb{R}$. Let $T \in \mathbb{R}_d[X_1, .., X_n]$ be a specific polynomial, we define the following morphism for all $P \in \mathbb{R}_r[X_1, .., X_n]$:

$$\mathscr{L} : \left\{ \begin{array}{rcl} \mathbb{R}_r[X_1, \ldots, X_n] & \to & \mathbb{R}_{dr}[X_1, \ldots, X_n] \\ P & \mapsto & TP. \end{array} \right.$$

As we can see, $\mathscr{L}$ takes as input a polynomial $Q \in \mathbb{R}_r[X_1, \ldots, X_n]$ and returns a polynomial $\mathscr{L}(Q) = TQ$, where T was our fixed specific multiplicative polynomial.

The *Kernel* of a morphism $\varphi : E \to F$ gives the elements from the initial domain E that are mapped by φ to the additive identity element noted 0_F. Formalizing the definition of a Kernel of a vector space, we have the following.

Definition 9. *Let E and F be two vectorial spaces over a field $\mathbb{K}$ and consider a morphism $\varphi : E \to F$. The Kernel of φ, denoted by $Ker(\varphi)$, is defined by:*

$$Ker(\varphi) = \{P \in E \mid \varphi(P) = 0_F\},$$

where 0_F is the zero element, the additive identity of F. □

The Kernel of a morphism $\varphi : E \to F$ is a linear subspace of its initial domain E. The notions of Kernel remain central for Chapter 3, 4, and 5 as we succeed in reducing the respective invariants generation problems to the computation of basis which generates Kernels of specific morphism.

It is important to emphasize that all the morphisms build in Chapters 3, 4, and 5 are from domains of finite dimension to domains of finite dimension. Therefore, one can consider their matrix representation using the well-known canonical basis of their initial and final domains. Every time we define a morphism we present and build its matrix representation (giving descriptions for their step by step construction) using the well-known canonical basis of the initial and final domains of the considered morphism. In this way we reduce the problem to linear algebra every time. Here we adapt the definition of Kernel to the context of matrices.

Definition 10. *Let M be a $m \times n$ matrix where the terms are drawn in a field $\mathbb{K}$. The Kernel of M, also called its* nullspace*, and denoted by $Ker(M)$, is defined by:*

$$Ker(M) = \{V \in \mathbb{K}^n \mid M \cdot V = 0_{\mathbb{K}^m}\}.$$

□

In Chapter 3, 4, and 5, we need to compute the Kernel of specific matrices. In fact, when we deal with square matrices, these Kernels are *Eigenspaces.* We give the definitions of Eigenvalues, Eigenvectors, and Eigenspaces, below.

Definition 11. *Let M be a $n \times n$ square matrix with terms in $\mathbb{K}$. A nonzero vector $X \in \mathbb{K}$ is an eigenvector for M associated with eigenvalue λ (a scalar in $\mathbb{K}$), if the following condition holds:*

$$M \cdot X = \lambda X, \text{ i.e., } (M - \lambda I_n) \cdot X = 0,$$

where I_n is the $n \times n$ matrix of the identity morphism.

The nullspace of $(M - \lambda I_n)$ is the eigenspace *of M associated with eigenvalue λ.* □

Our approaches described in Chapter 3, 4, and 5, are based on linear algebraic notions such as the ones described in this subsection. In Chapter 3 and 4 we build specific morphisms, we construct their matrix representations, compute the Kernel of these matrices and interpret the results in term of polynomial ideals, called invariant ideals.

For example, assume that a morphism φ from $\mathbb{R}_2[x,y]$ to $\mathbb{R}_4[x,y]$ needs to be considered at some point of the execution of our methods for a given application. And let say that its associated matrix M_φ, computed using the basis $C_1 = (x^2, xy, y^2, x, y, 1)$ of $\mathbb{R}_2[x,y]$ and the basis

$$C_2 = (x^4, yx^3, y^2x^2, y^3x, y^4, x^3, x^2y, xy^2, y^3, x^2, xy, y^2, x, y, 1)$$

of $\mathbb{R}_4[x,y]$, is the one described below:

$$\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & -1 & 0 & 0 \\
0 & 0 & -2 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & -1 & 0 & 0 \\
0 & 0 & -1 & 0 & -1 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & -2 \\
0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}.$$

Then, the eigenspace is generated by the following computed basis

$$\{(1,0,0,0,0,0)^\top, (0,1,0,-1,0,0)^\top, (0,0,1,0,-2,1)^\top\}.$$

The vectors of the basis are interpreted in the canonical basis C_1 of $\mathbb{R}_2[x_1,x_2]$:

- the vector $(1,0,0,0,0,0)^\top$ is interpreted by the polynomial x^2,
- the vector $(0,1,0,-1,0,0)^\top$ is interpreted by the polynomial $xy - x$,
- the vector $(0,0,1,0,-2,1)^\top$ is interpreted by the polynomial $y^2 - 2y + 1$.

Then, the ideal generated by the basis $\{x^2,\ xy - x,\ y^2 - 2y + 1\}$ will be of great interest. In fact, we show in these chapters that these ideals are actually inductive invariant ideals.

In Subsection 3.6 and in Subsection 4.6 we refer to the notions of *matrix rank* and of *Singular Value Decomposition* (SVD). We just saw that we can calculate the eigenvalues and associated eigenspaces, and so the nullspace of a square matrix. When the considered matrix M is a rectangular matrix, we have similar linear algebraic objects. In that case, we can use its *Singular Value Decomposition* that provides an explicit representation of its rank and kernel by computing unitary matrices U and V, and a regular diagonal matrix S (with non negative real numbers in the diagonal), such that $M = USV$. The diagonal terms $S_{i,i}$ of the matrix S are the *singular values* of M. Basically, we obtain similar linear algebraic tools to compute the nullspace of a matrix M.

Definition 12. *The rank of a matrix is the dimension of the image of the matrix. In other words, it is the number of linearly independent columns or rows of the matrix. It is also the number of singular values of the matrix.* $\square$

Let M be a $m \times n$ matrix. We know that $rank(M) + Dim(Ker(M)) = n$ where $Dim(Ker(M))$ is the dimension of the nullspace of the matrix M.

To compute the basis of eigenspaces and nullspaces, we use well-known state-of-the-art algorithms, for example those that Mathematica provides.

More details on abstract algebra, algebraic geometry and linear algebra, emphasizing the computational aspects, can be found in the textbooks Dummit and Foote [42], Cox, Little, and O'Shea [38], Bochnak, Coste, and Roy [15] and in Land [76].

Capítulo 3

Algebraic Invariant Generation

The contributions presented in this chapter are also published in our articles [80, 79, 84].

Abstract: We present new computational methods that can automate the discovery and the strengthening of non-linear interrelationships among the variables of a program containing non-linear loops, that is, that give rise to multivariate polynomial and fractional manipulations. Our methods have lower complexities than the mathematical foundations of the previous approaches, which used Grobner basis computation, quantifier elimination or cylindrical algebraic decomposition. We show that the preconditions for discrete transitions can be viewed as morphisms over a vector space of degree bounded by polynomials. These morphisms can, thus, be suitably represented by matrices. We introduce fractional and polynomial consecution, more general forms for approximating consecution. As far as it is our knowledge, it is the first invariant generation methods that handle multivariate fractional loops.

The new relaxed consecution conditions are also encoded as morphisms represented by matrices. By doing so, we succeeded in reducing the non-linear loop invariant generation problem to the intersection between eigenspaces of specific endomorphisms and initial linear or algebraic constraints. We provide general sufficient conditions for the existence of non-trivial non-linear loop invariants. Our algorithm also incorporates a strategy to guess the degree bounds which allow for the generation of ideals of non-trivial invariants.

3.1 Introduction

We present a new method that addresses various deficiencies of state-of-the-art non-linear invariant generation methods. An invariant at a location of a program is an assertion true of any reachable program state associated to this location. We provide mathematical techniques and design efficient algorithms to automate the discovery and the strengthening of non-linear interrelationships among the variables of a program containing non-linear loops, which give rise to multivariate polynomial and fractional manipulations.

It is well-known that the automation and effectiveness of formal program verification depends on the ease with which invariants can be automatically generated. Actually, the verification problem of safety properties, such as no null pointer deferenciation, buffer overflows, memory leak or outbounds, and array accesses, can be reduced to the problem of invariant generation [78]. Invariants are also essential to prove and establish liveness properties such as progress or termination [78]. Furthermore, the standard techniques [78] for program verification use invariant assertion directly to prove program properties, or to provide lemmas that can be used to established other safety and liveness program properties. We look for invariants that strengthen what we wish to prove, and so allow us to establish the desired property. Also, they can provide precise over-approximations to the set of reachable states.

We know that the weakest precondition method [41, 50], the Floyd-Hoare [50, 66] inductive assertion technique, and the standard ranking functions technique [78], require loop invariants to establish correctness and to render the method completely automatic. Also, to establish termination verification, the standard ranking functions technique requires the automatic generation of invariants.

To generate loop invariants, one need to discover *inductive* assertions that hold at any steps of the loop. An *inductive* assertion also holds at the first time the loop location is reached, this is the initiation condition, and is preserved under every instructions that cycle back to the loop location, this being the consecution condition. If we choose transition systems as the representation model and automata as the computational model, we can say that the invariant holds in the initial state of the system (the initial condition) and that every possible transition preserves it (the consecution conditions). In other words, the invariant holds in any possible reachable state.

In the case of loops describing a linear system, *Farka's lemma* [122] can be used to encode the conditions for being a *linear* invariant. On the other hand, for *non-linear* invariants, the difficulty of automatic generation remains very challenging. By today known methods, they require a high number of Gröbner Bases computation [120], first-order quantifier elimination [131, 32] or cylindrical algebraic decomposition [24]. In E. Rodriguez-Carbonell and D. Kapur [109] forward propagation techniques use an abstract interpretation [36, 35] framework and Gröbner bases construction to compute invariants as fixed points of operations on ideals. Abstract interpretation introduces imprecision, and *widening* operators must be provided manually by the user in order to assure termination. A too coarse abstraction would limit these approaches to trivial invariant generation in the presence of non linear loops. L. Kovacs et al. [72, 73], attempt to generate invariants from a restricted class of (P-solvable) loops. Their methods use techniques from algebra and combinatorics, like Gröbner bases [67], variable elimination, algebraic dependencies and symbolic summation.

More recent approaches have been constraint-based [120, 109, 69, 110, 118, 114, 105]. In these cases, a candidate invariant with a fixed degree and unknown parametric coefficients, *i.e.*, a template form, is proposed as the target invariant to be generated. The conditions for invariance are then encoded, resulting in constraints on the unknown coefficients whose solutions yield invariants. One of the main advantage of such constraint-based approaches is that they are goal-oriented. The main challenge for these techniques remains in the fact that they still require a high number of Gröbner Bases [21] computations, first-order quantifier elimination [131, 32], cylindrical algebraic decomposition [24], or abstraction operators. And known algorithms for those problems are, at least, of double exponential complexity.

Despite tremendous progress over the years [120, 10, 109, 113, 24, 73, 72, 34, 96, 110, 114, 1, 104], the problem of loop invariant generation remains very challenging for non-linear discrete systems. In this chapter we present new methods for the automatic generation of loop invariants for non-linear systems. As will be seen, these methods give rise to more efficient algorithms, with much lower complexity in space and time.

We develop the new methods by first extending our previous work on non-linear non-trivial invariant generation for discrete programs with nested loops and conditional statements, [80, 84]. Further proof details and examples can be found in associated technical reports and articles [79, 80, 84].

The contributions of this chapter are summarized in Section 1.2.1.

In Section 3.2 we present ideals of polynomials and their possible interaction with inductive assertions. In Section 3.3 we introduce new consecution conditions, extended to fractional systems. In Section 3.4 we consider the case where the loop is linear. We present results for the existence of *non-trivial* invariants. We translate the problem in term of linear algebra, and we present a complete decision procedure for the automatic generation of *non-trivial* non-linear invariants. In Section 3.5 we extend our method to non-linear loops. In Section 3.6 we propose a strategy to obtain optimal degree bounds. In Section 3.7 we provide a complete generalization by considering loops describing multivariate fractional systems. And in Section 3.8 we show how to handle conditions and nested loops. We conclude our approach in Section 3.9.

3.2 Inductive Algebraic Assertions

For the definitions of field, multivariate polynomials, polynomial ring, we refer to Section 2.2 in Chapter 2.

The notions of ideal and their associated basis are detailed in definitions 4 and 5 from Section 2.3 in Chapter 2. For the definitions of varieties and algebraic assertions, we refer to definitions 6 from Section 2.3 in Chapter 2.

The linear algebraic notions of morphism and Kernel are central in this chapter and their definitions could be found in Section 2.4 in Chapter 2. Let $\mathbb{K}[X_1, .., X_n]$ be the ring of multivariate polynomials over the set of variables $\{X_1, .., X_n\}$.

In this Chapter, we will use the following notations:

- A primed symbol x' refers to the next state value of x, after a transition is taken.
- We denote by $\mathbb{R}_d[X_1, .., X_n]$ the ring of multivariate polynomials of degree at most d over the set of real variables $\{X_1, .., X_n\}$.
- We write $Vect(v_1, ..., v_n)$ for the vector space generated by the basis $v_1, ..., v_n$.
- We write $Ker(M)$ for the kernel of M and $Rank(M)$ for the rank of M.

We use *transition systems* as representation of imperative programs and *automata* as their computational models.

The contribution and novelty in our approach clearly set it apart from those in [120] as their constraint-based techniques are based on several Grobner Basis computations and on solving non linear problems for each location. Nevertheless, they introduce a useful formalism to treat programs loops, and we start from similar definitions for transitions systems, inductive invariants and consecution conditions.

Definition 13. *A* transition system *is given by* $\langle V, L, \mathcal{T}, l_0, \Theta \rangle$*, where*

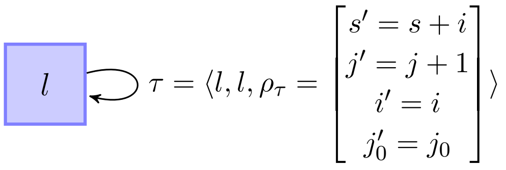
- *V is a set of variables,*
- *L is a set of locations and $l_0 \in L$ is the initial location.*
- *A* state *is given by an interpretation of the variables in V.*
- *A* transition $\tau \in \mathcal{T}$ *is given by a tuple $\langle l_{pre}, l_{post}, \rho_\tau \rangle$, where l_{pre} and l_{post} name the pre- and post- locations of τ. The transition relation ρ_τ is a first-order assertion over $V \cup V'$, where V correspond to current-state variable values and V' to the next-state variable values.*
- *Θ is the initial condition, given as a first-order assertion over V.*

The transition system is said to be affine *when ρ_τ is an affine form. And it is said to be* algebraic *when ρ_τ is an algebraic form.* □

Example 1. *Consider the following program corresponding to the multiplication of* 2 *numbers.*

```
        ...
int s, i, j, j_0;
        ...
//initialization
        ...
(s=0)&&(j=j_0)
        ...
 While (...){
   s := s+i;
   j := j-1;
  }
        ...
```

Its computational model is described by the following automaton:


l
$\tau = \langle l, l, \rho_\tau = \begin{bmatrix} s' = s + i \\ j' = j + 1 \\ i' = i \\ j_0' = j_0 \end{bmatrix} \rangle$

with $V = \{s, i, j, j_0\}$, $\Theta = (s = 0 \wedge j = j_0)$, $l_0 = l$, $L = \{l\}$ *and* $\mathcal{T} = \{\tau\}$. □

Definition 14. *Let W be a transition system. An* invariant *at location $l \in L$ is defined as an assertion over V which holds at all states reaching location l. An* invariant *of W is an assertion over V that holds at all locations.* □

To generate loop invariants, one needs to discover *inductive* assertion that holds at any steps of the loop.

Given our representational and computational models we can say that an invariant holds in the initial state of the system, that is the initial condition. We can also say that every possible transition preserves the invariant, that being the consecution conditions.

Definition 15. *Let $W = \langle V, L, \mathcal{T}, l_0, \Theta \rangle$ be a transition system and let $\mathbb{D}$ be an assertion domain.*

An assertion map for W is a map $\eta : L \to \mathbb{D}$. We say that η is inductive *if and only if the following conditions hold:*

- ***Initiation:*** $\Theta \models \eta(l_0)$
- ***Consecution:*** *For all τ in $\mathcal{T}$ s.t. $\tau = \langle l_i, l_j, \rho_\tau \rangle$ we have*

$$\eta(l_i) \wedge \rho_\tau \models \eta(l_j)'.$$

□

From R. W. Floyd [50], we know that if η is an inductive assertion map then $\eta(l)$ is an invariant at l for W.

3.3 New Continuous consecution conditions

In this section we treat discrete transitions by extending and adapting our previous work on loop invariant generation for discrete programs [79, 80]. We also consider discrete transitions that are part of connected components and circuits, thus generalizing the case of simple propagation.

Now we show how to encode continuous consecution conditions.

Definition 16. *Consider a transition system $W = \langle V, L, \mathcal{T}, l_0, \Theta \rangle$. Let $\tau = \langle l_i, l_j, \rho_\tau \rangle$ be a transition in $\mathcal{T}$ and let η be an algebraic inductive map with $\eta(l_i) \equiv (P_\eta(X_1, .., X_n) = 0)$ and $\eta(l_j) \equiv (P'_\eta(X_1, .., X_n) = 0)$ where P_η is a multivariate polynomial in $\mathbb{R}[X_1, .., X_n]$ such that it has null values at l_i and at l_j, i.e., before and after taking the transition. This do not implies that P_η is the null polynomial. We identify the following notions when encoding* continuous consecution conditions*:*

- *We say that η satisfies a* Fractional-scale consecution for τ *if and only if there exists a multivariate fractional $\frac{T}{Q}$ such that*

$$\rho_\tau \models (P_\eta(X'_1, .., X'_n) - \frac{T}{Q} P_\eta(X_1, .., X_n) = 0).$$

  We also say that P_η is a $\frac{T}{Q}$-scale discrete invariant.

- *We say that η satisfies a* Polynomial-scale consecution for τ *if and only if there exists a multivariate polynomial T such that*

$$\rho_\tau \models (P_\eta(X'_1, .., X'_n) - T P_\eta(X_1, .., X_n) = 0).$$

  We also say that P_η is a polynomial-scale and a T-scale discrete invariant.

- *We say that η satisfies a* Constant-scale consecution for τ *if and only if there exists a constant λ such that*

$$\rho_\tau \models (P_\eta(X_1', .., X_n') - \lambda P_\eta(X_1, .., X_n) = 0).$$

We also say that P_η is a constant-scale and a λ-scale discrete invariant.

$\square$

Constant-scale consecution encodes the fact that the numerical value of the polynomial P_η, associated with assertion $\eta(l_i)$, is given by λ times its numerical value throughout the transition τ. *Polynomial-scale* consecution encodes the fact that the numerical value of the polynomial P_η, associated with assertion $\eta(l_i)$, is given by T times its numerical value throughout the transition τ, where T is a polynomial in $\mathbb{R}[X_1, ..., X_n]$. Also, the T polynomials can be understood as *template multiplicative factors.* In other words, they are polynomials with unknown coefficients.

We are able to handle the general case when the loop describes a multivariate fractional system with *Fractional-scale* consecution. *Fractional-scale* consecution encodes the fact that the numerical value of the polynomial P_η, associated with assertion $\eta(l_i)$, is given by $\frac{T}{Q}$ times its numerical value throughout the transition τ. The fractional $\frac{T}{Q}$ can display unknown coefficients. As can be seen, the consecution conditions are relaxed when going from constant to fractional scaling.

3.4 Discrete transition with affine systems

In this section we use constant-scale consecution encoding.

Consider a transition systems corresponding to the loop $\tau = \langle l_i, l_i, \rho_\tau \rangle$ and its affine transition relation ρ_τ such that:

$$\rho_\tau \equiv \begin{bmatrix} X_1' = L_1(X_1, \ldots, X_n) \\ \vdots \\ X_n' = L_n(X_1, \ldots, x_n) \end{bmatrix}. \tag{3.1}$$

Where

$$L_i(X_1, ..., X_n) = \sum_{k=1}^{n} c_{i,k-1} X_k + c_i$$

are affine or linear forms.

3.4.1 Generating λ-scale Invariants

We have the following λ-scale invariant characterization.

Theorem 1. *Consider a transition system corresponding to a loop τ as described in Eq. (3.4). A polynomial Q in $\mathbb{R}[X_1, .., X_n]$ is a λ-scale invariant for constant-scale consecution with* parametric constant $\lambda \in \mathbb{R}$ *for τ if and only if*

$$Q(L_1(X_1, .., X_n), .., L_n(X_1, .., X_n)) = \lambda Q(X_1, .., X_n). \tag{3.2}$$

□

Demonstração. If $Q(X'_1, .., X'_n) - \lambda Q(X_1, .., X_n)$ belongs to the ideal I generated by the family $(X'_1 - L_1, \dots, X'_n - L_n)$, then there exists a family $(A_1, \dots, A_n)$ of polynomials in $\mathbb{R}[X'_1, .., X'_n, X_1, .., X_n]$ such that

$$Q(X'_1, .., X'_n) - \lambda Q(X_1, .., X_n) = (X'_1 - L_1)A_1 + \cdots + (X'_n - L_n)A_n.$$

Letting $X'_i = L_i$, we obtain

$$Q(L_1(X_1, ..., X_n), .., L_n(X_1, ..., X_n)) = \lambda Q(X_1, ..., X_n).$$

Conversely, suppose that

$$Q(L_1(X_1, \dots, X_n), .., L_n(X_1, \dots, X_n)) = \lambda Q(X_1, \dots, X_n),$$

then, as $Q(X'_1, .., X'_n)$ is equal to $Q(L_1, .., L_n)$ modulo the ideal I, we get that $Q(X'_1, .., X'_n) = \lambda Q(X_1, \dots, X_n)$ modulo I. □

In this case, $Q \in \mathbb{R}[X_1, .., X_n]$ is of degree r. We show that for good choices of λ there always exists such a λ-invariant that is not trivial.

We note that $Q(L_1(X_1, .., X_n), .., L_n(X_1, .., X_n))$ is also of degree r because all L_i's are of degree 1.

Transposing the situation and Eq. (3.2) to linear algebra, consider the morphism

$$\mathscr{M} : \left\{ \begin{array}{ccc} \mathbb{R}_r[X_1, \dots, X_n] & \rightarrow & \mathbb{R}_r[X_1, \dots, X_n] \\ Q(X_1, ..., X_n) & \mapsto & Q(L_1(X_1, .., X_n), \dots, L_n(X_1, .., X_n)). \end{array} \right.$$

This is indeed an endomorphism because all L_i's are of degree 1. Let M be its matrix in the canonical basis of $\mathbb{R}_r[X_1, ., X_n]$.

First, we show how we build the matrix M.

Example 2. (Running example) *Consider the following loop $\tau = \langle l_i, l_i, \rho_\tau \rangle$ with*

$$\rho_\tau = \begin{bmatrix} x_1' = 2x_1 + x_2 + 1 \\ x_2' = 3x_2 + 4 \end{bmatrix}. \tag{3.3}$$

In this example we have two polynomials of degree 1, with two variables. They are

$$L_1(x_1, x_2) = 2x_1 + x_2 + 1,$$

and

$$L_2(x_1, x_2) = 3x_2 + 4.$$

Now consider the associated endomorphism $\mathscr{M}$ from $\mathbb{R}_2[x_1, x_2]$ to $\mathbb{R}_2[x_1, x_2]$). Using the basis

$$B_1 = (x_1^2, x_1x_2, x_2^2, x_1, x_2, 1)$$

of $\mathbb{R}_2[x_1, x_2]$, we define the matrix M. To do so, we compute $\mathscr{M}(P)$ for all elements P in the basis

$$B_1 = (x_1^2, x_1x_2, x_2^2, x_1, x_2, 1)$$

and we express the results in the same basis B_1. In other words, to get the first column of M we first consider

$$P(x_1, x_2) = x_1^2$$

as the first element of B_1, and we compute

$$\mathscr{M}(P) = P(L_1(x_1, x_2), L_2(x_1, x_2))$$

which is expressed in B_1 as

$$\mathscr{M}(x_1^2) = 4\, x_1^2 + 4\, x_1x_2 + 1\, x_2^2 + 4\, x_1 + 2\, x_2 + 1 \times 1$$

$$M = \begin{pmatrix} 4 & 0 & 0 & 0 & 0 & 0 \\ 4 & 6 & 0 & 0 & 0 & 0 \\ 1 & 3 & 9 & 0 & 0 & 0 \\ 4 & 8 & 0 & 2 & 0 & 0 \\ 2 & 7 & 24 & 1 & 3 & 0 \\ 1 & 4 & 16 & 1 & 4 & 1 \end{pmatrix}.$$

□

Now, let $Q \in \mathbb{R}[X_1, .., X_n]$ be a λ-scale invariant for constant-scale consecution with *parametric constant* $\lambda \in \mathbb{R}$ for a given system defined by $L_1, .., L_n \in \mathbb{R}[X_1, .., X_n]$.

By theorem 1, we have

$$Q(L_1(X_1, ..., X_n), .., L_n(X_1, ..., X_n)) = \lambda Q(X_1, ..., X_n).$$

Using the associated endomorphism $\mathscr{M}$, we have:

$$\begin{array}{rcl} Q(L_1(X_1, ..., X_n), .., L_n(X_1, ..., X_n)) & = & \lambda Q(X_1, ..., X_n) \\ \Leftrightarrow \mathscr{M}(Q) & = & \lambda Q \\ \Leftrightarrow \mathscr{M}(Q) & = & \lambda \mathscr{I}(Q) \\ \Leftrightarrow (\mathscr{M} - \lambda \mathscr{I})(Q) & = & 0_{\mathbb{R}[X_1, .., X_n]} \\ \Leftrightarrow Q & \in & Ker(M - \lambda I), \end{array}$$

where $\mathscr{I}$ is the identity endomorphism and I is the associated identity matrix of $\mathbb{R}_r[X_1, ..X_n]$. Hence, λ must be an eigenvalue of M if we want to find a non null λ-invariant whose coefficients will be those of an eigenvector.

We can now state the following theorem.

Theorem 2. *A polynomial Q of $\mathbb{R}_r[X_1, .., X_n]$ is λ-invariant for constant-scale consecution if and only if there exists an eigenvalue λ of M such that Q belongs to the eigenspace corresponding to λ.* □

We also notice that the last column of M is always $(0, \ldots, 0, 1)^\top$ by definition of the matrix M. Thus 1 is always an eigenvalue of M, with a corresponding eigenvector, which gives the trivial λ-invariant $Q(X_1, .., X_n) = a$, where a is the coefficient of the constant term. . Eigenvalue 1 always gives the constant polynomial as a λ-invariant, but it might give better invariants for other eigenvectors if $dim(Ker(M - \lambda I)) \geq 2$, as we will see in the sequel.

Example 3. *Looking at the eigenvalues of the matrix M of the previous running example 2, we fix λ to be 4, we get that the corresponding eigenspace is generated by the vector*

$$(1, -2, 1, -6, 6, 9)^\top.$$

As a λ-invariant polynomial Q for constant-scale consecution with parameter 4, we get

```
Constant scaling discrete step
Lambda = 4 Eigenspace
{{1, -2, 1, -6, 6, 9}}
```

Interpreted in the canonical basis of $\mathbb{R}[x_1, x_2]$, the associated 4-invariant is

$$Q(x_1, x_2) = 1{x_1}^2 - 2x_1x_2 + {x_2}^2 - 6x_1 + 6x_2 + 9.$$

□

Example 4. *(General Case for 2 Variables) We first treat the general case where the transition system has only two variables. We will look for a λ-invariant Q of degree two. Let*

$$\rho_\tau = \begin{bmatrix} x_1' = c_{1,0}x_1 + c_{1,1}x_2 + c_{1,2} \\ x_2' = c_{2,0}x_1 + c_{2,1}x_2 + c_{2,2} \end{bmatrix}$$

Recall that we must solve the equation $Q(c_{1,0}X_1 + c_{1,1}X_2 + c_{1,2}, c_{2,0}X_1 + c_{2,1}X_2 + c_{2,2}) = \lambda Q(X_1, X_2)$. Thus, for M we get the following matrix:

$$\begin{pmatrix} {c_{1,0}}^2 & c_{1,0}c_{2,0} & {c_{2,0}}^2 & 0 & 0 & 0 \\ 2c_{1,0}c_{1,1} & c_{1,0}c_{2,1} + c_{1,1}c_{2,0} & 2c_{2,0}c_{2,1} & 0 & 0 & 0 \\ {c_{1,1}}^2 & c_{1,1}c_{2,1} & {c_{2,1}}^2 & 0 & 0 & 0 \\ 2c_{1,0}c_{1,2} & c_{1,0}c_{2,2} + c_{1,2}c_{2,0} & 2c_{2,0}c_{2,2} & c_{1,0} & c_{2,0} & 0 \\ 2c_{1,1}c_{1,2} & c_{1,1}c_{2,2} + c_{1,2}c_{2,1} & 2c_{2,1}c_{2,2} & c_{1,1} & c_{2,1} & 0 \\ {c_{1,2}}^2 & c_{1,2}c_{2,2} & {c_{2,2}}^2 & c_{1,2} & c_{2,2} & 1 \end{pmatrix}$$

We see that the last column is as predicted, besides the matrix is block diagonal. Thus its characteristic polynomial is $P(\lambda) = (1 - \lambda)P_1(\lambda)P_2(\lambda)$, with P_1 being the characteristic polynomial of

$$\begin{pmatrix} c_{1,0} & c_{2,0} \\ c_{1,1} & c_{2,1} \end{pmatrix},$$

and P_2 being the characteristic polynomial of

$$\begin{pmatrix} {c_{1,0}}^2 & c_{1,0}c_{2,0} & {c_{2,0}}^2 \\ 2c_{1,0}c_{1,1} & c_{1,0}c_{2,1} + c_{1,1}c_{2,0} & 2c_{2,0}c_{2,1} \\ {c_{1,1}}^2 & c_{1,1}c_{2,1} & {c_{2,1}}^2 \end{pmatrix}.$$

Here P_2 is of degree 3 and has at least one real root, which can be computed by Lagrange's resolvent method. Choosing λ to be this root, the corresponding eigenvectors will give non-trivial λ-invariants of degree two as at least one of the coefficients of the monomial x_1^2, x_1x_2 and x_2r must be non null for such an eigenvector. □

Corollary 1. *Let M the matrix introduce in this section. The problem of finding a* non-trivial λ-invariant *is* decidable *if one of the following assertions is true:*

- *M is block triangular (with 4×4 blocks or less) ,*
- *The eigenspace associated with eigenvalue 1 is of dimension greater than 1.*

□

Demonstração. Suppose that M be block triangular with 4×4 blocks or less. Then it's characteristic polynomial will a product of polynomials of degree less than four whose roots can be calculated by Lagrange's resolvent method [76].

For the second assertion, we already know that 1 is an eigenvalue. Suppose that the corresponding eigenspace is of dimension exactly one. Then the only vectors in that space are the constant polynomials. Whereas if it is of dimension two or more, than we get polynomials that are non trivial in the eigenspace. With theorem 2 below, we see that it is a particularly interesting case. □

3.4.2 Intersection with an initial hyperplane

Let $Q \in \mathbb{R}_r[X_1, .., X_n]$ be a λ-invariant for constant-scale consecution, meaning that

$$Q(L_1(X_1, .., X_n), .., L_n(X_1, .., X_n)) = \lambda Q(X_1, .., X_n).$$

Now let $u_1, ..., u_n$ be the initial values of $X_1, ..., X_n$. For the initial step we need $Q(u_1, ..., u_n) = 0$.

We have the following linear form in $\mathbb{R}_r[X_1, ..., X_n]$:

$$P \mapsto P(u_1, ..., u_n).$$

Hence initial values correspond to a hyperplane in $\mathbb{R}_r[X_1, ..., X_n]$, given by the kernel of $P \mapsto P(u_1, ..., u_n)$.

Now, if we add the initiation step, $Q(X_1, ..., X_n) = 0$ will be an inductive invariant (see Definition 19) if and only if there exists an eigenvalue λ of M such that Q belongs to the intersection of the eigenspace corresponding to λ and the hyperplane $Q(u_1, \ldots, u_n) = 0$, given by the initial values $(u_1, \ldots, u_n)$.

Theorem 3. *A polynomial Q in $\mathbb{R}_r[X_1, .., X_n]$ is an inductive invariant for the affine loop (see Definition 15) with initial values $(u_1, \ldots, u_n)$ if and only if there is an eigenvalue λ of M such that Q is in the intersection of the eigenspace of λ and the hyperplane $Q(u_1, \ldots, u_n) = 0$.* □

In the following corollary, we state an important result.

Corollary 2. *There will be a non-null invariant polynomial for any given initial values if and only if there exists an eigenspace of M with dimension at least 2.* □

Demonstração. ($\Rightarrow$) There is a λ-scale invariant for any initial value. Then the corresponding eigenspace has dimension at least 2. Indeed, assume that the space was of dimension only 1, which is at least necessary to have λ-invariants. Taking any nonzero vector Q in the eigenspace (i.e. a λ-invariant), Q should lie in any hyperplane of initial values, i.e., for every n-tuple $(u_1, \ldots, u_n)$, one would have $Q(u_1, \ldots, u_n) = 0$, i.e, $Q = 0$, which is absurd.

($\Leftarrow$) Any eigenspace of M with dimension at least 2 will intersect any space (semi-hyperplan, ...) given by any initial constraints. As any hyperplane is of co-dimension one in V_r, it must have a nonzero intersection with any subspace of dimension strictly greater than one. □

Example 5. *Now, we return to running example 2. Matrix M has 6 distinct eigenvalues, so that eigenspaces are of dimension 1. We denote by E_λ the eigenspace corresponding to λ.*

- E_4 *has basis* $(1, -2, 1, -6, 6, 9)^\top$,
- E_6 *has basis* $(0, 1, -1, 2, -5, 6)^\top$,
- E_9 *has basis* $(0, 0, 1, 0, 4, 4)^\top$,
- E_2 *has basis* $(0, 0, 0, 1, -1, -3)^\top$,
- E_3 *has basis* $(0, 0, 0, 0, 1, 2)^\top$, *and*
- E_1 *has basis* $(0, 0, 0, 0, 0, 1)^\top$.

Suppose that the initiation step is given by $(x_1 = 0, x_2 = -2)$, i.e. $(u_1, u_2) = (0, 2)$ which corresponds to the hyperplane $Q(0, 2) = 0$ in $\mathbb{R}_2[x_1, x_2]$. Here, we start with simple initial conditions, we will consider general conditions in the sequel. Our Theorem 3 applies, and it is clear that $(0, 0, 1, 0, 4, 4)^\top$ belongs to this hyperplane, so that ${X_2}^2 + 4X_2 + 4 = 0$ is an inductive invariant for the loop with these specific initial conditions. □

Example 6. *We study the following transition system [120], corresponding to the multiplication of 2 numbers, and where the transition considered is $\tau = \langle l_i, l_i, \rho_\tau \rangle$ with*

$$\rho_\tau = \begin{bmatrix} s' = s + i \\ j' = j + 1 \\ i' = i \\ j_0' = j_0 \end{bmatrix}.$$

We need to find a λ such that

$$Q(s + i, j + 1, i, j_0) = \lambda Q(s, j, i, j_0).$$

- ***Step 1****: We build the associated matrix M:*

$$\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & -1 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 1
\end{pmatrix}$$

- ***Step 2****: We compute the eigenvectors which will provide us with a basis of non-trivial λ-invariants. Here an evident eigenvalue is* 1.

- ***Step 3****: It is clear in view of the matrix M that*

$$dim(Ker(M - I)) \geq 2.$$

As the eigenspace associated to eigenvalue 1 *is of dimension* 2*, our Corollary 2 applies.*

For example the vector

$$(1, 0, 0, 0, 0, 1, 0, 0, -1, 0, 0, 0, 0, 0, 0)^\top$$

is the eigenvector corresponding to the λ-invariant

$$s + ji - ij_0.$$

Note that without Grobner bases and quantifier elimination we found the invariant $s + ji - ij_0 = 0$ obtained in S. Sankaranarayanan, H.B. Sipma, and Z. Manna [120]. The consecution scale technique will give a non-null invariant whatever the initial values are, and this explains why a non-trivial invariant was found in S. Sankaranarayanan, H.B. Sipma, and Z. Manna [120]). □

3.4.3 Limit of constant-scale consecution

Let's consider an algebraic transition relation

$$\rho_\tau \equiv \begin{bmatrix} x'_1 = P_1(x_1,\dots,x_n) \\ \vdots \\ x'_m = P_m(x_1,\dots,x_n) \end{bmatrix}, \tag{3.4}$$

where each polynomial P_i has a degree greater than 1.

Example 7. *Let's consider the following loop:*

$$\rho_\tau \equiv \begin{bmatrix} x' = x(y+1) \\ y' = y^2 \end{bmatrix}.$$

At the step k of the iteration, this loop compute the sum: $1+y+\cdots+y^{2^k-1}$. We consider $P(x,y) = a_0x^2 + a_1xy + a_2y^2 + a_3x + a_4y + a_5$ as candidate λ-invariant. The loop ideal of $\mathbb{K}[x',y',x,y]$ which is generated by the following Grobner Bases $\{x'-x(y+1), y'-y^2\}$, with the total-degree lexicographic ordering given by the precedence: $x' > y' > x > y$. Modulo this loop ideal, we have $P(x',y') = P(x(y+1),y^2)$ and we denote $P'(x,y) = P(x(y+1),y^2)$. After expanding we get $P'(x,y) = a_0x^2y^2 + a_1xy^3 + a_2y^4 + 2a_0x^2y + a_1xy^2 + a_0x^2 + a_3xy + a_4y^2 + a_3x + a_5$. If we try the constant-scale consecution with parameter λ we obtain:

$$\begin{cases} a_0 = 0 & a_1 = 0 & a_3 = \lambda a_3 \\ a_1 = 0 & a_0 = \lambda a_0 & \lambda a_4 = 0 \\ a_2 = 0 & a_3 = \lambda a_1 & a_5 = \lambda a_5 \\ 2a_0 = 0 & a_4 = \lambda a_2 & \end{cases}$$

After simplification we get: $a_0 = a_1 = a_2 = a_3 = a_4 = 0$ and $a_5 = \lambda a_5$. If $\lambda \neq 1$ then $a_5 = 0$, which leads to a null *invariant. Otherwise, $\lambda = 1$ and we obtain a* constant *invariant (a_5). Also, by considering the initial condition, we remark that it will imply that the constant invariant a_5 is null. So, using a constraint-based approach [120] with constant-scaling, we can obtain only constant or null (trivial) invariant.* □

In the following section, we show how we handle this problem.

3.5 Handling algebraic discrete transition systems

3.5.1 T-scale invariant generation

Consider an algebraic transition system:

$$\rho_\tau \equiv \begin{bmatrix} X_1' = P_1(X_1,..,X_n) \\ \vdots \\ X_n' = P_n(X_1,..,X_n) \end{bmatrix}, \tag{3.5}$$

where the P_i's are in $\mathbb{R}[X_1,..,X_n]$. We have the following T-scale discrete invariant characterization.

Theorem 4. *A polynomial Q in $\mathbb{R}[X_1,..,X_n]$ is a T-scale discrete invariant for polynomial-scale consecution with a* parametric polynomial $T \in \mathbb{R}[X_1,...,X_n]$ *for τ if and only if*

$$Q(P_1(X_1,..,X_n),..,P_n(X_1,..,X_n)) = T(X_1,..,X_n)Q(X_1,..,X_n).$$

$\square$

Demonstração. If $Q(X_1',..,X_n') - TQ(X_1,..,X_n)$ belongs to the ideal I generated by the family $(X_1' - P_1,\dots,X_n' - P_n)$, then there exists a family $(A_1,\dots,A_n)$ of polynomials in $K[X_1',..,X_n',X_1,..,X_n]$ such that

$$Q(X_1',..,X_n') - TQ(X_1,..,X_n) = (X_1' - P_1)A_1 + \cdots + (X_n' - P_n)A_n.$$

Letting $X_i' = P_i$, we obtain

$$Q(P_1(X_1,...,X_n),...,P_n(X_1,...,X_n)) = TQ(X_1,...,X_n).$$

Conversely, suppose

$$Q(P_1(X_1,\dots,X_n),..,P_n(X_1,\dots,X_n)) = TQ(X_1,\dots,X_n),$$

then as $Q(X_1',..,X_n')$ is equal to $Q(P_1,..,P_n)$ modulo the ideal I, we get that $Q(X_1',..,X_n') = \lambda Q(X_1,\dots,X_n)$ modulo I. $\square$

Example 8. *Reconsider Example 7. We take $(y = y_0, x = 1)$ as initial values. We propose to use* polynomial scale consecution *with a parametric polynomial*

$$T(x,y) = b_0y^2 + b_1x + b_2y + b_3.$$

We obtain

$$P'(s,x) = (b_0y^2 + b_1x + b_2y + b_3) \cdot P(x,y).$$

In other words, we obtain the following multi-parametric linear system, with parameters b_0, b_1, b_2, b_3:

$$\begin{cases} a_0 = b_0 a_0, & 0 = b_2 a_5 + b_3 a_4, & a_3 = b_1 a_4 + b_2 a_3 + b_3 a_1, \\ a_1 = b_0 a_1, & 0 = b_0 a_4 + b_2 a_2, & a_4 = b_0 a_5 + b_2 a_4 + b_3 a_2, \\ a_2 = b_0 a_2, & a_3 = b_1 a_5 + b_3 a_3, & a_1 = a_3 b_0 + b_1 a_2 + b_2 a_1, \\ a_5 = b_3 a_5, & a_0 = b_1 a_3 + b_3 a_0, & \\ 0 = b_1 a_0, & 2a_0 = b_1 a_1 + b_2 a_0. & \end{cases}$$

Now we describe a decision procedure for parameter valuations. Considering the first three equations and choose $b_0 = 1$. In this way we maintain high degree invariant for, otherwise, the coefficients a_0, a_1, a_2 of the highest degree terms would be null. Then, we obtain another system with $b_1 a_0 = 0$. For the same degree, choose $b_1 = 0$. Then we have $b_2 a_0 = 2a_0$. As a direct consequence, b_2 is set to 2. Since equation $b_3 a_0 = a_0$ is in the resulting system, b_3 is set to 1. Finally, we obtain the following system :

$$\begin{cases} a_3 + a_1 = 0 \\ a_4 + 2a_2 = 0 \\ a_2 - a_5 = 0. \end{cases}$$

With less equations than variables, we will have a non-trivial *solution for the generation of a T-invariant. Now, we add the hyperplane corresponding to the initial values: $a_2 {y_0}^2 + (a_1 + a_4) y_0 + a_0 + a_1 + a_5 = 0$. As there are six variables and four equations, we will have a* non-trivial *solution. A possible solution is the vector*

$$(y_0(1 - y_0), 1, 1, -1, -2, 1)^\top,$$

that is, $y_0(1 - y_0)x^2 + xy + y^2 - x - 2y + 1 = 0$ is an invariant. Note that $T(x, y) = y^2 + y + 1$. □

Remarks 1. *That is a simple constraint-based procedure, which can fail in more complex cases. Shortly, we will present a superior technique, form a more encompassing point of view.* □

3.5.2 General theory for discrete transition with polynomial systems

If $Q \in \mathbb{R}[X_1, .., X_n]$ is of degree r and the maximal degree of the P_i's is d, then we are looking for a T of degree $e = dr - r$. Write its ordered coefficients as $\lambda_0, ..., \lambda_s$, with $s + 1$ being the number of monomials of degree inferior to e.

Let M be the matrix, in the canonical basis of $\mathbb{R}_r[X_1, .., X_2]$ and $\mathbb{R}_{dr}[X_1, .., X_n]$, of the morphism

$$\mathscr{M} : \begin{cases} \mathbb{R}_r[X_1, \dots, X_n] & \to & \mathbb{R}_{dr}[X_1, \dots, X_n] \\ Q(X_1, ..., X_n) & \mapsto & Q(P_1(X_1, .., X_n), \dots, P_n(X_1, .., X_n)). \end{cases}$$

Let L be the matrix, in the canonical basis of $\mathbb{R}_r$ and $\mathbb{R}_{dr}$, of the morphism

$$\mathscr{L} : \begin{cases} \mathbb{R}_r[X_1, \ldots, X_n] & \to & \mathbb{R}_{dr}[X_1, \ldots, X_n] \\ P & \mapsto & TP. \end{cases}$$

Matrix L will have a very simple form: its non zero coefficients are the λ_i's, and it has a natural block decomposition.

Now let $Q \in \mathbb{R}[X_1, .., X_n]$ be a T-scale discrete invariant for a transition relation defined by the P_i's. Then

$$\begin{aligned} Q(P_1(X_1, .., X_n), .., P_n(X_1, .., X_n)) &= T(X_1, .., X_n)Q(X_1, .., X_n) \\ \Leftrightarrow \mathscr{M}(Q) &= \mathscr{L}(Q) \\ \Leftrightarrow (\mathscr{M} - \mathscr{L})(Q) &= 0_{\mathbb{R}[X_1,..,X_n]} \\ \Leftrightarrow Q &\in Ker(M - L). \end{aligned}$$

A T-scale discrete invariant is nothing else than a vector in the kernel of $M - L$. Our problem is equivalent to finding a L such that $M - L$ has a non trivial kernel.

Theorem 5. *Consider M as described above. Then, there will be a T-scale discrete invariant if and only if there exists a matrix L, corresponding to $P \mapsto TP$, such that $M - L$ has a nontrivial kernel. Further, any vector in the kernel of $M - L$ will give rise to a T-scale invariant.* □

Demonstração. In fact, a polynomial Q is T-invariant if and only if

$$Q(P_1(X_1, .., X_n), .., P_n(X_1, .., X_n)) = T(X_1, .., X_n)Q(X_1, .., X_n),$$

i.e., if and only if $\mathscr{M}(Q) = \mathscr{L}(Q)$, or $(\mathscr{M} - \mathscr{L})(Q) = 0$. Writing this in matrix equivalent terms, we get the desired result. □

Again, the last column of M is $(0, ..., 0, 1)^\top$, and the last column of L is $(0, .., 0, \lambda_0, .., \lambda_s)^\top$. Hence, choosing every λ_i to be zero, except for $\lambda_s = 1$, the last column of M - L will be null. With this choice of L (or $T = 1$), we at least get T-invariants corresponding to constant polynomials.

Now, $M - L$ having a non trivial kernel is equivalent to its rank being less than the dimension $v(r)$ of V_r. This is equivalent to the fact that each $v(r) \times v(r)$ subdeterminant of $M - L$ is equal to zero [76]. Those determinants are polynomials with variables $(\lambda_0, \lambda_1, \cdots, \lambda_s)$, which we will denote by

$$V_1(\lambda_0, \lambda_1, \cdots, \lambda_s), \ldots, V_s(\lambda_0, \lambda_1, \cdots, \lambda_s).$$

Theorem 6. *There is a non trivial T-scale invariant if and only if the polynomials $(V_1, .., V_s)$ admit a common root, other than the trivial one $(0, \dots, 0, 1)$.* □

Remarks 2. *This theorem provides us with important existence results. But there is a more practical way of computing invariant ideals without computing common roots. We will get to that in a moment.* □

Example 9. *(Loop with two variables, T-scale invariant of degree 2) We first study the general case of degree two algebraic transition systems with two variables in the loop. Such transition systems have the form:*

$$\rho_\tau \equiv \begin{bmatrix} x' = c_0x^2 + c_1xy + c_2y^2 + c_3x + c_4y + c_5 \\ y' = d_0x^2 + d_1xy + d_2y^2 + d_3x + d_4y + d_5 \end{bmatrix},$$

$$M = \begin{pmatrix}
{c_0}^2 & c_0d_0 & {d_0}^2 & 0 & 0 & 0 \\
2c_0c_1 & c_0d_1 + c_1d_0 & 2d_0d_1 & 0 & 0 & 0 \\
2c_0c_2 + {c_1}^2 & c_0d_2 + c_1d_1 + c_2d_0 & 2d_0d_2 + {d_1}^2 & 0 & 0 & 0 \\
2c_1d_1 & c_1d_2 + c_2d_1 & 2d_1d_2 & 0 & 0 & 0 \\
{c_2}^2 & c_2d_2 & {d_2}^2 & 0 & 0 & 0 \\
2c_0c_3 & c_0d_3 + c_3d_0 & 2d_0d_3 & 0 & 0 & 0 \\
2(c_0c_4 + c_1c_3) & c_0d_4 + c_1d_3 + c_3d_1 + c_4d_0 & 2(d_0d_4 + d_1d_3) & 0 & 0 & 0 \\
2(c_1c_4 + c_2c_3) & c_1d_4 + c_2d_3 + c_3d_2 + c_4d_1 & 2(d_1d_4 + d_2d_3) & 0 & 0 & 0 \\
2c_2c_4 & c_2d_4 + c_4d_2 & 2d_2d_4 & 0 & 0 & 0 \\
2c_0c_5 + {c_3}^2 & c_0d_5 + c_3d_3 + c_5d_0 & 2d_0d_5 + {d_3}^2 & c_0 & d_0 & 0 \\
2(c_1c_5 + c_3c_4) & c_1d_5 + c_3d_4 + c_4d_3 + c_5d_1 & 2(d_1d_5 + d_3d_4) & c_1 & d_1 & 0 \\
2c_2c_5 + {c_4}^2 & c_2d_5 + c_4d_4 + c_5d_2 & 2d_2d_5 + {d_4}^2 & c_2 & d_2 & 0 \\
2c_3c_5 & c_3d_5 + c_5d_3 & 2d_3d_5 & c_3 & d_3 & 0 \\
2c_4c_5 & c_4d_5 + c_5d_4 & 2d_4d_5 & c_4 & d_4 & 0 \\
{c_5}^2 & c_5d_5 & {d_5}^2 & c_5 & d_5 & 1
\end{pmatrix}$$

and

$$L = \begin{pmatrix} \lambda_0 & 0 & 0 & 0 & 0 & 0 \\ \lambda_1 & \lambda_0 & 0 & 0 & 0 & 0 \\ \lambda_2 & \lambda_1 & \lambda_0 & 0 & 0 & 0 \\ 0 & \lambda_2 & \lambda_1 & 0 & 0 & 0 \\ 0 & 0 & \lambda_2 & 0 & 0 & \\ \lambda_3 & 0 & 0 & \lambda_0 & 0 & 0 \\ \lambda_4 & \lambda_3 & 0 & \lambda_1 & \lambda_0 & 0 \\ 0 & \lambda_4 & \lambda_3 & \lambda_2 & \lambda_1 & 0 \\ 0 & 0 & \lambda_4 & 0 & \lambda_2 & 0 \\ \lambda_5 & 0 & 0 & \lambda_3 & 0 & \lambda_0 \\ 0 & \lambda_5 & 0 & \lambda_4 & \lambda_3 & \lambda_1 \\ 0 & 0 & \lambda_5 & 0 & \lambda_4 & \lambda_2 \\ 0 & 0 & 0 & \lambda_5 & 0 & \lambda_3 \\ 0 & 0 & 0 & 0 & \lambda_5 & \lambda_4 \\ 0 & 0 & 0 & 0 & 0 & \lambda_5 \end{pmatrix}.$$

For the rank of $M-L$ to be less than 6, one has to calculate each 6×6 sub-determinant obtained by canceling 9 lines of $M-L$. They will be polynomials of degree less than 6 in the variables $(\lambda_0, ..., \lambda_5)$. Now, L is such that $M-L$ will be of degree less than 6 if and only if $(\lambda_0, ..., \lambda_5)$ are roots of each of those polynomials. □

Remarks 3. *In many cases, it is easy to find a matrix L such that $M-L$ has a non trivial kernel. We describe two decidable classes: (i) suppose that in the previous case, c_2, c_4 and c_5 are null, then one can choose $(\lambda_0, \dots, \lambda_s)$ in order to make the first column zero; and (ii) the third column can be canceled using good choices for the λ_i's, if d_0, d_3 and d_5 are zero.* □

3.5.3 Generating invariant ideals with an initiation step

Consider initial values given by unknown parameters $(X_1 = u_1, \dots, X_n = u_n)$. The initial step defines, on $\mathbb{R}_r[x_1, \dots, x_n]$, a linear form

$$P \mapsto P(u_1, ..., u_n).$$

Hence, in terms of linear algebra, initial values correspond to a hyperplane of $\mathbb{R}_r[X_1, .., X_n]$, given by the kernel of $P \mapsto P(u_1, ..., u_n)$, which is

$$\{Q \in \mathbb{R}_r[X_1, .., X_n] \mid Q(u_1, \dots, u_n) = 0\}.$$

Theorem 7. *Let Q be in $\mathbb{R}_r[X_1, .., X_n]$. Then Q is an inductive invariant for the transition system with initial values $(u_1, .., u_n)$ if and only if there exists a matrix $L \neq 0$ (the* *one of $P \mapsto TP$), corresponding to T in $\mathbb{R}_e[X_1, .., X_n]$, such that Q is in the intersection of $Ker(M - L)$ and the hyperplane given by the initial values $Q(u_1, \dots, u_n) = 0$.*

The invariants correspond to vectors in the intersection. □

Now, if $Dim(Ker(M - L)) \geq 2$ then $Ker(M - L)$ would intersect any initial (semi-)hyperplane.

We can state the following Corollary, important in practice.

Corollary 3. *There are non-trivial invariant for any given initial values if and only if there exists a matrix L such that $Ker(M - L)$ has dimension at least 2. The basis of $Ker(M - L)$ being a basis for non-trivial invariants.* □

There are non-trivial invariants for any given initial values if and only if there exists a matrix L, corresponding to the template multiplicative in T such that $Ker(M - L)$ has dimension at least 2.

3.5.4 Running example

Example 10. *(Running example) Let's consider the following transition:*

$$\tau = \langle l_i, l_j, \rho_\tau \equiv \begin{bmatrix} x' = xy + x \\ y' = y^2 \end{bmatrix} \rangle.$$

***Step 1**: We build the matrix $M - L$. The maximal degree of ρ_τ is $d = 2$, and so the T-scale invariant will be of degree $r = 2$. Also, T is of degree $e = dr - r = 2$ and we write $\lambda_0, ..., \lambda_5$ as its ordered coefficients. Then its canonical form is $T = \lambda_0 x^2 + \lambda_1 xy + \lambda_2 y^2 + \lambda_3 x + \lambda_4 y + \lambda_5$. Consider the associated morphisms $\mathscr{M}$ and $\mathscr{L}$ from $\mathbb{R}_2[x, y]$ to $\mathbb{R}_4[x, y]$. Using the basis*

$$C_1 = (x^2, xy, y^2, x, y, 1)$$

of $\mathbb{R}_2[x, y]$ and the basis

$$C_2 = (x^4, yx^3, y^2x^2, y^3x, y^4, x^3, x^2y, xy^2, y^3, x^2, xy, y^2, x, y, 1)$$

of $\mathbb{R}_4[x,y]$*, our algorithm compute the matrix* $M-L$ *as*

$$M-L=\begin{pmatrix}
-\lambda_0 & 0 & 0 & 0 & 0 & 0\\
-\lambda_1 & -\lambda_0 & 0 & 0 & 0 & 0\\
1-\lambda_2 & -\lambda_1 & -\lambda_0 & 0 & 0 & 0\\
0 & 1-\lambda_2 & -\lambda_1 & 0 & 0 & 0\\
0 & 0 & 1-\lambda_2 & 0 & 0 & 0\\
-\lambda_3 & 0 & 0 & -\lambda_0 & 0 & 0\\
2-\lambda_4 & -\lambda_3 & 0 & -\lambda_1 & -\lambda_0 & 0\\
0 & 1-\lambda_4 & -\lambda_3 & -\lambda_2 & -\lambda_1 & 0\\
0 & 0 & -\lambda_4 & 0 & -\lambda_2 & 0\\
1-\lambda_5 & 0 & 0 & -\lambda_3 & 0 & -\lambda_0\\
0 & -\lambda_5 & 0 & 1-\lambda_4 & -\lambda_3 & -\lambda_1\\
0 & 0 & -\lambda_5 & 0 & 1-\lambda_4 & -\lambda_2\\
0 & 0 & 0 & 1-\lambda_5 & 0 & -\lambda_3\\
0 & 0 & 0 & 0 & -\lambda_5 & -\lambda_4\\
0 & 0 & 0 & 0 & 0 & 1-\lambda_5
\end{pmatrix}.$$

Step 2*: We then reduce the rank of* $M-L$ *by assigning values to the* λ_i*'s. Our procedure fixes* $\lambda_0=\lambda_1=\lambda_3=0$*,* $\lambda_2=\lambda_5=1$ *and* $\lambda_4=2$*, so that* $T(x,y)=y^2+2y+1$*. The first column of* $M-L$ *becomes zero and the second column is equal to the fourth. Hence, the rank of* $M-L$ *is less than* 4 *and its kernel has dimension at least* 2*. Any vector in this kernel will be a* T*-invariant.*

Step 3*: Now matrix* $M-L$ *satisfies the hypotheses of Theorem 5(iii). So, there will always be invariants, whatever the initial values. We compute the basis of* $Ker(M-L)$*:*

```
Polynomial scaling discrete step
T(x,y) = y^2 + 2 y + 1
Module of degree 6 and rank 3 and Kernel of dimension 3
{{1, 0, 0, 0, 0, 0}, {0, 1, 0, -1, 0, 0}, {0, 0, 1, 0, -2, 1}}
```

The vectors of the basis are interpreted in the canonical basis C_1 *of* $\mathbb{R}_2[x,y]$*:*

```
Basis of invariant Ideal
{x^2, x y - x, y^2 - 2 y + 1}
```

We thus obtained an ideal for non trivial inductive invariants. In other words, for all G_1*,* G_2*,* $G_3\in\mathbb{R}[x,y]$*,*

$$G_1(x,y)(x^2)+G_2(x,y)(xy-x)+G_3(x,y)(y^2-2y+1)=0$$

is an inductive invariant. For instance, consider the initial step $(y=y_0, x=1)$*. A possible invariant is*

$$y_0(1-y_0)x^2+xy-x+y^2-2y+1=0.$$

□

3.6 Obtaining optimal degree bounds for discrete transition systems

To guarantee the existence of non-trivial invariants, we are looking for a polynomial T such that

$$Ker(M - L) \neq 0.$$

The pseudo code depicted in Algorithm 1 illustrates the strategy. Algorithm 1 is in a standard form, but its contribution relies on very general sufficient conditions for the existence and the computation of invariants. As input we have r, the candidate degree for the set of basis invariants elements, and $P_1, ..P_n$, the n polynomials given by the transition relation in considered loop.

We first compute d, the maximal degree of the P_i's as can be seen by `Max_degree(`$\{P_1, ..., P_n\}$`)`, at line 4. Then, we detail the cases were the transitions are defined by non linear systems, i.e. $d \leq 2$.

Then we define T as a polynomial of degree $dr - r$ in its canonical form,*i.e.* with parametrized coefficients. See `Template_Canonical_Form(`$dr - r$`)`, at line 7.

We can then build a decision procedure to assign values to the coefficients of T in such a way that $Ker(M - L) \neq 0$. As we saw in the previous section, $Ker(M - L) \neq 0$ is equivalent to having

$$Rank(M - L) < Dim(\mathbb{R}_r[X_1, \dots, X_n]).$$

In other words, it is equivalent to having $M - L$ with rank strictly less than the dimension $v(r)$ of $\mathbb{R}_r[X_1, \dots, X_n]$. We then reduce the rank of $M - L$ by assigning values of terms in M to parameters in L.
See `Reduce_Rank_Assigning_Values(`$M - L$`)`, at line 10. By so doing we can zero or identify some columns or lines of $M - L$. Next, we determine whether the matrix obtained, $\overline{M - L}$, has a trivial kernel by first computing its rank and then checking if $(\mathbf{Rank}(\overline{M - L}) < \mathbf{Dim}(R_r[X_1, .., X_n]))$ holds. See line 11.

In the case where $\overline{M - L}$ has a trivial kernel, we can increase the degree r of invariants until Theorem 5 (or Corollary 3) applies, or until stronger hypotheses occur, *e.g.* if all $v(r) \times v(r)$ sub-determinants are null. See `return Ideal_Loop_Inv_Gen(`$r + 1, P_1, ..., P_n, X_1, ..., X_n$, at line 12.

If there is no ideal for non-trivial invariants for a value r_i then we conclude that there is no ideal of non-trivial invariants for all degrees $k \leq r_i$. This can also be used to guide other constraint-based techniques, since checking for invariance with a template of degree less or equal to r_i will not be necessary. Otherwise, we compute and output the basis of the nullspace of matrix $\overline{M - L}$, in order to construct an ideal basis for non trivial invariants. See `Nullspace_Basis`, at line 15. For the latter, we use well-known state-of-the-art algorithms, for example those that Mathematica provides. These algorithms calculate the eigenvalues and associated eigenspaces of $\overline{M-L}$ when it is a square matrix. When $\overline{M-L}$ is a rectangular matrix, we can use its *singular value decomposition* (SVD). A SVD of $\overline{M-L}$ provides an explicit representation of its rank and kernel by computing unitary matrices U and V and a regular diagonal matrix S such that

$$\overline{M-L} = USV.$$

We compute the SVD of a $v(r+d-1)\times v(r)$ matrix $\bar{M}$ by a two step procedure. First, we reduce it to a bi-diagonal matrix, with a cost of $O(v(r)^2 v(r+d-1))$ flops. The second step relies on an iterative method, as is also the case for other eigenvalue algorithms. In practice, however, it suffices to compute the SVD up to a certain precision, *i.e.* up to a machine epsilon. In this case, the second step takes $O(v(r))$ iterations, each using $O(v(r))$ flops. So, the overall cost is $O(v(r)^2 v(r+d-1))$ flops. It is standard to try to generate invariantes at each step associated to a guessing r value. State-of-the-art methods require several step in doubly exponential complexity for each choosen r value while our methods only require a polynomial step.

For the implementation of the algorithm we could rewrite Corollary 3 as follows.

Corollary 4. *Let $\overline{M-L} = U\cdot S\cdot V$ be the singular value decomposition of matrix $\overline{M-L}$ described just above. There will be a non trivial T-invariant for any given initial condition if and only if the number of non-zero elements in matrix S is less than $v(r)-2$, where $v(r)$ is the dimension of $\mathbb{R}_r[x_1,\ldots,x_n]$. Moreover, the orthonormal basis for the nullspace obtained from the decomposition directly gives an ideal for non-linear invariants.* □

It is important to emphasize that eigenvectors of $\overline{M-L}$ are computed after the parameters of L_T have been assigned. When the discrete transition system has several variables and none or few parameters, which correspond to practical cases, $\overline{M-L}$ will be over the reals and there will be no need to use the symbolic version of these algorithms.

3.7 Invariant generation for discrete transitions with fractional systems

We now want to deal with transition systems ρ_τ of the following type:

$$\begin{bmatrix} X_1' = \frac{P_1(X_1,..,X_n)}{Q_1(X_1,..,X_n)} \\ \vdots \\ X_n' = \frac{P_n(X_1,..,X_n)}{Q_n(X_1,..,X_n)} \end{bmatrix}, \tag{3.6}$$

Algorithm 1: **Ideal_Loop_Inv_Gen**($r, P_1, ..., P_n, X_1, ..., X_n$)

```
/*Guessing the degree bounds for discrete transitions.*/
Data: $r$ is the candidate degree for the set of basis invariants elements we are
      looking for, $P_1, ..P_n$ the $n$ are polynomials given by the considered loop, and
      $X_1, ..X_n \in V$
Result: $Ideal\_Inv$, a basis of ideal of invariants.
begin
1     int $d$
2     Template $T$
3     Matrix $M$, $L$
4     $d \longleftarrow$ Max_degree($\{P_1, ..., P_n\}$)
5     /*$d$ is the maximal degree of $P_i$'s*/
6     if $d >= 2$ then
7         $T \longleftarrow$ Template_Canonical_Form($dr - r$)
8         $M \longleftarrow$ Matrix_D($r, dr, P_1, ..., P_n$)
9         $L \longleftarrow$ Matrix_L($r, dr, T$)
10        $\bar{M - L} \longleftarrow$ Reduce_Rank_Assigning_Values($M - L$)
11        if Rank($\bar{M}$) >= Dim($R_r[X_1, .., X_n]$) then
12            return Ideal_Loop_Inv_Gen($r + 1, P_1, ..., P_n, X_1, ..., X_n$)
13            /*We need to increase the degree r of candidates invariants.*/
14        else
15            return Nullspace_Basis($\bar{M - L}$)
16            /*There exists an ideal of invariants that we can compute*/
17    else
18        ... /*We refer to our previous work for constant scaling.*/
end
```

where the P_i's and Q_i's belong to $\mathbb{R}[X_1,..,X_n]$ and the P_i is relatively prime to Q_i.

This case, one needs to relax the consecution conditions to fractional-scale as soon as fractions appear in the transition relation.

Theorem 8. *(F-scale invariant characterization) A polynomial Q in $\mathbb{R}[X_1,..,X_n]$ is a F-scale invariant for fractional discrete scale consecution with a* parametric fractional *$F \in \mathbb{R}(X_1,..,X_n)$ for τ if and only if*

$$Q\left(\frac{P_1}{Q_1},..,\frac{P_n}{Q_n}\right) = FQ.$$

□

Demonstração. If $Q(X_1',..,X_n') - FQ(X_1,..,X_n)$ belongs to the fractional ideal J generated by the family $(X_1' - P_1/Q_1,\dots,X_n' - P_n/Q_n)$, then there exists a family $(A_1,\dots,A_n)$ of fractional functions in $K(X_1',..,X_n',X_1,..,X_n)$ such that

$$Q(X_1',..,X_n') - FQ(X_1,..,X_n) = (X_1' - P_1/Q_1)A_1 + \cdots + (X_n' - P_n/Q_n)A_n.$$

Letting $X_i' = P_i/Q_i$, we obtain $Q(P_1/Q_1,..,P_n/Q_n) = \lambda Q(X_1,\dots,X_n)$. Conversely, suppose that $Q(P_1/Q_1,..,P_n/Q_n) = FQ(X_1,..,X_n)$. Then, as $Q(X_1',..,X_n')$ is equal to $Q(P_1/Q_1,..,P_n/Q_n)$ modulo the ideal J, we get $Q(X_1',..,X_n') = FQ(X_1,..,X_n)$ modulo J. □

Let d be the maximal degree of the P_i's and Q_i's, and let Π be the least common multiple (lcm) of the Q_i's. Now let $U = {X_1}^{i_1}..{X_n}^{i_n}$ be a monomial of degree less than r (i.e. $i_1 + .. + i_n \leq r$). Then,

$$\Pi^r U(P_1/Q_1,\dots,P_n/Q_n) = \Pi^r (P_1/Q_1)^{i_1}...(P_n/Q_n)^{i_n}.$$

But as $Q_j^{i_j}$ divides Π^{i_j}, for all j, we see that $Q_1^{i_1}...Q_n^{i_n}$ divides $\Pi^{i_1+\dots+i_r}$, which divides Π^r. We deduce that $\Pi^r Q(P_1/Q_1,\dots,P_n/Q_n)$ is a polynomial for every Q in $\mathbb{R}_r[X_1,..,X_n]$.

Now suppose that $F = T/S$, with T relatively prime to S, satisfies the equality of the previous theorem. Suppose, further, that we are looking for bases for invariants Q of degree r. Then, multiplying by Π^r we get

$$\Pi^r Q(P_1/Q_1,\dots,P_n/Q_n) = (\Pi^r TQ)/S.$$

As we have no "a priory" information on Q, in most of the cases Q will be relatively prime to S. In this case we see that S will divides Π^r, and we can suppose that it has denominator Π^r. So, let F be of the form T/Π^r, and we just argued that this constraint is weak.

Now let $\mathscr{M}$ be the morphism

$$\mathscr{M} : \begin{cases} \mathbb{R}_r[X_1, \ldots, X_n] & \to & \mathbb{R}_{nrd}[X_1, \ldots, X_n] \\ Q & \mapsto & \Pi^r Q(\frac{P_1}{Q_1}, .., \frac{P_n}{Q_n}) \end{cases}$$

And let M be its matrix in the canonical bases. Let T be a polynomial in $\mathbb{R}_{nrd-r}[X_1, .., X_n]$ and let $\mathscr{L}$ denote the morphism of vector spaces

$$\mathscr{L} : \begin{cases} \mathbb{R}_r[X_1, \ldots, X_n] & \to & \mathbb{R}_{nrd}[X_1, \ldots, X_n] \\ Q & \mapsto & TQ \end{cases}$$

Also, let L be its matrix in the canonical bases. As we show in the following theorem, our problem is equivalent to finding a L such that $M - L$ has a non trivial kernel.

Theorem 9. *Consider M and L as described above. Then, there exists F-scale invariants, where F is of the form T/Π^r, if and only if there exists a matrix L such that $Ker(M-L) \neq \emptyset$. In this situation, any vector in the kernel of $M - L$ will give rise to a F-scale discrete invariant.* □

This is similar to Theorems 6 and 7. For the initiation step, we have a hyperplane in V_r. In order for the transition system to make sense, the n-tuple of initial values must not be a root of any of the Q_i's, and so must be their iterates, as long as the loop is applied. In this way, they will not cancel Π^r.

We have the following:

Theorem 10. *(Non trivial invariants using Fractional scale consecution) We have a non trivial invariant if and only if there exists a matrix L such that the intersection of the kernel of $M - L$ and the hyperplane given by the initial values is not zero, the invariants corresponding to vectors in the intersection.* □

We also have the following important corollary.

Corollary 5. *(Non trivial invariants using Fractional-scale consecution and for any initial value) We will have a non-trivial invariant for any non-trivial initial value if there exists a matrix L such that the dimension of $Ker(M - L)$ is at least 2.* □

Example 11. *(Running example)) Consider the system*

$$\rho_\tau \equiv \begin{bmatrix} x_1' = \frac{x_2}{(x_1+x_2)} \\ x_2' = \frac{x_1}{(x_1+2x_2)} \end{bmatrix}. \tag{3.7}$$

We are looking for F-scale invariant polynomials of degree 2. The lcm of $(x_1 + x_2)$ and $(x_1 + 2x_2)$ is their product, so that $\mathscr{M}$ is given by:

$$Q \in \mathbb{R}_2[x_1, x_2] \mapsto [(x_1 + x_2)(x_1 + 2x_2)]^2 Q(\frac{x_1}{(x_1 + x_2)}, \frac{x_2}{(x_1 + 2x_2)})].$$

As both $\frac{x_2}{(x_1+x_2)}$ *and* $\frac{x_1}{(x_1+2x_2)}$ *have "degree" zero,*

$$[(x_1+x_2)(x_1+2x_2)]^2 Q(\frac{x_2}{(x_1+x_2)}, \frac{x_1}{(x_1+2x_2)})$$

will be a linear combination of degree 4, if it is non null.

Hence, $\mathscr{M}$ *has values in*

$$Vect(x_1^4, x_1^3x_2, x_1^2x_2^2, x_1x_2^3, x_2^4).$$

with T *and* Q *in* $\mathbb{R}_2[x_1,x_2]$ *we verify that*

$$[(x_1+x_2)(x_1+2x_2)]^2 Q(\frac{x_2}{(x_1+x_2)}, \frac{x_1}{(x_1+2x_2)}) = TQ.$$

As the left member is in $Vect(x_1^4, x_1^3x_2, x_1^2x_2^2, x_1x_2^3, x_2^4)$*,* T *must be of the form*

$$\lambda_0 x_1^2 + \lambda_1 x_1 x_2 + \lambda_2 x_2^2$$

and Q *of the form*

$$a_0x_1^2 + a_1x_1x_2 + a_3x_2^2.$$

We see that we can take Q *in* $Vect(x_1^2, x_1x_2, x_2^2)$*, and similarly for* T*.*

Then both $\mathscr{M}$*,* $\mathscr{L} : (Q \mapsto TQ)$ *are morphisms from*

$$Vect(x_1^2, x_1x_2, x_2^2)$$

in

$$Vect(x_1^4, x_1^3x_2, x_1^2x_2^2, x_1x_2^3, x_2^4).$$

In the corresponding canonical basis, the matrix $M - L$ *is*

$$M - L = \begin{pmatrix} -\lambda_0 & 0 & 1 \\ -\lambda_1 & 1-\lambda_0 & 2 \\ 1-\lambda_2 & 3-\lambda_1 & 1-\lambda_0 \\ 4 & 2-\lambda_2 & -\lambda_1 \\ 4 & 0 & -\lambda_2 \end{pmatrix}.$$

Taking $\lambda_0 = 1, \lambda_1 = 3$ *and* $\lambda_2 = 2$ *cancels the second column and so, the kernel will be equal to* $Vect(0,1,0)$*. Now, Corollary 5 applies to* $M - L$*:*

```
Fractional scaling discrete step
T(x,y) / Q(x,y) = 1 / ((x + y) (x + 2 y))^2
Module of degree 3 and rank 1 and Kernel of dimension 2
{{0, 1, 0}}
Basis of invariant Ideal
{ x y }
```

It was clear from the beginning that the corresponding polynomial x_1x_2 is $\frac{1}{[(x_1+x_2)(x_1+2x_2)]^2}$-scale invariant.

In particular, it is an invariant for the initial values $(0,1)$. Moreover, it clearly never cancels x_1+x_2 and x_1+2x_2, because they are of the form $(a,0)$ or $(0,b)$ with a and b strictly positive. □

3.8 Branching conditions and nested loops

We have generated a basis of a vector space which describes invariants for transitions. A global invariant would be any invariant which is in the intersection of these vector spaces. In this way, we avoid the definition of a single isomorphism for the whole transitions system. Instead, we generate the basis for each separate consecution conditions.

To compute the basis of global invariants, we could use the following Theorem. It proposes to *multiply* all the elements of each computed basis. By so doing, we also avoid the heavy computation of ideal intersections This approach is a sound, but not complete, way of computing ideals for global invariants, and it has a low computational complexity. In order to take into account initial conditions we could intersect these vector spaces of invariants with the initial hyperplanes deduced from the isomorphism associated with initial requirements.

Here, we show how our method deals with the conditional statements inside loops. Let's consider the following type of loop :

```
...
while(B_1){
  [I_1;]
  if(B_2){
          [I_2;]
        }
  else{
       [I_3;]
     }
  [I_4;]
}
...
```

where I_is are notations that represent a block of multivariate fractional instructions. First we represent the loop with the following two transitions

$$\tau_1 = \langle l_i, l_i, (\mathcal{B}_1 \wedge \mathcal{B}_2), \rho_{\tau_1}\rangle$$

and

$$\tau_2 = \langle l_i, l_i, (\mathcal{B}_1 \wedge \neg\mathcal{B}_2), \rho_{\tau_2}\rangle,$$

where:

$$\rho_{\tau_1} \equiv [x_1' = F_{1,[I_1;I_2;I_4;]_\circ}(x_1,...,x_n), \dots, x_n' = F_{n,[I_1;I_2;I_4;]_\circ}(x_1,...,x_n)]$$

and

$$\rho_{\tau_2} \equiv [x'_1 = F_{1,[I_1;I_3;I_4;]_\circ}(x_1, ...x_n) \ldots, x'_n = F_{n,[I_1;I_3;I_4;]_\circ}(x_1, ..., x_n)]$$

with $[;;]_\circ$ denoting our operator based on separation rewriting rules to compose blocks of instructions.

We first independently generates the ideals of invariants $\xi_1 = (\mu_1, ..., \mu_n)$ and $\xi_2 = (\kappa_1, ..., \kappa_p)$ for the respective transitions τ_1 and τ_2. Any element $\mu_i \in \xi_1$ refers to an inductive invariant $\mu_i(X_1, ..., X_n) = 0$ corresponding to the *partial loops* described by transition τ_1. Similarly, any $\kappa_i \in \xi_2$ refers to an inductive invariant $\kappa_i(X_1, ..., X_n) = 0$ for the loop described by the transition τ_2.

Then we can take

$$\mu_i(X_1, ..., X_n) * \kappa_i(X_1, ..., X_n) = 0$$

as global loop invariants, since these invariant will remind true in any sequences of transitions during the execution of the loop.

Theorem 11. *Let $I = \{I_1, ..., I_k\}$ a set of ideals in $\mathbb{R}[X_1, ..., X_n]$ such that $I_j = (f^{(j)}{}_1, ..., f^{(j)}_{n_j})$ where $j \in [1, k]$. Let*

$$\otimes(I_1, ..., I_k) = \{\delta_1, ..., \delta_{n_1 n_2 \ldots n_k}\}$$

be such that all elements δ_i in $\otimes(I_1, ..., I_k)$ are formed by the product of one element from each ideal in I. Assume that all I_j's are ideals for invariants for a loop at location l_j, described by a transition τ_j. Now, if all l_j describe the same location or program point l, then we have several transitions looping *at the same point. Thus we can obtain an encoding of possible execution paths of a loop containing conditional statements.*

It is clear then that $\otimes(I_1, ..., I_k)$ is an ideal of non-trivial non-linear invariants for the entire loop located at l. □

We deal with loop conditions using the same methods that we propose to handle initiation conditions. We know, for instance, that if our Corollary 3 holds, then there exists invariants for any (semi-)hyperplane that could be induced by the loop conditions. We illustrate this point in figure 3.1.

Let $(P_i(x_1, .., x_n) < 0)$ be semi-algebraic loop conditions at location l and let Q be an inductive invariant for $\mathcal{D}(l)$. Thus $(P_i(x_1, .., x_n) - Q(x_1, .., x_n) < 0)$ is also an inductive invariant. Then, we can build an operator, similar to the one introduced in Theorem 11, to generate, in a different way, ideals of non-trivial invariants at a state l with semi-algebraic loop conditions.

Example 12. *Consider the following loop. An example of an invariant generated is $xyu_0z^2 - u_0^2z^2 + xyzu + xyu^2 - xyz - 2xyu + xy = 0$*

```
//initialization
       ...
int u_0;
       ...
((M > 0)&&(Z = 1)&&(U = u_0)...)
       ...
  While ((X>=1) || (Z>=z_0)){
       ...
       If(Y > M){
            X = Y / (X + Y);
            Y = X / (X + 2 * Y);
       }
       Else{
           Z = Z * (U + 1);
           U = U^2;
       }
  }
   ...
```

*Once again, here there are no need for Grobner Basis computation and the complexity of the steps described remain linear. Example 12 illustrate our method for the case where the loop contains two conditional statements. We first generate an invariant for the loop corresponding to the first conditional "**if**", at line* 6*, using Fractional-Scaling.*

```
If_1 :
Fractional scaling discrete step
T(x,y) / Q(x,y) = 1 / ((x + y) (x + 2 y))^2
...
Basis of invariant Ideal
{ x y }
...
```

See Example 11 for more details. Then we compute the invariant

```
Else_1 :
...
Basis of invariant Ideal
{ u_0z^2-u_0^2z^2+zu+u^2-z-2u+1, ... }
...
```

*corresponding to the other alternative transition τ_2 of the loop. See "**Else**", at line* 10*. The **Join** operator only returned the product of the two previously computed invariants:*

```
While_1 :
...
{ xyu_0z^2-u_0^2z^2+xyzu+xyu^2-xyz-2xyu+xy, ... }
...
```

□

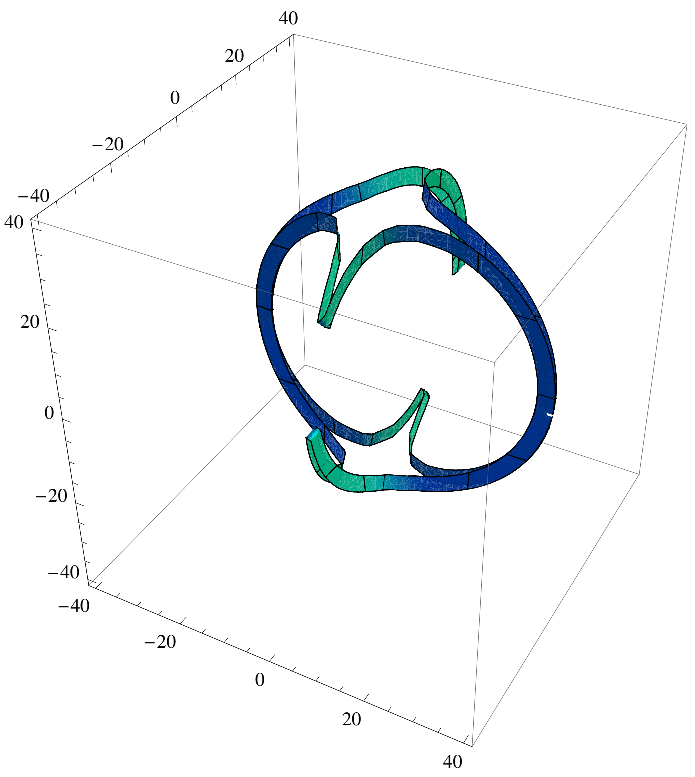


Figura 3.1: Intersection between the conditional loop: $800 < (x-5)^2+(y-5)^2+(y_0-5)^2 < 1000$ and the invariant $y_0(1-y_0)x^2+xy-x+y^2-2y+1=0$ from the invariant ideal $(\{x^2, xy-x, y^2-2y+1\})$ computed for the running example 10.

In the presence of nested loops, our method generates ideals for invariants for each inner-loop and then generates a global invariant.

3.9 Chapter Discussions: Performances and Limitations

When using previous modern approaches [120, 10, 109, 113, 24, 73, 72, 34, 96, 110, 114, 104] based on fixed point computation, or on the constraint-based approaches, the computation of Gröbner Bases, quantifier elimination, cylindrical algebraic decomposition, direct resolution of algebraic systems or the construction of abstraction operators are required. The notions of Gröbner Bases and their computations, and the *ideal membership problem* are at the heart of most recent approaches. Focusing on the most recent and best performing techniques [120, 109, 69, 110, 118, 114, 105], one could synthesize these constraint-based approaches in three main steps. But, to understand the difficulties they raise, we need first to give more details on Gröbner Bases and the *ideal membership problem.*

Consider a n multivariate polynomial, $Q = \sum_{i_1,..,i_n} a_{i_1,..,i_n} {x_1}^{i_1}...{x_n}^{i_n}$, where $a_{i_1,...,i_n}$ are in a field K. How do we know if p is in an ideal I of $K[X_1, ..., X_n]$? This is *Ideal membership problem.* We can consider the Gröbner Bases $G = \{g_1, ..., g_s\}$ of I. There exist an algorithm [21, 46] that compute such bases as long as we know a finite generating bases of I. We can compute the normal form of Q for I using a Gröbner Bases G. Denote the normal form by $NF_G(Q)$. Gröbner Bases are used because they are bases that guarantee the confluence and termination of those reductions. Then $NF_G(Q) = \sum_{i_1,..,i_n} f(a)_{i_1,..,i_n} {x_1}^{i_1}...{x_n}^{i_n}$, here $f(a)_{i_1,..,i_n}$ are combinations of $a_{i_1,..,i_n}$. Then $(Q \in I)$ is equivalent to $(NF_G(Q) = 0)$, in other words all the coefficients $f(a)_{i_1,...,i_n}$ are null. Known algorithms for computing the Gröbner bases G and the algorithms performing the normal form reduction $NF_G(Q)$ to Q are of doubly exponential complexity.

In the mentioned approaches, the loop instructions are considered in order to form varieties and to build associated algebraic assertions and its generated ideal I (see Section 2.3 in Chapter 2). In a first step, these techniques compute the Gröbner Bases G of I. Then, they consider a template polynomial Q (i.e., a polynomial with unknown coefficients) as the *candidate invariant.* As we have seen just above, Q is an invariant if it belongs to the ideal I, in other words if $(NF_G(Q) = 0)$. So, in the second step of these techniques, the reductions $NF_G(Q)$ is performed and a equation systems $(NF_G(Q) = 0)$ is generated. This set of constraints form the *candidate invariant constraints.* Finally, they attempt to solve directly the candidate invariant constraint systems. We clearly see that each single step of these approaches induce a high numbers of computations that are of double exponential complexity. Also, there are no conditions that is identified over the degree of their candidate invariants, and that would guarantee the non-triviality of the resulting invariant, when it can be computed. Moreover, we have shown (see Section 3.4.3) that as soon as the loop contains a non-linear instruction, the constraints considered in their final step is a non-linear equation systems, which makes their resolution unfeasible.

In terms of performance and efficiency, we succeeded in reducing the non-linear loop invariant generation problem to a linear algebraic problem, i.e. the computation of eigenspaces of specific morphisms. Our techniques have few computational steps: we compute first some specific matrices and we then compute their nullspaces. Each step performed by our techniques remains of polynomial complexity. Further, our approaches do not generate an invariant at a time. Instead we generate an ideal of invariants which is an enormous (infinite) structure. We also handle fractional systems and our algorithm incorporates a strategy to guess degree bounds which allow for the automatic generation of ideals of non-trivial invariants. Moreover, as one of the main results, we provide very general sufficient conditions allowing for the existence and computation of invariant ideals. Note that these conditions could be directly used by any invariant generation method.

In terms of limitations, our techniques concern logic with non-linear (or linear) arithmetic. In order to be considered as a verification tool for complex software one would need to extend the logic and consider their associated formal methods in order to handle more complex instructions. To statically analyse complex software one would need to consider several different logics and combine their associated independent formal methods and tools. For instance, one need heap logic and their formal methods to handle recursive data structures and pointers. And we would need to consider another logic and its associated formal methods in order to handle array or threads. Also these static analysis methods and their associated automated proof methods applied first on algorithm than software. As it is done in other standard program verification techniques on need to adapt the methods to variables over the reals dealing with overflow properties.

One of the main challenge for the formal methods community is to first identify a logic for each type of semantics induced by a software. Then one needs to develop formal methods independently for each logic and finally propose techniques to composed them in a tool-bus. In Chapter 6, our techniques are combined with other formal methods and their associated tools. We compose our techniques with formal methods restricted to logics with uninterpreted functions, that is, a logics handling function calls and operating system calls. Our primary goal and motivation are to provide invariant generation methods for static analysis. Also, the methods proposed in this chapter could be extended to probalistic programs and timed automata. On the other hand, our secondary motivation is to propose a new domain of applications for invariant generations, one that requires the computation of complex invariants. In Chapter 6 we generate invariants over malwares that are used as strong semantic aware signatures that can be used to analyse and identify these malwares. In this context we are considering binary codes (where structure such as pointers and array do not appear, for instance). These binary code induce a logic with non-linear arithmetic and the methods described here allow for the generations of complex and precise invariants. And the more the invariant, i.e. signature generated, is complex all the better it is as it will be harder to morph such signatures in an automatic way.

Capítulo 4

Generating Invariants for Non-linear Hybrid Systems

The concepts and contributions exposed in this chapter are also introduced in our publications [83, 88, 84, 81].

Abstract: We describe powerful computational techniques, relying on linear algebraic methods, for generating ideals of non-linear invariants of algebraic hybrid systems. We show that the preconditions for discrete transitions and the Lie-derivatives for continuous evolution can be viewed as morphisms, and so can be suitably represented by matrices. We reduce the non-trivial invariant generation problem to the computation of the associated eigenspaces by encoding the new consecution requirements as specific morphisms represented by such matrices. More specifically, our methods are the first to establish very general sufficient conditions that show the existence and allow the computation of invariant ideals. Our methods also embody a strategy to estimate certain degree bounds, leading to the discovery of rich classes of inductive, *i.e.* provable, invariants. By reducing the problem to related linear algebraic manipulations we are able to address various deficiencies of other state-of-the-art invariant generation methods, including the efficient treatment of non-linear hybrid systems. Our approach avoids first-order quantifier elimination, Gröbner basis computation or direct system resolution, thereby circumventing difficulties met by other recent techniques.

4.1 Introduction

Hybrid systems [63, 4] exhibit both discrete and continuous behaviors, as one often finds when modeling digital system embedded in analog environments. Most safety-critical systems, *e.g.* aircraft, automobiles, chemicals and nuclear power plants, biological systems, operate semantically as non-linear hybrid systems. As such, they can only be adequately modeled by means of non linear arithmetic over the real numbers involving multivariate polynomials and fractional or transcendental functions.

The analysis of hybrid systems has been one of the main challenges for the formal verification community for several decades. Some verification approaches for treating such models are based on inductive invariant generation methods [77, 126] and also on the Abstract Interpretation framework [36, 35], combined with the reduction of safety-critical properties to invariant properties [64, 9]. We look for invariants that strengthen what we wish to prove, and so allow us to establish the desired properties. Also, they can provide precise over-approximations of the set of reachable states in the continuous state space.

More recent approaches have been constraint-based [118, 114, 105, 1, 115]. In these cases, a candidate invariant with a fixed degree and unknown parametric coefficients, *i.e.*, a template form, is proposed as the target invariant to be generated. The conditions for invariance are then encoded, resulting in constraints on the unknown coefficients whose solutions yield invariants. One of the main advantage of such constraint-based approaches is that they are goal-oriented. But, on the other hand, they still require the computation of several Gröbner Bases [21] or require first-order quantifier elimination [131, 32]. And known algorithms for those problems are, at least, of double exponential complexity. SAT Modulo Theory decision procedures and polynomial systems [16, 53, 114] could also, eventually, lead to decision procedures for linear theories and, thus, to decidable systems.

Such ideas strive to generate linear or polynomial invariants over hybrid systems that exhibit affine or polynomial systems as continuous evolution modes. Nonetheless, despite tremendous progress over the years [118, 125, 111, 114, 1, 124, 8, 52, 3, 16, 115, 104], the problem of invariant generation for hybrid systems remains very challenging for both non-linear discrete systems as well as non-linear differential systems with non abstracted local and initial conditions.

In this work we use hybrid automata as computational models for hybrid systems. A hybrid automaton describes the interaction between the discrete transitions and the continuous dynamics, the latter being governed by local differential equations. We present new methods for the automatic generation of non-linear invariants for non-linear hybrid systems. These methods give rise to more efficient algorithms, with much lower complexity in space and time.

First, we extend and generalize our previous work on invariant generation for hybrid systems [84, 83, 88]. To do so, we provide methods to generate non trivial basis of provable invariants for local continuous evolution modes described by non linear differential rules. These invariants can provide precise over-approximations of the set of reachable states in the continuous state space. As a consequence, they can determine which discrete transitions are possible and can also verify if a given property is fulfilled or not.

Next, in order to generate invariants for hybrid systems, we complete and extend our previous work on non linear invariant generation for discrete programs [80, 79]. The contribution and novelty in our approaches clearly differ from those in [118] as their constraint-based techniques are based on several Gröbner Basis or Syzygy Basis [119], computations and on solving non linear problems for each location. Nevertheless, they introduce a useful formalism to treat the problem, and we start from similar definitions for hybrid systems, inductive invariants and consecution conditions.

We then propose methods to identify suitable morphisms to encode the relaxed consecution requirements. We show that the preconditions for discrete transitions and the Lie-derivatives for continuous evolutions can be viewed as morphisms over a vector space of terms, with polynomially bounded degrees, which can be suitably represented by matrices. The relaxed consecution requirements are also encoded as morphisms represented by matrices. By so doing, we do not need to start with candidate invariants that generate intractable solving problems. Moreover, our methods are not constraint-based. Rather, we automatically identify the needed degree of a generic multivariate polynomial, or fractional, as a relaxation of the consecution condition. The invariant basis are, then, generated by computing the Eigenspace of another matrix that is constructed. We identify the needed approximations and the relaxations of the consecution conditions, in order to guaranteed sufficient conditions for the existence and computation of invariants. Moreover, the unknown parameters that are introduced are all fixed in such a way that certain specific matrices will have a non null kernel, guaranteeing a basis for non-trivial invariants.

The contribution of this chapter are summarized in Section 1.2.2. In Section 4.2 we introduce ideals of polynomials, inductive assertions and algebraic hybrid systems. In Section 4.3 we present new forms of approximating consecution for non-linear differential systems. In Section 4.5, we discuss morphisms suitable to handle non-linear differential rules and show how to generate invariants for differential rules. In Section 4.6 we introduce a strategy that can be used to choose the degree of invariants. Section 4.7 presents some experiments. In Section 4.8, we show how to generate ideals for global invariants by taking into account the ideal basis of local differential invariants, together with those derived from the discrete transition analysis and the initial constraints. We present our conclusions in Section 4.9.

In this writing, we strive to precede the most important proofs by sketches. Full proofs, more details and examples can be found in [84, 81, 79, 80].

4.2 Algebraic Hybrid Systems and Inductive Assertions

For the abstract algebraic definitions of field, multivariate polynomials and polynomial ring we refer to Section 2.2 in Chapter 2. The definitions of ideal and its basis, we refer to definitions 4 and 5 from Section 2.3 in Chapter 2.

The linear algebraic notions of morphism and nullspace (i.e., Kernel) are very important for this Chapter and in this chapter and one could found their presentation in Section 2.4 in Chapter 2.

Let $\mathbb{K}[X_1, .., X_n]$ be the ring of multivariate polynomials over the set of variables $\{X_1, .., X_n\}$.

Notationally, as is standard in static program analysis, a primed symbol x' refers to next state value of x after a transition is taken. We may also write $\dot{x}$ for the derivative $\frac{dx}{dt}$.

We denote by $\mathbb{R}_d[X_1, .., X_n]$ the ring of multivariate polynomials over the set of real variables $\{X_1, .., X_n\}$ of degree at most d.

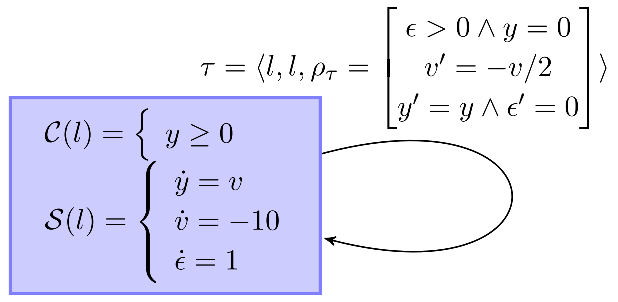
We write $Vect(v_1, ..., v_n)$ for the vectorial space generated by the basis $v_1, ..., v_n$.

Definition 17. *A* hybrid system *is described by a tuple $\langle V, V_t, L, \mathcal{T}, \mathcal{C}, \mathcal{S}, l_0, \Theta\rangle$, where*

- $V = \{a_1, .., a_m\}$ *is a set of parameters,*
- $V_t = \{X_1(t), .., X_n(t)\}$ *where $X_i(t)$ is a function of t,*
- L *is a set of locations and*
- l_0 *is the initial location.*
- *A* transition $\tau \in \mathcal{T}$ *is given by $\langle l_{pre}, l_{post}, \rho_\tau\rangle$, where l_{pre} and l_{post} name the pre- and post- locations of τ, and the* transition relation ρ_τ *is a first-order assertion over* $V \cup V_t \cup V' \cup V_t'$.
- *Also, Θ is the initial condition, given as a first-order assertion over $V \cup V_t$,*
- *and $\mathcal{C}$ maps each location $l \in L$ to a* local condition $\mathcal{C}(l)$ *denoting an assertion over* $V \cup V_t$.
- *Finally, $\mathcal{S}$ associates each location $l \in L$ to a* differential rule $\mathcal{S}(l)$ *corresponding to an assertion over* $V \cup \{dX_i/dt | X_i \in V_t\}$.
- *A* state *is any pair (location and interpretation of the variables) from* $L \times \mathbb{R}^{|V|}$.

□

Example 13. *The dynamic system of a bouncing ball ([121]) can be modeled by the following hybrid automaton:*


$\tau = \langle l, l, \rho_\tau = \begin{bmatrix} \epsilon > 0 \wedge y = 0 \\ v' = -v/2 \\ y' = y \wedge \epsilon' = 0 \end{bmatrix} \rangle$
$\mathcal{C}(l) = \left\{ \; y \geq 0 \right.$
$\mathcal{S}(l) = \left\{ \begin{array}{l} \dot{y} = v \\ \dot{v} = -10 \\ \dot{\epsilon} = 1 \end{array} \right.$

$V = \{y, v, \epsilon\}$, $\Theta = (v = 16 \wedge y = \epsilon = 0)$, $l_0 = l$, $L = \{l\}$ *and* $\mathcal{T} = \{\tau\}$. □

The evolution of variables and functions in an interval must satisfy the local conditions and the local differential rules.

Definition 18. *A run of a hybrid automaton is an infinite sequence*

$$(l_0, \kappa_0) \to \cdots \to (l_i, \kappa_i) \to \cdots$$

of states where l_0 *is the* initial location *and* $\kappa_0 \models \Theta$.

For any two consecutive states $(l_i, \kappa_i) \to (l_{i+1}, \kappa_{i+1})$ *in such a run, the condition describes a* discrete consecution *if there exists a transition* $\langle q, p, \rho_i \rangle \in \mathcal{T}$ *such that* $q = l_i$, $p = l_{i+1}$ *and* $\langle \kappa_i, \kappa_{i+1} \rangle \models \rho_i$ *where the primed symbols refer to* κ_{i+1}.

Otherwise, it is a continuous consecution *condition and we must have* $q \in L$, $\varepsilon \in \mathbb{R}$ *and a differentiable function* $\phi : [0, \varepsilon) \to \mathbb{R}^{|V \cup V_t|}$ *such that the following conditions hold:*

- *(i)* $l_i = l_{i+1} = q$;
- *(ii)* $\phi(0) = \kappa_i$, $\phi(\varepsilon) = \kappa_{i+1}$;
- *(iii) During the time interval* $[0, \varepsilon)$, ϕ *satisfies the local condition* $\mathcal{C}(q)$ *and the local differential rule* $\mathcal{S}(q)$. *That is, for all* $t \in [0, \varepsilon)$ *we must have* $\phi(t) \models \mathcal{C}(q)$ *and* $\langle \phi(t), d\phi(t)/dt \rangle \models \mathcal{S}(q)$.

A state (ℓ, κ) *is reachable if there is a run and some* $i \geq 0$ *such that* $(\ell, \kappa) = (\ell_i, \kappa_i)$. □

Example 14. *Returning to Example 13, consider the run:*

$$(l, \kappa_0) \xrightarrow{\mu_0} (l, \kappa_1) \xrightarrow{\mu_1} (l, \kappa_2),$$

where $\kappa_0 = (0, 16, 0)$. *In a valuation* $(a, b, c) \in \mathbb{R}^3$, *a is the value of* y, *b is the value of* v *and c is the value of* ϵ. *Clearly,* $\kappa_0 \models \Theta$, *as required.*

Now take $\kappa_1 = (0, -16, \varepsilon)$, where $\varepsilon = \frac{16}{5}$, and consider $\phi : [0, \varepsilon] \to \mathbb{R}^{|V_t|}$ such that $\phi(t) = (y(t), v(t), \epsilon(t)) = (-5t^2 + 16t, -10t + 16, t)$. Then $\phi(0) = (0, 16, 0) = \kappa_0$ and $\phi(\varepsilon) = (y(\varepsilon), v(\varepsilon), \epsilon(\varepsilon)) = \kappa_1$. Further, for all $t \in [0, \varepsilon]$ we get $\phi(t) \models \mathcal{C}(q)$ because $y(t)$ is clearly non-negative for $t \in [0, \varepsilon]$. Also, for all $t \in [0, \varepsilon]$ we have $\langle \phi(t), d\phi(t)/dt \rangle \models \mathcal{S}(q)$ because
$d\phi(t)/dt = (dy(t)/dt, dv(t)/dt, d\epsilon(t)/dt) = (v, -10, 1)$. So, by construction, μ_0 is a possible continuous consecution.

Now, since $\langle (0, -16, \varepsilon), (0, 8, 0) \rangle \models \rho_\tau$, if we let $\kappa_2 = (0, 8, 0)$, then μ_1 is a discrete consecution. □

Definition 19. *Let W be a hybrid system. An assertion φ over $V \cup V_t$ is an* invariant *at $l \in L$ if $\kappa \models \varphi$ whenever (l, κ) is a reachable state of W.* □

Definition 20. *Let W be a hybrid system and let $\mathbb{D}$ be an assertion domain. An assertion map for W is a map $\gamma : L \to \mathbb{D}$. We say that γ is* inductive *if and only if the following conditions hold:*

1. ***Initiation**: $\Theta \models \gamma(l_0)$;*

2. ***Discrete Consecution**: for all $\langle l_i, l_j, \rho_\tau \rangle \in \mathcal{T}$ we have*

$$\gamma(l_i) \wedge \rho_\tau \models \gamma(l_j)';$$

3. ***Continuous Consecution**: for all $l \in L$, and two consecutive states (l, κ_i) and (l, κ_{i+1}) in a possible run of W such that κ_{i+1} is obtained from κ_i according to the local differential rule $\mathcal{S}(l)$,*

$$\text{if } \kappa_i \models \gamma(l) \text{ then } \kappa_{i+1} \models \gamma(l).$$

Note that if

$$\gamma(l) \equiv (P_\gamma(X_1(t), .., X_n(t)) = 0) \forall t \in [0, \varepsilon)$$

where P_γ is a multivariate polynomial in $\mathbb{R}[X_1, .., X_n]$ such that it has null values on the trajectory

$$(X_1(t), ..., X_n(t))$$

during the time interval $[0, \varepsilon)$ which do not implies that P_γ is the null polynomial, then

$$\mathcal{C}(l) \wedge (P_\gamma(X_1(t), .., X_n(t)) = 0) \models (d(P_\gamma(X_1(t), .., X_n(t))/dt = 0)$$

during the local time interval. □

Hence, if γ is an inductive assertion map then $\gamma(l)$ is an invariant at l for W.

Example 15. *Consider the hybrid system of Example 13. It is easy to verify that the assertion*

$$y = v \times \epsilon + 5 \times \epsilon^2$$

is a provable, inductive invariant. We can see that the assertion holds during discrete transitions and the continuous evolution. □

4.3 New continuous consecution conditions

Now we show how to encode differential continuous consecution conditions. Consider a hybrid automaton W. Let $l \in L$ be a location which could, eventually, be in a circuit, and let η be an assertion map such that

$$\eta(l) \equiv (P_\eta(X_1(t), .., X_n(t)) = 0)$$

, where P_η is a multivariate polynomial in $\mathbb{R}[X_1, .., X_n]$ such that it has null values on the local trajectory

$$(X_1(t), ..., X_n(t))$$

during the local time interval $[0, \varepsilon)$ which do not implies that P_η is the null polynomial.

We have

$$\frac{dP_\eta}{dt} = \frac{\partial P_\eta(X_1, \ldots, X_n)}{\partial X_1}\frac{dX_1(t)}{dt} + \cdots + \frac{\partial P_\eta(X_1, \ldots, X_n)}{\partial X_n}\frac{dX_n(t)}{dt}.$$

Definition 21. *For a polynomial P in $\mathbb{R}_d[X_1, .., X_n]$, we define the polynomial $\mathscr{D}_P$ of $\mathbb{R}_d[Y_1, .., Y_n, X_1, .., X_n]$:*

$$\mathscr{D}_P(Y_1, .., Y_n, X_1, .., X_n) = \frac{\partial P(X_1, .., X_n)}{\partial X_1}Y_1 + ... + \frac{\partial P(X_1, .., X_n)}{\partial X_n}Y_n.$$

□

Hence,

$$\frac{dP_\eta}{dt} = \mathscr{D}_{P_\eta}(\dot{X}_1, .., \dot{X}_n, X_1, .., X_n).$$

Now, let (l, κ_i) and (l, κ_{i+1}) be two consecutive configurations in a run. Then we can express local state continuous consecutions as

$$\mathcal{C}(l) \wedge (P_\eta(X_1(t), .., X_n(t)) = 0) \models (dP_\eta/dt = 0)$$

during the local time interval.

Definition 22. *Let W be a hybrid automaton, $l \in L$ a location and let η be an algebraic inductive map with $\eta(l) \equiv (P_\eta(X_1(t),..,X_n(t)) = 0)$ for all t in the time interval of mode l (so, P_η has a null value over the local trajectory $(X_1(t),..,X_n(t))$). We identify the following notions to encode* continuous consecution conditions*:*

- *η satisfies a differential* Fractional-*scale consecution at l if and only if there exists a multivariate fractional $\frac{T}{Q}$ such that*

$$\mathcal{C}(l) \models (dP_\eta/dt - \frac{T}{Q}P_\eta = 0).$$

  We say that P_η is a fractional-scale and a $\frac{T}{Q}$-scale differential invariant.

- *η satisfies a differential* Polynomial-*scale consecution at l if and only if there exist a multivariate polynomial T such that*

$$\mathcal{C}(l) \models dP_\eta/dt - TP_\eta = 0.$$

  We say that P_η is a polynomial-scale and a T-scale differential invariant.

- *η satisfies a differential* Constant-*scale consecution at l if and only if there exists a constant $\lambda \in \mathbb{R}\backslash\{0\}$ such that*

$$\mathcal{C}(l) \models (dP_\eta/dt - \lambda P_\eta = 0).$$

  We say that P_η is a constant-scale and a λ-scale differential invariant.

- *η satisfies a differential* Strong-*scale consecution at l if and only if*

$$\mathcal{C}(l) \models (dP_\eta/dt = 0).$$

  Also, we say that P_η is a strong-scale differential invariant.

□

Differential Polynomial-scale consecution encode the fact that the numerical value of the Lie derivative of the polynomial P_η associated with assertion $\eta(l)$ is given by T times its numerical value throughout the time interval $[0, \varepsilon]$.

In [83, 88] we proposed methods for T-scale invariant generation where T is a constant (constant-scaling) or null (strong-scaling).

As can be seen, the consecution conditions are relaxed when going from strong to polynomial scaling.

Also, the T polynomials can be understood as *template multiplicative factors.* In other words, they are polynomials with unknown coefficients.

In the next section, we consider polynomial-scale consecution and then we could extend the methods to fractional-scale consecution conditions, as is done in Section 3.7 for discrete steps.

In later sections we show how to combine these conditions with others induced by discrete transitions. In [81], [79] one can find more details on how to handle other constraints associated to locations.

Theorem 12. *(Soundness) Let P be a continuous function and let*

$$\mathcal{S} = \begin{bmatrix} \dot{X}_1(t) = P_1(X_1(t), .., X_n(t)) \\ \vdots \\ \dot{X}_n(t) = P_n(X_1(t), .., X_n(t) \end{bmatrix}$$

be a differential rule, with initial condition $(x_1, .., x_n)$. Any polynomial which is a P-scale differential invariant for these initial conditions is actually an inductive invariant. □

Theorem 13. *(Completeness) There exist a differential rule $\mathcal{S}$ such that its invariants are not Polynomial-scale differential invariant. Such systems are then counter-example for completeness.* □

4.4 Differential Invariant Generation

Invariant generation for continuous time evolution is one of the main challenging step in static analysis and verification of hybrid systems. That is why we first restrict the analysis to differential system which appear in locations.

We start with strong-differential invariants generation.

4.4.1 Morphisms for *strong*-scale differential consecution

First, we consider a differential system of the form:

$$\mathcal{S} = \begin{bmatrix} \dot{X}_1 = P_1(X_1, \dots, X_n) \\ \vdots \\ \dot{X}_n = P_n(X_1, \dots, X_n) \end{bmatrix}. \tag{4.1}$$

We have the following lemma.

Lemma 1. *Let $Q \in \mathbb{R}[X_1, .., X_n]$ such that*

$$\mathscr{D}_Q(P_1, .., P_n, X_1, .., X_n) = 0.$$

Then Q is a strong-*scale differential invariant.* □

If $P \in \mathbb{R}[X_1, .., X_n]$ is of degree r and the maximal degree of the P_i's is d, then the degree of $\mathscr{D}_P(P_1, .., P_n, X_1, .., X_n)$ is $r+d-1$. Transposing the situation to linear algebra, consider the morphism

$$D : \left\{ \begin{array}{rcl} \mathbb{R}_r[X_1, \ldots, X_n] & \rightarrow & \mathbb{R}_{r+d-1}[X_1, \ldots, X_n] \\ P & \mapsto & \mathscr{D}_P(P_1, \ldots, P_n, X_1, \ldots, X_n). \end{array} \right.$$

Let M_D be its matrix in the canonical basis of $\mathbb{R}_r[X_1, ., X_n]$ and $\mathbb{R}_{r+d-1}[X_1, .., X_n]$.

Example 16. *(M_D for 2 variables, a degree 2 differential rule, and degree 2 invariants) Consider the following differential rules:*

$$\begin{bmatrix} \dot{x}(t) = x^2(t) + x(t)y(t) + 3y^2(t) + 3x(t) + 4y(t) + 4 \\ \dot{y}(t) = 3x^2(t) + x(t)y(t) + y^2(t) + 4x(t) + y(t) + 3 \end{bmatrix}. \tag{4.2}$$

In this example we write

$$P_1(x, y) = x^2 + xy + 3y^2 + 3x + 4y + 4$$

and

$$P_2(x, y) = 3x^2 + xy + y^2 + 4x + y + 3.$$

We consider the associated morphism D from $\mathbb{R}_2[x, y]$ to $\mathbb{R}_3[x, y]$. Using the basis

$$B_1 = (x^2, xy, y^2, x, y, 1)$$

of $\mathbb{R}_2[x, y]$ and

$$B_2 = (x^3, x^2y, xy^2, y^3, x^2, xy, y^2, x, y, 1)$$

of $\mathbb{R}_3[x, y]$, we define the matrix M_D.

To do so, we compute $D(P)$ for all elements P in the basis $(x^2, xy, y^2, x, y, 1)$ and we express the results in the basis $(x^3, x^2y, xy^2, y^3, x^2, xy, y^2, x, y, 1)$.

Hence, to get the first column of M_D we first consider

$$P(x, y) = x^2,$$

the first element of B_1, and we compute

$$D(P) = \mathscr{D}_P(P_1, P_2, x, y)$$

which is expressed in B_2 as

$$D(x^2) = \mathit{2}\,x^3 + \mathit{2}\,x^2y + \mathit{6}\,xy^2 + \mathit{0}\,y^3 + \mathit{6}\,x^2 + \mathit{8}\,xy + \mathit{0}\,y^2 + \mathit{8}\,x + \mathit{0}\,y + \mathit{0} \times 1$$

$$M_D = \begin{pmatrix} \mathit{2} & 3 & 0 & 0 & 0 & 0 \\ \mathit{2} & 2 & 6 & 0 & 0 & 0 \\ \mathit{6} & 2 & 2 & 0 & 0 & 0 \\ \mathit{0} & 3 & 2 & 0 & 0 & 0 \\ \mathit{6} & 4 & 0 & 1 & 3 & 0 \\ \mathit{8} & 7 & 8 & 1 & 1 & 0 \\ \mathit{0} & 4 & 2 & 3 & 1 & 0 \\ \mathit{8} & 3 & 0 & 3 & 4 & 0 \\ \mathit{0} & 4 & 6 & 4 & 1 & 0 \\ \mathit{0} & 0 & 0 & 4 & 3 & 0 \end{pmatrix}.$$

As we can see, a differential system $\mathcal{S}$ and a considered degree r, are the only required informations in order to build M_D. □

Now let $Q \in \mathbb{R}[X_1, .., X_n]$ be a strong-scale differential invariant for a given differential system defined by $P_1, .., P_n \in \mathbb{R}[X_1, .., X_n]$. Then

$$\begin{aligned} (\mathscr{D}_Q(P_1, .., P_n, X_1, .., X_n) = 0) \;&\Leftrightarrow\; (D(Q) = 0_{K[X_1,..,X_n]}) \\ &\Leftrightarrow\; (Q \in Ker(M_D)). \end{aligned}$$

We can see that Q will be a strong-scale differential invariant if and only if it is in the kernel of M_D.

Theorem 14. *A polynomial Q of $\mathbb{R}_r[X_1, .., X_n]$ is a* strong-*scale differential invariant for the differential system (4.1) if and only if it lies in the kernel of M_D.* □

Now we want to know when one can assert the existence of a non-trivial invariant polynomial of degree r. We denote by $v(r)$ the dimension of $\mathbb{R}_r[X_1, .., X_n]$.

If we add initial conditions of the form $(x_1(0) = u_1, \ldots, x_n(0) = u_n)$, we are looking for a *strong*-scale differential invariant in $\mathbb{R}_r[x_1, \ldots, x_n]$ that belongs to the hyperplane $P(u_1, \ldots, u_n) = 0$, *i.e.*, we are looking for Q in

$$ker(M_D) \cap \{P \mid P(u_1, \ldots, u_n) = 0\}.$$

We deduce the following theorem.

Theorem 15. *Let Q be in $\mathbb{R}_r[X_1, .., X_n]$. Then Q is an inductive invariant for the differential system with initial values $(u_1, .., u_n)$ if and only if Q is in the intersection of $Ker(M_D)$ and the hyperplane $Q(u_1, \dots, u_n) = 0$.* □

The intersection of the hyperplane $\{P | P(u_1, \dots, u_n) = 0\}$ with constant polynomials is always reduced to zero, and the intersection of any hyperplane with a subspace of $\mathbb{R}_r[x_1, \dots, x_n]$ has dimension of at least 1.

From the preceding theorem and the remark that follows it, there always exists non-trivial invariant when M_D has a kernel of dimension at least 2 (*i.e.* when M_D has rank at most $v(r) - 2$) as it will intersect any initial (semi-)hyperplane.

We deduce the following corollary.

Corollary 6. *There exists a strong-scale invariant of degree r for the differential system with initial conditions (any initial conditions, actually), if and only if the kernel of M_D is of dimension at least 2. The basis of $Ker(M_D)$ gives a* basis of a non-trivial invariant ideal □

So, if corollary 6 holds for a given differential systems, we will compute the basis of $Ker(M_D)$ to obtain a basis of non-trivial invariant. We will see in the following that such strategies is very effective and practical once the consecution condition is relaxed to constant-scaling and polynomial-scaling.

Now consider the following differential system with initial conditions

$$\begin{bmatrix} \dot{x}(t) = x(t) \\ \dot{y}(t) = ny(y) \\ (x(0), y(0)) = (\lambda, \mu) \end{bmatrix}, \tag{4.3}$$

where n is a parameter in V and $x(t)$, $y(t)$ are function of t in V_t. The solutions of this system are well known:

$$x(t) = \lambda e^t \quad \text{and}$$

$$y(t) = \mu e^{nt}.$$

Consider the polynomial $Q(x, y) = x^n/\lambda^n - y/\mu$. Hence, it is immediate that the polynomial assertion

$$x^n/\lambda^n - y/\mu = 0$$

is an invariant. It is actually a generator of the ideal of invariants. For if Q' is invariant, it is null on the points $(\lambda u, \mu u^n)$ for $u \in \mathbb{R}$ and so $x^n/\lambda^n - y/\mu$ divides Q'. For this system it is the most significant invariant one can get. Now, $Q(x, y) = x^n/\lambda^n - y/\mu$ is not a strong-scale differential invariant because $\partial_1 Q = nx^{n-1}/\lambda^n$ and $\partial_2 Q = -1/\mu$, and

$$\partial_1 Q(x, y)x + \partial_2 Q(x, y)y = nx^n/\lambda^n - y/\mu \neq 0.$$

In order to simplify the notation, take $n = 1$. We show that there cannot exist a non-trivial strong-scale differential invariant for the system

$$\begin{bmatrix} \dot{x} = x \\ \dot{y} = y \end{bmatrix}. \tag{4.4}$$

Suppose such an invariant exists. Write it as $Q(x, y) = \sum_{i,j} a_{i,j} x^i y^j$. The relation $\partial_1 Q(x, y)x + \partial_2 Q(x, y)y = 0$ implies $\sum_{i,j} i a_{i,j} x^i y^j + \sum_{i,j} j a_{i,j} x^i y^j = 0$, which gives $(i + j)a_{i,j} = 0$. As $i \geq 0$ and $j \geq 0$, this implies that all $a_{i,j} = 0$ but for $a_{0,0}$. Hence, Q is constant. Thus, even in cases where very simple invariants can be found, one will not find *strong*-scale differential invariants which are non-trivial inductive invariants. Therefore, we can conjecture that strong invariants exist in special cases. In the following we establish characterisation properties and classes of differential systems admitting *strong*-scale differential invariants which are non-trivial inductive invariants. We will use the following lemma.

Lemma 2. *Let $Q_1, \ldots, Q_n$ be n polynomials in $\mathbb{R}[X_1, \ldots, X_n]$. Then there exists a polynomial Q such that $\partial_1 Q = Q_1, \ldots, \partial_n Q = Q_n$ if and only if for any $i \neq j$, $1 \leq i, j \leq n$, one has $\partial_i Q_j = \partial_j Q_i$.* □

Let $Syz(P_1, .., P_n)$ denote the *Syzygy Module* [74] of $(P_1, \ldots, P_n)$.

Definition 23. *Let $P_1, \ldots, P_K$ be k polynomials in $\mathbb{R}[X_1, \ldots, X_n]$. The* Syzygy Module *of $(P_1, \ldots, P_k)$ is the following set:*

$$\{\ (Q_1, \ldots Q_k) \in \mathbb{R}[X_1, \ldots X_n] \mid Q_1 P_1 + Q_2 P_2 + \cdots + Q_k P_k = 0\ \}.$$

□

We can state the following theorem.

Theorem 16. *There exists a strong-scale invariant for a differential system if and only if there exists $(Q_1, .., Q_n)$ in $Syz(P_1, .., P_n)$ such that for any i, j with $i \neq j$ and $1 \leq i, j, \leq n$, one has $\partial_i Q_j = \partial_j Q_i$.* □

For example, when $n = 2$, we get the following class of systems for which one can always find a strong invariant:

$$\begin{bmatrix} \dot{x_1} = P_1(x_1, x_2) \\ \dot{x_2} = P_1(x_1, x_2) \end{bmatrix}. \tag{4.5}$$

with $\partial_2 P_2 = -\partial_1 P_1$. Indeed, $(P_2 - P_1)$ always belongs to $Syz(P_1, P_2)$. In fact, it is actually a basis when P_1 and P_2 are relatively prime.

Example 17. *Consider the following differential rules.*

$$\begin{bmatrix} \dot{x} = xy \\ \dot{y} = -y^2/2 \end{bmatrix}. \tag{4.6}$$

Here, we indeed have $\partial_2 P_2 = -\partial_1 P_1 = -y$. The corresponding invariant is $Q(x, y) = xy^2/2$.

Example 18. *Another example of systems admitting strong invariants is a generalization to dimension n of the rotational motion of a rigid body:*

$$\begin{bmatrix} \dot{x_1} = a_1 x_2 \dots x_n \\ \vdots \\ \dot{x_n} = a_n x_1 \dots x_{n-1} \end{bmatrix}. \tag{4.7}$$

We treat the case when the a_i's are non zero parameters, other cases being easier. Indeed, the vector

$$(Q_1 = x_1/a_1, Q_2 = -x_2/(n-1)a_2, \dots, Q_n = -x_n/(n-1)a_n)$$

belongs to $Syz(P_1, \dots, P_n)$, where

$$P_i = a_i x_1 \dots x_{i-1} x_{i+1} \dots x_n$$

belongs to the set of polynomials defining the differential rule.

Now if $i \neq j$, one has $\partial_i Q_j = \partial_j Q_i = 0$, and applying Theorem 16 we deduce that the system admits a strong invariant. In order to obtain an invariant, we just have to solve

$$\partial_1 Q = x_1/a_1; Q_2 = -x_2/(n-1)a_2; \dots; Q_n = -x_n/(n-1)a_n.$$

A trivial solution is

$$Q(x_1, \dots, x_n) = x_1^2/2a_1 - x_2^2/2(n-1)a_2 \dots - x_n^2/2(n-1)a_n.$$

Hence, the system admits as strong invariant the following assertion:

$$Q(x_1, \dots, x_n) = x_1^2/2a_1 - x_2^2/2(n-1)a_2 \dots - x_n^2/2(n-1)a_n = 0.$$

4.4.2 Morphisms for *constant*-scale differential consecution

Consider the differential system S depicted in Eq. 4.1. We state the following lemma.

Lemma 3. *Let $Q \in \mathbb{R}[X_1, ..., X_n]$ such that*

$$\mathscr{D}_Q(P_1, .., P_n, X_1, .., X_n) = \lambda Q(X_1, .., X_n).$$

Then Q is a λ-scale invariant. □

If Q has degree r, and the maximal degree of the P_i's is d, then we know that $\mathscr{D}_Q(P_1, ..., P_n, X_1, ..., X_n)$ has degree $r+d-1$. Hence we deduce that, in general, constant-scale consecution will work when the polynomials P_i of the differential transition system are of degree one, *i.e.* when the transition system is affine. So, suppose that the P_i's are of degree one. Now we want to find an invariant Q of degree r.

We reduce the problem again to linear algebra. Consider the endomorphism D of $\mathbb{R}_r[X_1, \dots, X_n]$ given by

$$D : \left\{ \begin{array}{rcl} \mathbb{R}_r[X_1, \dots, X_n] & \to & \mathbb{R}_r[X_1, \dots, X_n] \\ P & \mapsto & \mathscr{D}_P(P_1, \dots, P_n, X_1, \dots, X_n). \end{array} \right.$$

Using lemma 3, Q will be a λ-invariant for constant-scale consecution of degree at most r if and only if λ is an eigenvalue of D, and Q is an eigenvector for λ. By letting M_D be the matrix of D in the canonical basis of $\mathbb{R}_r[X_1, .., X_n]$ we can state the following theorem.

Theorem 17. *A polynomial Q of $\mathbb{R}_r[X_1, .., X_n]$ is a λ-scale invariant for continuous scale consecution of the differential system if and only if there exists an eigenvalue λ of M_D such that Q belongs to the eigenspace of M_D corresponding to λ.* □

Zero is always an eigenvalue of M_D, since its last column is always null. But this gives a constant eigenvector, which is less interesting.

In the following cases we describe the methods in the most general case for 2 variables and the generation of λ-invariant of degree 2

Example 19. *(General case for 2 variables and degree 2) Consider the differential system of the following form:*

$$\begin{bmatrix} \dot{x} = a_1x + b_1y + c_1 \\ \dot{y} = a_2x + b_2y + c_2 \end{bmatrix} \tag{4.8}$$

The matrix M_D in the basis $(x^2, xy, y^2, x, y, 1)$ is

$$M_D = \begin{pmatrix} 2a_1 & a_2 & 2b_2 & 0 & 0 & 0 \\ 2b_1 & a_1 + b_2 & 2a_2 & 0 & 0 & 0 \\ 0 & b_1 & 0 & 0 & 0 & 0 \\ 2c_1 & c_2 & 0 & a_1 & 0 & 0 \\ 0 & c_1 & 2c_2 & b_1 & b_2 & 0 \\ 0 & 0 & 0 & c_1 & c_2 & 0 \end{pmatrix}.$$

This matrix is block lower triangular, with blocks of size 3×3*. Hence, its characteristic polynomial is the product of two degree* 3 *polynomials, and roots of such polynomials can be computed by Cardan's method.*

Thus, one will always be able to find non-null λ*-scale invariants in this case.* □

We just proved the following proposition and gave a method for finding the corresponding invariants.

Proposition 1. *If we are looking at an affine differential transition system with polynomials in two variables, then one is always able to find good scale invariants.* □

As we did in [80] when dealing with discrete consecution, we can identify large decidable classes, *e.g.*

- (i) when M_D is block triangular with 4×4 blocks or less; and
- (ii) when the eigenspace associated with eigenvalue 1 is of dimension greater than 1; among others.

Theorem 18. *A polynomial* Q *in* $\mathbb{R}_r[X_1, .., X_n]$ *is a* λ*-scale invariant for the differential system with initial values* $(u_1, \ldots, u_n)$ *if and only if there exists an eigenvalue* λ *of* M_D *such that* Q *belongs to the intersection of the eigenspaces corresponding to* λ *and the hyperplane* $Q(u_1, \ldots, u_n) = 0$. □

Corollary 7. *There will be a non-null polynomial invariant for any given initial values if and only if there exists an eigenspace of* M_D *with dimension at least* 2. □

Example 20. *Consider system (4.3) again, which we could not handle using strong-scale invariant encoding. We recall that the differential system*

$$\begin{bmatrix} \dot{x} = x \\ \dot{y} = ny \\ (x(0), y(0)) = (\lambda, \mu) \end{bmatrix} \tag{4.9}$$

has an associated endomorphism

$$D : Q(x, y) \mapsto \partial_x Q(x, y)x + n\partial_y Q(x, y)y.$$

Writing its matrix in the basis

$$(x^n, x^{n-1}y, \ldots, xy^{n-1}, y^n, \ldots\ldots, x, y, 1)$$

we have:

$$\begin{pmatrix} n & \ldots & 0 & 0 \\ 0 & M_D & 0 & 0 \\ 0 & \ldots & n & 0 \\ 0 & \ldots & 0 & 0 \end{pmatrix}.$$

We see that the eigenspace corresponding to n has at least dimension 2, and it contains $Vect(x^n, y)$ (the vector spaces generated by x^n and y). Using the theorem on the existence on solutions for any initial conditions, we deduce that for the initial values $(x(0) = \lambda, y(0) = \mu)$ there exists an invariant of the form $ax^n + by$, and which must verify $a\lambda^n + b\mu = 0$. If λ and μ are non zero, which is the interesting case, one can take $a = \lambda^{-n}$ and $b = -\mu^{-1}$, which gives the inductive invariant

$$Q(x, y) = x^n/\lambda^n - y/\mu = 0.$$

□

4.5 Handling non-linear differential systems

We consider a *non-linear* differential system of the form:

$$\mathcal{S} = \begin{bmatrix} \dot{X}_1(t) = P_1(X_1(t), .., X_n(t)) \\ \vdots \\ \dot{X}_n(t) = P_n(X_1(t), .., X_n(t)) \end{bmatrix},$$

with the P_i's in $\mathbb{R}[X_1, .., X_n]$.

We know that as soon as one of the P_i's has degree more than one, we must use polynomial-scale consecution in order to obtain interesting invariants [81].

We have the following lemma.

Lemma 4. *Let $Q \in \mathbb{R}[X_1, .., X_n]$ such that*

$$\mathscr{D}_Q(P_1, .., P_n, X_1, .., X_n) = TQ$$

with T in $\mathbb{R}[X_1, .., X_n]$. Then Q is a T-scale invariant. □

If $P \in \mathbb{R}[X_1, .., X_n]$ is of degree r and the maximal degree of the P_i's is d, then the degree of $\mathscr{D}_P(P_1, .., P_n, X_1, .., X_n)$ is $r + d - 1$.

Hence, T must be searched in the subspace of $\mathbb{R}[X_1, .., X_n]$, which is of degree at most $r + d - 1 - r = d - 1$.

Transposing the situation to linear algebra, consider the morphism

$$D : \left\{ \begin{array}{rcl} \mathbb{R}_r[X_1, \dots, X_n] & \to & \mathbb{R}_{r+d-1}[X_1, \dots, X_n] \\ P & \mapsto & \mathscr{D}_P(P_1, \dots, P_n, X_1, \dots, X_n). \end{array} \right.$$

Let M_D be its matrix in the canonical basis of $\mathbb{R}_r[X_1, ., X_n]$ and $\mathbb{R}_{r+d-1}[X_1, .., X_n]$. Here, we build matrices M_D in a same manner as we did and describe in Section 4.4.1 Example 16.

Choosing a generic T in $\mathbb{R}_{d-1}[X_1, .., X_n]$, we define the associated morphism

$$\overline{T} : \begin{cases} \mathbb{R}_r[x_1, \ldots, x_n] & \rightarrow & \mathbb{R}_{r+d-1}[x_1, \ldots, x_n] \\ P & \mapsto & TP. \end{cases}$$

Denote by L_T its matrix in the canonical basis, obtained as in the computation of M_D. Matrices L_T corresponding to multiplication by polynomials T of $\mathbb{R}_{d-1}[x_1, \ldots, x_n]$ have a very precise form, dependent on the coefficients of T. Thus, for fixed n, r and d, they can be easily identified. We will call $M(pol)$ the set of such matrices. It is, in fact, a (vector-)subspace of matrices corresponding to morphisms from $\mathbb{R}_r[x_1, \ldots, x_n]$ to $\mathbb{R}_{r+d-1}[x_1, \ldots, x_n]$.

To be even more precise, if T is a *generic template* in $\mathbb{R}_{d-1}[X_1, .., X_n]$, call $t_1, .., t_{v(d-1)}$ its coefficients where $v(d-1)$ is the dimension of $\mathbb{R}_{d-1}[X_1, .., X_n]$.

Then L_T's coefficients are in $\{t_1, .., t_{v(d-1)}\}$ and it has a natural block decomposition. We will call $M(pol)$ the set of such matrices.

In order to fix ideas, we show what happens for two variables, P_i's of maximal degree 3, and we are looking for an invariant in $\mathbb{R}_2[x, y]$. Hence, T lies in $\mathbb{R}_2[x, y]$.

Example 21. *A generic T is of the form*

$$T(x, y) = t_1x^2 + t_2xy + t_3y^2 + t_4x + t_5y + t_6.$$

Using the basis $B_2 = (x^2, xy, y^2, x, y, 1)$ of $\mathbb{R}_2[x, y]$ and the basis

$$B_4 = (x^4, x^3y, x^2y^2, xy^3, y^4, x^3, x^2y, xy^2, y^3, x^2, xy, y^2, x, y, 1)$$

of $\mathbb{R}_4[x, y]$, we define the matrices L_T.

To do so, we compute $\overline{T}(P)$ for all elements P in the basis B_2 and we express the results in the basis B_4.

In other words, to get the first column of L_T we first consider

$$P(x, y) = x^2$$

the first element of B_2, and we compute

$$\overline{T}(P) = TP$$

which is expressed in B_4 as

$$\overline{T}(x^2) = t_1\, x^4 + t_2\, x^3y + t_3\, x^2y^2 + 0\, xy^3 + 0\, y^4 + t_4\, x^3 + t_5\, x^2y + 0\, xy^2 + 0\, y^3 + t_6\, x^2 + 0\, xy + 0\, y^2 + 0\, x + 0\, y + 0 \times 1$$

$$\begin{pmatrix} t_1 & 0 & 0 & 0 & 0 \\ t_2 & 0 & 0 & 0 & 0 \\ t_3 & t_1 & 0 & 0 & 0 \\ 0 & t_2 & 0 & 0 & 0 \\ 0 & t_3 & 0 & 0 & 0 \\ t_4 & 0 & t_1 & 0 & 0 \\ t_5 & 0 & t_2 & t_1 & 0 \\ 0 & t_4 & t_3 & t_2 & 0 \\ 0 & t_5 & 0 & t_3 & 0 \\ t_6 & 0 & t_4 & 0 & t_1 \\ 0 & 0 & t_5 & t_4 & t_2 \\ 0 & t_6 & 0 & t_5 & t_3 \\ 0 & 0 & t_6 & 0 & t_4 \\ 0 & 0 & 0 & t_6 & t_5 \\ 0 & 0 & 0 & 0 & t_6 \end{pmatrix}.$$

This determines $M(pol)$. □

Now let $Q \in \mathbb{R}[X_1,..,X_n]$ be a T-scale invariant for a given differential system defined by $P_1,..,P_n \in \mathbb{R}[X_1,..,X_n]$. Then

$$\begin{aligned} (\mathscr{D}_Q(P_1,..,P_n,X_1,..,X_n) = TQ) &\Leftrightarrow D(Q) = \overline{T}(Q) \\ &\Leftrightarrow ((D-\overline{T})(Q) = 0_{\mathbb{R}[X_1,..,X_n]}) \\ &\Leftrightarrow (Q \in Ker(M_D - L_T)). \end{aligned}$$

So, a T-scale invariant is nothing else than a vector in the kernel of $M_D - L_T$.

Theorem 19. *There is a polynomial-scale invariant for the differential system if and only if there exists a matrix L_T in $M(pol)$, corresponding to a polynomial T of $\mathbb{R}_{d-1}[x_1,..,X_n]$, such that $Ker(M_D - L_T)$ is not reduced to zero. And, any vector in the kernel of $M_D - L_T$ will give a T-scale differential invariant.* □

Now notice that $M_D - L_T$ with a non trivial kernel is equivalent to it having rank strictly less than the dimension $v(r)$ of $\mathbb{R}_r[x_1,\dots,x_n]$. By a classical theorem [76], this is equivalent to the fact that each $v(r) \times v(r)$ sub-determinant of $M_D - L_T$ is equal to zero. Those determinants are polynomials with variables $(t_1, .., t_{v(d-1)})$, which we will denote by

$$E_1(t_1, ..., t_{v(d-1)}), ..., E_s(t_1, ..., t_{v(d-1)}).$$

Theorem 20. *There is a non trivial T-scale invariant if and only if the polynomials $(E_1, .., E_s)$ admit a common root, other than the trivial one* $(0, ..., 0)$. □

This theorem provides us with important existence results. But there is a more practical way to get invariant ideals without computing common roots. Consider initial values given by unknown parameters

$$(x_1(0) = u_1, \dots, x_n(0) = u_n).$$

The initial step defines a linear form on $\mathbb{R}_r[x_1, \dots, x_n]$, namely

$$I_u : P \mapsto P(u_1, ..., u_n).$$

Hence, initial values correspond to a hyperplane of $\mathbb{R}_r[X_1, .., X_n]$ given by the kernel I_u, which is

$$\{Q \in \mathbb{R}_r[X_1, .., X_n] \mid Q(u_1, \dots, u_n) = 0\}.$$

Theorem 21. *Let Q be in $\mathbb{R}_r[X_1, .., X_n]$. Then Q is an inductive invariant for the differential system with initial values $(u_1, .., u_n)$ if and only if there exists a matrix $L_T \neq 0$ in $M(pol)$, corresponding to T in $\mathbb{R}_{d-1}[X_1, .., X_n]$, such that Q is in the intersection of $Ker(M_D - L_T)$ and the hyperplane $Q(u_1, \dots, u_n) = 0$.* □

Now, if

$$Dim(Ker(M_D - L_T)) \geq 2$$

then $Ker(M_D - L_T)$ would intersect any initial (semi-)hyperplane.

Corollary 8. *There are non-trivial invariants for any given initial values if and only if there exists a matrix L_T in $M(pol)$ such that $Ker(M_D - L_T)$ has dimension at least* 2. □

Also, we have $(Dim(Ker(M_D - L_T)) \geq 2)$ if and only if we also have $Rank(M_D - L_T) \leq Dim(\mathbb{R}_r[X_1, .., X_n]) - 2$. Further, we also show how to assign values to the coefficients of T in order to guarantee the existence and generation of invariants.

Example 22. ***(Running example)*** *Consider the following differential rules with* $P_1 = x^2 + 2xy + x$ *and* $P_2 = xy + 2y^2 + y$*:*

$$\begin{bmatrix} \dot{x}(t) = x^2(t) + 2x(t)y(t) + x(t) \\ \dot{y}(t) = x(t)y(t) + 2y^2(t) + y(t) \end{bmatrix}. \tag{4.10}$$

- ***Step 1****: We build matrix* $M_D - L_T$*.*
  The maximal degree of the systems is $d = 2$ *and the* T*-scale invariant will be of degree* $r = 2$*. Then,* T *is of degree* $d - 1 = 1$ *and we write* t_1, t_2, t_3 *for its unknown coefficients,* (i.e. *the canonical form is*

  $$T(x, y) = t_1 x + t_2 y + t_3.$$

  Using the basis

  $$(x^2, xy, y^2, x, y, 1)$$

  of $\mathbb{R}_2[x, y]$ *and the basis*

  $$(x^3, x^2y, xy^2, y^3, x^2, xy, y^2, x, y, 1)$$

  of $\mathbb{R}_3[x, y]$*, the matrix* $M_D - L_T$ *is:*

  $$M_D - L_T = \begin{pmatrix} 2-t_0 & 0 & 0 & 0 & 0 & 0 \\ 4-t_1 & 2-t_0 & 0 & 0 & 0 & 0 \\ -t_2 & 4-t_1 & 2-t_0 & 0 & 0 & 0 \\ 0 & 0 & 4-t_1 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1-t_0 & 0 & 0 \\ 0 & 2-t_2 & 0 & 2-t_1 & 1-t_0 & 0 \\ 0 & 0 & 2-t_2 & 0 & 2-t_1 & 0 \\ 0 & 0 & 0 & 1-t_2 & 0 & -t_0 \\ 0 & 0 & 0 & 0 & 1-t_2 & -t_1 \\ 0 & 0 & 0 & 0 & 0 & -t_2 \end{pmatrix}.$$

- ***Step 2****: Now the unknown* t_i*'s are given values so as to guarantee the existence of invariants.*
  Our algorithm proposes to fix $t_1 = 2$*,* $t_2 = 4$ *and* $t_3 = 2$ *to get*

  $$T(x, y) = 2x + 4y + 2.$$

  Matrix $M_D - L_T$ *has its second and third columns equal to zero. So, the rank of* $M_D - L_T$ *is less than 4 and its kernel has dimension at least 2. Any vector in this kernel will be a* T*-scale differential invariant.*

- ***Step 3*** *Now, Corollary 8 applies to* $M_D - L_T$.
  So, there will always be invariants, whatever the initial values. We compute and output the basis of $Ker(M_D - L_T)$*:*

  ```
  Polynomial scaling continuous evolution
  T(x,y) = 2 x + 4 y + 2
  Module of degree 6 and rank 2 and Kernel of dimension 4
  {{0, 1, 0, 0, 0, 0}, {0, 0, 1, 0, 0, 0}}
  ```

  The vectors of the basis are interpreted in the canonical basis of $\mathbb{R}_2[x,y]$*:*

  ```
  Basis of invariant Ideal
  {x y, y^2}
  ```

  We have an ideal for non trivial inductive invariants and we search for one of the form
  $$axy + by^2.$$
  If the system has initial conditions $x(0) = \lambda$ *and* $y(0) = \mu$*, then*
  $$a\lambda\mu + b\mu^2 = 0,$$
  and
  $$\mu xy - \lambda y^2 = 0$$
  is an invariant for all μ *and* λ.

□

4.6 Obtaining optimal degree bounds

In order to guarantee the existence of non-trivial invariants of degree r, we need a polynomial T such that
$$Ker(M_D - L_T) \neq 0.$$

First, define T as a polynomial with parametrized coefficients. We can then build a decision procedure to assign values to the coefficients of T in such a way that $Ker(M_D - L_T) \neq 0$.

The pseudo code depicted in Algorithm 2 illustrates this strategy. Algorithm 2 is in a standard form, but its contribution relies on very general sufficient conditions for the existence and the computation of invariants.

Algorithm 2: **Ideal_Inv_Gen**($r, P_1, ..., P_n, X_1, ..., X_n$)

```
/*Guessing the degree bounds.*/
Data: $r$ is the degree for the set of invariants we are looking for, $P_1, ..P_n$ are the $n$
      polynomials given by the considered differential rules, and $X_1, ..X_n \in V_t$ are
      functions of time.
Result: $B_{Inv}$, a basis of ideal of invariants.
begin
1     int $d$
2     Template $T$
3     Matrix $M_D$, $L_T$
4     $d \longleftarrow$ Max_degree($\{P_1, ..., P_n\}$)
5     /*$d$ is the maximal degree of $P_i$'s*/
6     if $d >= 2$ then
7         $T \longleftarrow$ Template_Canonical_Form($d - 1$)
8         $M_D \longleftarrow$ Matrix_D($r, r + d - 1, P_1, ..., P_n$)
9         $L_T \longleftarrow$ Matrix_L($r, r + d - 1, T$)
10        $\bar{M} \longleftarrow$ Reduce_Rank_Assigning_Values($M_D - L_T$)
11        if Rank($\bar{M}$) >= Dim($R_r[X_1, .., X_n]$) then
12            return Ideal_Inv_Gen($r + 1, P_1, ..., P_n, X_1, ..., X_n$)
13            /*We need to increase the degree r of candidates invariants.*/
14        else
15            return Nullspace_Basis($\bar{M}$)
16            /*There exists an ideal of invariants that we can compute*/
17    else
18        ... /*We refer to our previous work for strong and constant scaling.*/
end
```

From the differential rules, we obtain matrix M_D (see line 8) with real entries. We can then define degree bounds for matrices L_T that can be used to approximate the consecution requirements (see line 9).

As we recall from Section 4.5, $Ker(M_D - L_T) \neq 0$ is equivalent to having $M_D - L_T$ with rank strictly less than the dimension $v(r)$ of $\mathbb{R}_r[x_1, \ldots, x_n]$. We then reduce the rank of $M_D - L_T$ by assigning values of terms in M_D to parameters in L_T (see line 10).

Next, we determine whether the obtained matrix $\bar{M}$ has a trivial kernel by first computing its rank and then checking if

$$(\mathbf{Rank}(\bar{M}) < \mathbf{Dim}(R_r[X_1, .., X_n]))$$

holds (see line 11).

By so doing, we can increase the degree r of invariants until Theorem 19 (or Corollary 8) applies or until stronger hypotheses occur, *e.g.* if all $v(r) \times v(r)$ sub-determinants are null.

Then, we compute and output the basis of the nullspace of matrix $\bar{M}$ in order to construct an ideal basis for non trivial invariants (see `Nullspace_Basis`, line 15).

We can directly see that if there is no ideal for non-trivial invariants for a value r_i then we conclude that there is no ideal of non-trivial invariants for all degrees $k \leq r_i$. This could guide other constraint-based techniques, since checking for invariance with a template of degree less or equal to r_i will not be necessary. In case there is no ideal for invariants of degree r (see line 12), we first increment the value of r by 1 before the recursive call to `Ideal_Inv_Gen`.

We thus showed how to reduce the invariant generation problem to the problem of computing a kernel basis for polynomial mappings. For the latter, we use well-known state-of-the-art algorithms, *e.g.* that Mathematica provides. These algorithms calculate the eigenvalues and associated eigenspaces of $\bar{M}$ when it is a square matrix. When $\bar{M}$ is a rectangular matrix, we can use its *singular value decomposition* (SVD). A SVD of $\bar{M}$ provides an explicit representation of its rank and kernel by computing unitary matrices U and V and a regular diagonal matrix S such that

$$\bar{M} = USV.$$

We compute the SVD of a $v(r+d-1) \times v(r)$ matrix $\bar{M}$ by a two step procedure. First, reduce it to a bi-diagonal matrix, with a cost of $O(v(r)^2 v(r+d-1))$ flops. The second step relies on an iterative method, as is also the case for other eigenvalue algorithms. In practice, however, it suffices to compute the SVD up to a certain precision, *i.e.* up to a machine epsilon. In this case, the second step takes $O(v(r))$ iterations, each using $O(v(r))$ flops. So, the overall cost is $O(v(r)^2 v(r+d-1))$ flops.

For the implementation of the algorithm we could rewrite Corollary 8 as follow.

Corollary 9. *Let $\bar{M} = U\cdot S\cdot V$ be the singular value decomposition of matrix $\bar{M}$ described just above. There will be a non trivial T-invariant for any given initial condition if and only if the number of non-zero elements in matrix S is less than $v(r)-2$, where $v(r)$ is the dimension of $\mathbb{R}_r[x_1,\ldots,x_n]$. Moreover, the orthonormal basis for the nullspace obtained from the decomposition directly gives an ideal for non-linear invariants.* □

It is important to emphasize that eigenvectors of $\bar{M}$ are computed after the parameters of L_T have been assigned. When the differential system has several variables and none or few parameters, $\bar{M}$ will be over the reals and there will be no need to use the symbolic version of these algorithms.

4.7 Examples and Experimental Results

By reducing the problem to Linear Algebra, we are able to combine it with new optimization techniques, as illustrated in the following examples. Depending on the form of the monomials present in the system, we may be able to find T and a vector X such that $X \in Ker(M_D - L_T)$ without defining T as a template, *i.e.* without using a polynomial with unknown coefficients for scaling consecution. The idea is to directly obtain a suitable T by *factorization*. For instance, we can identify the following large classes of systems where the methods apply.

Example 23. *Let $s \in \mathbb{N}$ be a positive and consider the following differential rules:*

$$\begin{bmatrix} \dot{x}_1(t) = \sum_{k=0}^{s} a_k x_1(t)^{k+1} x_2(t)^k \cdots x_n(t)^k \\ \vdots \\ \dot{x}_n(t) = \sum_{k=0}^{s} a_k x_1(t)^k \cdots x_{n-1}(t)^k x_n(t)^{k+1} \end{bmatrix}. \tag{4.11}$$

This differential system contains parameters and variables that are time functions. We denote the polynomials thus

$$\begin{aligned} P_1 &= \sum_{k=0}^{s} a_k x_1^{k+1} x_2^k \ldots x_n^k; \\ &\vdots \\ P_n &= \sum_{k=0}^{s} a_k x_1^k \ldots x_{n-1}^k x_n^{k+1}. \end{aligned}$$

Let D be the morphism associated with (4.11) and let M_D be its matrix in the canonical basis. Then, it is immediate that

$$\mathscr{D}_P(x_i) = P_i.$$

Now, for this particular class of P_i's, we see that

$$\mathscr{D}_P(x_i) = x_i T,$$

where

$$T = \sum_{k=0}^{s} a_k x_1^k x_2^k \dots x_{n-1}^k x_n^k.$$

This means that if $\overline{T}$ is the morphism associated to multiplication by T, we have

$$\mathscr{D}_P(x_i) = \overline{T}(x_i)$$

for each i. Let L_T be its matrix in the canonical basis. We deduce that

$$Vect(x_1, .., x_n) \subset Ker(M_D - L_T).$$

Hence, for $n \geq 2$, the space $Ker(M_D - L_T)$ has dimension greater than 2, and we can apply our existence theorem for invariants, given any initial values.

We can then search for an invariant of the form

$$a_1 x_1 + \dots + a_n x_n.$$

Given the initial conditions

$$(x_1(0) = \lambda_1, \dots, x_n(0) = \lambda_n),$$

a vector $(a_1 \cdots a_n)^\top$ is such that the polynomial

$$a_1 x_1 + \dots + a_n x_n$$

is an invariant for (4.11) whenever it belongs to the kernel of the linear form with matrix $(\lambda_1, \dots, \lambda_n)$. Summarizing, with polynomial scaling, any polynomial $Q = a_1 x_1 + \dots + a_n x_n$ with $(a_1 \cdots a_n)^\top$ in the kernel of $(\lambda_1, \dots, \lambda_n)$ is an invariant for (4.11). □

Example 24. *In order to handle air traffic management systems [104, 127] automatically, we consider the given differential system:*

$$\begin{bmatrix} \dot{x_1} = a_1 cos(\omega t + c) \\ \dot{x_2} = a_2 sin(\omega t + c) \end{bmatrix}. \tag{4.12}$$

This models the system satisfied by one of the two airplanes. We introduce the new variables d_1 and d_2 to handle the transcendental functions, axiomatizing them by differential equations, so that d_1 and d_2 satisfy

$$\begin{bmatrix} \dot{d_1} = -a_1/a_2 \omega d_2 \\ \dot{d_2} = a_2/a_1 \omega d_1. \end{bmatrix} \tag{4.13}$$

If D is the morphism associated to this system, it is immediate that

$$D(a_2^2d_1^2) = -2a_1a_2\omega d_1d_2$$

whereas

$$D(a_1^2d_2^2) = 2a_1a_2\omega d_1d_2.$$

From [83, 88], it implies that

$$Vect(a_2^2d_1^2 + a_1^2d_2^2) \subset Ker(D)$$

and so

$$a_2^2d_1^2 + a_1^2d_2^2$$

is a strong-*scale invariant (*i.e. *a T-scale invariant where T is null) for the system. But*

$$\dot{x_1} = d_1 = [a_1/(a_2\omega)]\dot{d_2}$$

and

$$\dot{x_2} = d_2 = [-a_2/(a_1\omega)]\dot{d_1}.$$

Therefore, there exist constants c_1 and c_2, determined by the initial values, such that

$$x_1 = a_1/a_2\omega d_2 + c_1$$

and

$$x_2 = d_2 = -a_2/a_1\omega d_1 + c_2.$$

This implies that

$$(a_2x_1 - k_1)^2 + (a_1x_2 - k_2)^2 = 0,$$

with $k_1 = a_2c_1$ and $k_2 = a_1c_2$, is an invariant of the first system. Hence the two airplanes, at least for some lapse of time, follow an elliptical path. □

In the following two examples we have shown again how to deal with differential systems with parameters and several variables.

Example 25. *Consider the following differential rules:*

$$\begin{bmatrix} \dot{x} = ax^2(t) + bx(t)y(t) + cx(t) \\ \dot{y} = ax(t)y(t) + by^2(t) + cy(t) \end{bmatrix}. \tag{4.14}$$

In this example we have two polynomials of degree 2, with two variables $x(t)$ and $y(t)$ in V_t and three parameters a, b, c, in V: $P_1 = ax^2 + bxy + cx$ and $P_2 = axy + by^2 + cy$. Using the basis $(x^2, xy, y^2, x, y, 1)$ of $\mathbb{R}_2[x, y]$ and the basis $(x^3, x^2y, xy^2, y^3, x^2, xy, y^2, x, y, 1)$ of $\mathbb{R}_3[x, y]$, the matrix M_D is depicted on the left just below. Here the polynomial T we use for scaling must be of degree 1. Hence $T(x, y) = t_1x + t_2y + t_3$ where t_1, t_2, t_3 are unknown parameters that will be assigned to values in order to guarantee the existence and generation of invariants. The associated matrix L_T of the T-multiplication morphism has the form depicted at the right just below:

$$M_D = \begin{pmatrix} 2a & 0 & 0 & 0 & 0 & 0 \\ 2b & 2a & 0 & 0 & 0 & 0 \\ 0 & 2b & 2a & 0 & 0 & 0 \\ 0 & 0 & 2b & 0 & 0 & 0 \\ 2c & 0 & 0 & a & 0 & 0 \\ 0 & 2c & 0 & b & a & 0 \\ 0 & 0 & 2c & 0 & b & 0 \\ 0 & 0 & 0 & c & 0 & 0 \\ 0 & 0 & 0 & 0 & c & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \qquad L_T = \begin{pmatrix} t_1 & 0 & 0 & 0 & 0 & 0 \\ t_2 & t_1 & 0 & 0 & 0 & 0 \\ t_3 & t_2 & t_1 & 0 & 0 & 0 \\ 0 & 0 & t_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & t_1 & 0 & 0 \\ 0 & t_3 & 0 & t_2 & t_1 & 0 \\ 0 & 0 & t_3 & 0 & t_2 & 0 \\ 0 & 0 & 0 & t_3 & 0 & t_1 \\ 0 & 0 & 0 & 0 & t_3 & t_2 \\ 0 & 0 & 0 & 0 & 0 & t_3 \end{pmatrix}.$$

Then taking

$$T(x, y) = 2ax + 2by + 2c,$$

i.e. $t_1 = 2a$, $t_2 = 2b$ *and* $t_3 = 2c$ *one verifies that the matrix* $M_D - L_T$ *has its second and third columns equal to zero. Hence, the rank of* $M_D - L_T$ *is less than* 4, *and our existence theorem for any given initial values applies.* □

Example 26. *Consider the following differential rules:*

$$\begin{bmatrix} \dot{x}(t) = x^2(t) + x(t)y(t) - x(t)z(t) \\ \dot{y}(t) = 2x(t)y(t) + y^2(t) \\ \dot{z}(t) = z(t)y(t) - 2z^2(t) \end{bmatrix}. \tag{4.15}$$

Our method shows that

$$x^2(t) - y(t)z(t) - x_0 = 0$$

is an inductive invariant with polynomial scaling $T = 2(x + y - z)$ *with* $x_0 = x(0)$ *an initial parameter.* □

By analogy with the discrete case (see Chapter 3), the way invariants are computed in the context of fractional-scale continuous consecution is similar to the case of polynomial-scale consecution.

Table 4.1 summarizes the type of linear algebraic problems associated with each consecution approximation. The last column gives some existential results that could be reused by any constraint-based approach or reachability analysis.

In Table 4.2 we list some experimental results.

Tabela 4.1: Linear algebraic problems and consecution approximations

Aprox.Consec.	Lin. Algeb. Prob.	Existence Conditions
Strong	nullspaces	$Ker(M_D) \neq \emptyset$ or (see [83]) $\exists (Q_1, .., Q_n) \in Syz(P_1, .., P_n)$, s.t $\partial_i Q_j = \partial_j Q_i$
Lambda	eigenspaces	$Ker(M_D) \geq 2$ for any init. cond., and $Ker(M_D) \neq \emptyset$ otherwise.
Polynomial	nullspaces	$Ker(M_D - L_T) \geq 2$ for any init. cond., and $Ker(M_D - L_T) \neq \emptyset$ otherwise.

4.8 Putting all together: Global invariants

In the previous sections and in Chapter 3 we have shown how to handle continuous and discrete consecution conditions and how to generate ideals of invariants for each states and for the transitions structure. To be more precise, we thus generated a basis of a vectorial space which describes invariants for each location, transitions and initial conditions. A global invariant would be any invariant which is in the intersection of these three vector spaces. In this way, we avoid the definition of a single isomorphism for the whole hybrid system. Instead, we generate the basis for each separate consecution condition. To compute the basis of global invariants, we could use theorem 22. It proposes to *multiply* all the elements of each computed basis. By so doing, we also avoid the heavy computation of ideal intersections. This approach is a sound, but not complete, way of computing ideals for global hybrid invariants, and it has a lower computational complexity.

Theorem 22. *Let W be a hybrid system and let l be one of its locations. Let $I = \{I_1, ..., I_k\}$ a set of ideals in $\mathbb{R}[X_1, ..., X_n]$ such that $I_j = ({f^{(j)}}_1, ..., f^{(j)}_{n_j})$ where $j \in [1, k]$. Let $\otimes(I_1, ..., I_k) = \{\delta_1, ..., \delta_{n_1 n_2 \ldots n_k}\}$ be such that all elements δ_i in $\otimes(I_1, ..., I_k)$ are formed by the product of one element from each ideal in I. Assume that the I_j's are collections of invariant ideals associated to $\mathcal{S}(l)$, its differential rule, $\mathcal{C}(l)$, its local conditions, and all invariant ideals generated considering all incoming transitions at l. Then $\otimes(I_1, ..., I_k)$ is a non-trivial invariant ideal for location l.* $\square$

Tabela 4.2: Experimental results: Basis of invariant ideals obtained automatically by our prototype. All examples are treated in Section 4.4.1, Section 4.4.2, Section 4.5 and Section 4.7

Differential Syst.	Scal.	CPU/s
See Section 4.7, system 4.14 (also from [81]).	*Poly.*	`1.12`
See Section 4.7, system 4.13 (also from [81]).	*Poly.*	`2.04`
See Section 4.7, system 4.15 (also from [88]).	*Poly.*	`0.34`
See Section 4.7, systems 4.11 (also from [84]).	*Poly.*	`98.49`
See Section 4.5, system 4.10 (also from [84]).	*Poly.*	`0.43`
See Section 4.4.2, system 4.9 (also from [83]).	*Lamb.*	`2.48`
See Section 4.4.1, system 4.6 (also from [81]).	*Str.*	`0.02`
See Section 4.7, systems 4.12 (also from [84, 104, 127]).	*Str.*	`1.29`
See Section 4.4.1, system 4.4 (also from [81]).	*Lamb.*	`0.03`
See Section 4.4.1, system 4.7 (also from [83]).	*Str.*	`15.90`
See Section 4.4.2, system 4.8 (also from [81]).	*Lamb.*	`1.04`

Corollary 10. *Let l be a state and let $\mathcal{C}(l) \equiv (P_i(x_1,..,x_n) < 0)$ be its semi-algebraic local conditions and Q be an inductive invariant for $\mathcal{D}(l)$, its differential rule, and all ideals of invariants generated considering all incoming transitions at l. Then $(P_i(x_1,..,x_n) - Q(x_1,..,x_n) < 0)$ is an inductive invariant.* □

Semi-algebraic local state conditions, as well as initiation and transition guards are assertions of the form $(P_i(x_1,..,x_n) < 0)$ with $P_i \in K[x_1,..,x_n]$. Then, we obtain an operator, similar to the one introduced in Theorem 22, to generate ideals of non-trivial invariants at a state l with semi-algebraic local conditions. We can then generate ideals of non-trivial semi-algebraic invariants.

4.9 Chapter Discussions: Performances and Limitations

Despite tremendous progress over the years [118, 125, 111, 114, 1, 124, 8, 52, 3, 16, 115, 104], automatic generation of invariant for hybrid systems remains very challenging for non-linear differential systems. In the academic and industrial formal methods and static analysis communities, it was clearly established that reasoning about non-linear differential systems is presently a critical bottleneck and that verification of critical systems deeply need new symbolic techniques with fast numerical approaches to handle such non-linear systems. More recent approaches have been constraint-based [118, 114, 105, 1, 115].

In these approaches, the local differential systems are seen as varieties and their algebraic assertions and their induced ideal J. First, the Gröbner bases of J is computed. Then, a candidate invariant Q is considered. Q is taken with a fixed degree and unknown parametric coefficients, *i.e.*, it is a template form that can be understood as the target invariant to be generated. Then, the normal form reduction $NF_G(Q)$ of G over Q is obtained in order to generate a system $(NF_G(Q) = 0)$ of equations encoding the conditions for invariance, resulting in constraints on the unknown coefficients whose solutions yield invariants. Each single computation steps, i.e., computations of Gröbner bases, normal form reductions of the template and the resolution of the constraints, require a high numbers of operations, and are of double exponential complexity. Moreover the set of constraints they generate remains non-linear when the local continuous rules are non-linear differential systems. Even for linear local continuous rules, the constraints generated could form a very complex non-linear differential system which makes their resolution intractable.

In terms of performance and efficiency, we succeeded in reducing the invariant generation problem for non-linear hybrid systems to linear algebraic problems, i.e. to the computation of eigenspaces of specific morphisms. Each computational step required by our techniques remains of polynomial complexity. We compute first our specific matrices and then we compute their nullspaces. We can also handle non-linear hybrid systems, extended with parameters and variables that are functions of time. We note that these type of hybrid system are still not treated by other state-of-the-art invariant generation methods. Instead of generating an invariant at a time, our approaches are capable of computing an ideal of invariants. Our algorithm embodies a strategy to guess the degree bounds which allow the non-triviality of the computed invariants. It is also important to emphasize the fact that the very general sufficient conditions allowing for the existence and computation of invariant ideals provided in this Chapter could be directly used by any constraint-based invariant generation method [114, 118, 115, 1], or by any analysis methods based on over-approximations and reachability [103, 104, 107]. Our examples show the strenght of our methods by illustrating that they are beyond other current state-of-the-art approaches.

In terms of limitations, our methods generate invariants that are assertions with equality and it would be interesting to generate also inequalities in order to facilitate the development of reachability analysis. Also, the theoretical limit of the techniques proposed in this Chapter can be gauged by the fact that they are sound but not complete (see Section 4.3), as predicted. Also, in Section 4.7 we show how our techniques can handle some local continuous rules involving transcendental functions that are trigonometric, like *cos*, but the proposed techniques can not handle transcendental functions such as *log* or *exp*. Also, the methods presented here generate polynomial equalities and an important first generalisation would be to develop new techniques to be able to generates more precise invariant that could contain transcendental functions (see Chapter 5).

Capítulo 5

Multivariate Formal Power Series and Transcendental Invariant Generation for Non Linear Hybrid Systems

The contribution exposed in this Chapter also appear in our articles [85, 87].

Abstract: We present the first verification methods that automatically generate bases of invariants expressed by *multivariate formal power series* and *transcendental functions*. We also discuss their convergence over hybrid systems that exhibit non linear models.

We reduce the invariant generation problem to linear algebraic matrix systems, from which one can provide effective methods for solving the original problem. More specifically, we obtain very general sufficient conditions for the existence and the computation of formal power series invariants over multivariate polynomial continuous differential systems. The formal power series invariants generated are often composed by the expansion of some well-known transcendental functions like *log* or *exp* and have an analysable closed-form. This facilitates the use of the invariants to verify safety properties. Our examples with non linear continuous evolution, similar to those present today in many critical hybrid embedded systems, show the strength of our results. For some of the examples and we can prove that they do not have "finite"polynomial invariants, a result which is beyond other recent approaches to the problem.

5.1 Introduction

As we have shown in chapter 4, hybrid systems [63, 4] exhibit both discrete and continuous behaviorsMost safety-critical systems, *e.g.* aircraft, automobiles, chemical plants and biological systems, operate as non-linear hybrid systems and can only be adequately modeled by means of non-linear arithmetic over the real numbers and involving multivariate polynomial, fractional or transcendental functions. In this work, we use hybrid automata as computational models for hybrid systems. A hybrid automaton can describe interactions between discrete transitions and continuous dynamics, the latter being governed by local differential equations.

Some known verification approaches are based on inductive invariant generation [77], which can be extended to hybrid systems to verify safety-critical properties. Also, they can provide precise over-approximations of the set of reachable states in the continuous state space. Given that, they can be used to determine which discrete transitions are possible and can also be used to verify if a given property is fulfilled or not.

More recent works strive to generate linear or polynomial invariants over hybrid systems that exhibit affine or polynomial systems as continuous evolution modes. Despite tremendous progress over the past years [118, 125, 111, 114, 1, 115, 107, 3, 104], generating invariants for hybrid systems remains very challenging for non-linear discrete systems, as well as for non-linear differential systems with non-trivial local and initial conditions.

We look for invariants that strengthen what we wish to prove, and so allow us to establish the desired property. In this work, we present new methods for the automatic generation of invariants in the form of assertions where continuous functions are expressed by multivariate formal power series. Such methods can then be applied to systems with continuous evolution modes described by multivariate polynomials or fractional differential rules. As far as we know, there are no other methods that deal with this type of systems or that can automatically generate this type of invariants.

We develop the new methods by first extending our previous work on non-linear invariant generation for discrete models with nested loops and conditional statements that describe multivariate polynomial or fractional systems [80, 79]. Then, we generalize our previous work on non-linear invariant generation for hybrid systems [84, 83, 88, 81].

The contributions of this Chapter are summarized in Section 1.2.3.

This Chapter is organized as follows. In Section 5.2 we first recall the notion of algebraic hybrid systems, and we introduce our notations and representations for multivariate formal series. In Section 5.3 we present new forms for approximating consecution with multivariate formal power series. In Section 5.4 we reduce the problem to triangular linear algebraic matrix systems. In Section 5.5 we provide very general sufficient conditions for the existence of invariants and, further, we show how to automatically compute such invariants. In Section 5.6 we present a convergence analysis and illustrate it in Section 5.7. We show the efficiency of our methods in Section 5.8 by generating closed-form invariants for systems that are intractable by other state-of-the-art formal methods and static analysis approaches. Section 5.9 offers our conclusions.

5.2 Hybrid Systems and Multivariate Formal Power Series

In this subsection, we recall some basic notions.

5.2.1 Hybrid Systems

We use the notion of hybrid automata as the computational model for hybrid systems.

Definition 24. *A* hybrid system *is described by a tuple* $\langle V, V_t, L, \mathcal{T}, \mathcal{C}, \mathcal{S}, l_0, \Theta \rangle$*, where*

- $V = \{a_1, .., a_m\}$ *is a set of parameters.*
- $V_t = \{X_1(t), .., X_n(t)\}$ *where* $X_i(t)$ *is a function of* t*.*
- L *is a set of locations,*
- l_0 *is the initial location.*
- *A* transition $\tau \in \mathcal{T}$ *is given by* $\langle l_{pre}, l_{post}, \rho_\tau \rangle$*, where* $l_{pre} \in L$ *and* $l_{post} \in L$ *name the pre- and post- locations of* τ*, and the* transition relation ρ_τ *is a first-order assertion over* $V \cup V_t \cup V' \cup V_t'$*.*
- Θ *is the initial condition, given as a first-order assertion over* $V \cup V_t$*.*
- $\mathcal{C}$ *maps each location* $l \in L$ *to a* local condition $\mathcal{C}(l)$ *denoting an assertion over* $V \cup V_t$*.*
- *Finally,* $\mathcal{S}$ *associates each location* $l \in L$ *to a* differential rule $\mathcal{S}(l)$ *corresponding to an assertion over* $V \cup \{dX_i/dt | X_i \in V_t\}$*.*
- *A state is any pair from* $L \times \mathbb{R}^{|V|}$*.*

We refer to Chapter 4 and Section 4.2 for more details and examples of hybrid systems and invariant maps. □

5.2.2 Multivariate Formal Power Series

Let W be a hybrid system, and let $\gamma(l)$ be an inductive assertion, as in Definition 20. Recall that an inductive assertion holds at the initial state and at every other possible states in a run. Hence, if γ is an inductive assertion map then $\gamma(l)$ is an invariant at l for W (see Section 4.2 Chapter 4).

Let us describe the continuous evolution rules by a polynomial differential system S of the form:

$$S = \begin{bmatrix} \dot{x}_1(t) = P_1(x_1(t), ..., x_n(t)) \\ \dot{x}_2(t) = P_2(x_1(t), ..., x_n(t)) \\ \vdots \\ \dot{x}_n(t) = P_n(x_1(t), ..., x_n(t)) \end{bmatrix}$$

Definition 25. *A formal power series in the indeterminates $x_1, \dots, x_n$ is an expression of the following form:*

$$\sum_{(i_1,...,i_n)\in\mathbb{N}^n} f_{i_1,...,i_n} x^{i_1} ... x^{i_n},$$

where the coefficients $f_{i_1,...,i_n}$ belong to $\mathbb{R}$. □

Definition 26. *Whenever $i = (i_1, ..., i_n) \in \mathbb{N}^n$, we denote the sum $i_1 + \cdots + i_n$ by $|i|$. We say that an order $<$ is a lexicographical total ordering in $\mathbb{N}^n$ if for any two elements $i = (i_1, ..., i_n)$ and $j = (j_1, ..., j_n)$ in $\mathbb{N}^n$ we have that*

$$(j_1, ..., j_n) < (i_1, ..., i_n)$$

holds if and only if one of the following condition holds:

- *(i) $|j| < |i|$; or*
- *(ii) $|j| = |i|$, and the first non null component of $i - j$ is positive.*

□

With $|i| = k$, where $i = (i_1, ..., i_n)$, the monomials

$$x_1^{i_1} ... x_n^{i_n}$$

, form an ordered basis for the vector space of homogeneous polynomials of total degree k. This means that any homogeneous polynomial of total degree k can be written in the following ordered form:

$$\sum_{|i|=k} f_{i_1,\dots,i_n} x_1^{i_1} \dots x_n^{i_n}.$$

As a consequence, since a formal power series $F(x_1, .., x_n)$ is the direct sum of its homogeneous components, it can be written in the following ordered form:

$$F(x_1, .., x_n) = \sum_{k \geq 1} \sum_{|i|=k} f_{i_1,\dots,i_n} x_1^{i_1} \dots x_n^{i_n}.$$

We will use the following notation [17]: the coefficients of homogeneous polynomials of degree k will be denoted by

$$F_k = \begin{bmatrix} f_{k,0,0,\dots,0} & f_{k-1,1,0,\dots,0} & f_{k-1,0,1,\dots,0} & \cdots & f_{0,0,0,\dots,k} \end{bmatrix}^\top$$

and the basis of homogeneous monomials of degree k will be denoted by the following vector:

$$X^k = [\ {x_1}^k \ {x_1}^{k-1}x_2 \ {x_1}^{k-1}x_3 \ \dots \ {x_n}^k \ ]^\top.$$

where the coordinates are ordered with respect to the lexicographical total ordering given as in Definition 26. With this notation, the formal power series $F(x_1, .., x_n)$ can be written as

$$\sum_{k \geq 1} F_k \cdot X^k = F_1 \cdot X^1 + ... + F_k \cdot X^k + ...,$$

where

$$F_k \cdot X^k$$

denotes the scalar product

$$\langle F_k, X^k \rangle.$$

The polynomial $P_i(x_1, \dots, x_n)$ can thus be written in the form

$$P_i(x_1, \dots, x_n) = P_1^i \cdot X^1 + ... + P_m^i \cdot X^m,$$

where m is the maximal degree among all polynomials P_i, and the P_j^i are the coefficients vector of P_i. Denote by $x(t)$ the vector

$$(x_1(t), ..., x_n(t))^\top.$$

Then S can be written as

$$\dot{x} = A_1 \cdot X^1(t) + ... + A_m \cdot X^m(t),$$

where $A_j = \begin{bmatrix} P_j^1 & \dots & P_j^n \end{bmatrix}^\top$. In particular, A_1 is the $n \times n$ matrix which is actually equal to the *Jacobian* matrix of the polynomial system given by the P_i's at zero.

5.3 New continuous consecutions

Now we show how to encode differential continuous consecution conditions. Let S be a polynomial differential system as in Eq. (5.2.2).

Definition 27. *A function F from $\mathbb{R}^n$ to $\mathbb{R}$ is said to be a λ-invariant for a system S if*

$$\frac{d}{dt}F(x_1(t), ..., x_n(t)) = \lambda F(x_1(t), ..., x_n(t)),$$

for any solution

$$x(t) = (x_1(t), ..., x_n(t))$$

of S. □

Definition 27 encodes the fact that the numerical value of the Lie derivative of F is given by λ times its numerical value throughout the time interval $[0, \varepsilon)$. Without loss of generality we will assume that λ is a constant. It is worth noticing, however, that our methods will also work when λ is a multivariate fractional or multivariate polynomial, as we proposed for the case of multivariate polynomial invariants generation [83, 88, 81].

Now, we want to establish sufficient conditions over S for it to admit λ-invariants which are formal power series.

Note that a formal power series

$$F(x) = F_1 \cdot X^1 + ... + F_k \cdot X^k + ...$$

is a λ-invariant if the following conditions holds:

$$\sum_{i=0}^{n} \frac{\partial F(x)}{\partial x_i} P_i(x) = \lambda F(x). \tag{5.1}$$

Using our notation, we obtain:

$$\sum_{i=0}^{n} \frac{\partial(F_1 \cdot X^1 + ... + F_k \cdot X^k + ...)}{\partial x_i}(P_1^i \cdot X^1 + ... + P_m^i \cdot X^m)$$
$$-\lambda(F_1 \cdot X^1 + ... + F_k \cdot X^k + ...) = 0. \tag{5.2}$$

5.4 Reduction to linear algebra

By directly expanding the left side of Eq. (5.2) and collecting terms corresponding to increasing degrees, we have:

$$(1): \sum_{j=1}^{n} \frac{\partial(F_1X^1)}{\partial x_j} P_1^j X^1 - \lambda F_1 X^1 = 0$$

$$(2): \sum_{j=1}^{n} [\frac{\partial(F_1X^1)}{\partial x_j} P_2^j X^2 + \frac{\partial(F_2X^2)}{\partial x_j} P_1^j X^1] - \lambda F_2 X^2 = 0$$

$$(3): \sum_{j=1}^{n} [\frac{\partial(F_1X^1)}{\partial x_j} P_3^j X^3 + \frac{\partial(F_2X^2)}{\partial x_j} P_2^j X^2 + \frac{\partial(F_1X^1)}{\partial x_j} P_3^j X^3] - \lambda F_3 X^3 = 0$$

$$\vdots \qquad \vdots$$

$$\begin{aligned}(m): \quad & \sum_{j=1}^{n} [\frac{\partial(F_1X^1)}{\partial x_j} P_m^j X^m + \frac{\partial(F_2X^2)}{\partial x_j} P_{m-1}^j X^{m-1} + \cdots + \frac{\partial(F_mX^m)}{\partial x_j} P_1^j X^1] \\ & -\lambda F_m X^m = 0\end{aligned}$$

$$\begin{aligned}(m+1): \quad & \sum_{j=1}^{n} [\frac{\partial(F_2X^2)}{\partial x_j} P_m^j X^m + \frac{\partial(F_3X^3)}{\partial x_j} P_{m-1}^j X^{m-1} + \cdots + \frac{\partial(F_{m+1}X^{m+1})}{\partial x_j} P_1^j X^1] \\ & -\lambda F_{m+1} X^{m+1} = 0\end{aligned}$$

$$\vdots \qquad \vdots$$

The equation corresponding to degree k is:

$$\begin{aligned}&\sum_{j=1}^{n} [\ \frac{\partial(F_{k-min(k,m)+1}X^{k-min(k,m)+1})}{\partial x_j} P_{min(k,m)}^j X^{min(k,m)} \\ &+ \frac{\partial(F_{k-min(k,m)+2}X^{k-min(k,m)+2})}{\partial x_j} P_{min(k,m)-1}^j X^{min(k,m)-1} + \cdots + \frac{\partial(F_kX^k)}{\partial x_j} P_1^j X^1\ ] - \lambda F_k X^k = 0\end{aligned}$$

By taking a different notion of consecution, we can treat more general systems than those that appeared in the determinant analysis of integrability of differential systems in Boularas [17]. Take the linear morphism $D_{p-k,p}$ from $\mathbb{R}_{p-k}[x_1,\ldots,x_n]$ to $\mathbb{R}_p[x_1,\ldots,x_n]$, given by

$$D_{p-k,p}: \left\{ \begin{array}{rcl} \mathbb{R}_{p-k}[x_1,\ldots,x_n] & \to & \mathbb{R}_p[x_1,\ldots,x_n] \\ P(X = x_1,\ldots,x_n) & \mapsto & \sum_{j=1,\ldots,n} (\partial_j P(X)) P_{k+1}^j . X^{k+1} \end{array} \right.$$

which can be represented by matrix $M_{p-k,p}$, in the ordered canonical basis of $\mathbb{R}_{p-k}[x_1,\ldots,x_n]$ and $\mathbb{R}_p[x_1,\ldots,x_n]$, respectively.

Its l-th column is the decomposition of the polynomial

$$\sum_{j=1,\ldots,n} (\partial_j P(X)) P_{k+1}^j . X^{k+1},$$

where $P(X)$ is the l-th monomial in the ordered basis

$$\left\{ x_1^p, x_1^{p-1}x_2, x_1^{p-1}x_3, \ldots, x_n^p \right\}.$$

We can reduce the infinite system, described just above, to the following linear algebraic system:

$$\begin{cases} (M_{1,1} - \lambda I_2)F_1 = 0 \\ M_{1,2}F_1 + (M_{2,2} - \lambda I_2)F_2 = 0 \\ M_{1,3}F_1 + M_{2,3}F_2 + (M_{3,3} - \lambda I_4)F_3 = 0 \\ \vdots \\ M_{k-min(k,m)+1,k}F_{k-min(k,m)+1} + M_{k-min(k,m)+2,k}F_{k-min(k,m)+2} \\ + \cdots + (M_{k,k} - \lambda I_{k+1})F_k = 0 \\ \vdots \end{cases} \tag{5.3}$$

By the definition of $D_{p-k,p}$, we can symbolically compute all the matrices $M_{m,n}$. We will use the following result.

Lemma 5. *Assume that matrix $A = M_{1,1}$ is triangular,i.e.*

$$A = \begin{bmatrix} \lambda_1 & & & & \\ \star & \lambda_2 & & & \\ \star & \star & \ddots & & \\ \star & \star & \star & \lambda_{n-1} & \\ \star & \star & \star & \star & \lambda_n \end{bmatrix}.$$

Then $M_{p,p}$ is also triangular with diagonal terms

$$i_1\lambda_1 + \cdots + i_n\lambda_n,$$

where $i_1 + \cdots + i_n = p$. □

Demonstração. In this case,

$$P_1^j.X^1 = \lambda_j x_j + a_{j,j+1}x_{j+1} + \cdots + a_{j,n}x_n.$$

Now consider the monomial basis

$$P(X) = x_1^{i_1} \dots x_n^{i_n},$$

where $i_1 + \cdots + i_n = p$.

One has

$$\begin{array}{rcl} D_{p,p}(X) & = & i_1 x_1^{i_1-1}\dots x_n^{i_n}(\lambda_1 x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n) \\ & & +i_2 x_1^{i_1} x_2^{i_2-1}\dots x_n^{i_n}(\lambda_2 x_2 + a_{2,3}x_3 + \cdots + a_{2,n}x_n) \\ & & +\cdots+ \\ & & +i_n x_1^{i_1}\dots x_n^{i_n-1}(\lambda_n x_n) \\ & = & (i_1\lambda_1 + \cdots + i_n\lambda_n)x_1^{i_1}\dots x_n^{i_n} + \Omega \end{array},$$

where Ω is a sum of higher terms monomials that come after $x_1^{i_1}\dots x_n^{i_n}$ in the ordered basis of $R_p[x_1,\dots,x_n]$.

Then, matrix $M_{p,p}$ corresponding to $D_{p,p}$ in the canonical ordered basis of $R_p[x_1,\dots,x_n]$, is:

$$\begin{bmatrix} p\lambda_1 & & & & & & \\ \star & (p-1)\lambda_1+\lambda_2 & & & & & \\ \star & \star & \ddots & & & & \\ \star & \star & \star & \sum_{k=1}^{n} i_k\lambda_k & & & \\ \star & \star & \star & \star & \ddots & & \\ \star & \star & \star & \star & \star & \lambda_{n-1}+(p-1)\lambda_n & \\ \star & \star & \star & \star & \star & \star & p\lambda_n \end{bmatrix}$$

Thus, it is also triangular with diagonal terms

$$i_1\lambda_1 + \cdots + i_n\lambda_n,$$

where $i_1 + \cdots + i_n = p$. □

5.5 Sufficient general existence conditions and the computation of invariants

First, we show what happens when a λ-invariant converges. Next, we examine the computation of λ-invariants.

Theorem 23. (Soundness) *Let F be a λ-invariant for a system S.*

Let U be an open subset of $\mathbb{R}^n$, where F is defined by a normally convergent power series.

If there is an initial condition

$$x_1(0),...,x_n(0)$$

in U such that

$$F(x_1(0),...,x_n(0)) = 0,$$

then

$$F(x_1(t), ..., x_n(t)) = 0$$

for all t such that $x_1(t), ..., x_n(t)$ remain in U, i.e., F is an invariant of S for the initial condition

$$x_1(0), ..., x_n(0).$$

□

Demonstração. As the power series defining F converges normally on U, so does any of its derivatives. Thus,

$$\dot{F}(x_1(t), ..., x_n(t)) = \sum_{i=1}^{n} \partial_i F(x_1(t), ..., x_n(t))\dot{x_i}(t) = \lambda F(x_1(t), ..., x_n(t))$$

because of the λ-invariant property.

So, $F(x_1(t), ..., x_n(t))$ must be equal to

$$t \mapsto ke^{\lambda t}$$

for some constant k. But as

$$F(x_1(0), ..., x_n(0)) = 0,$$

then k is zero, and so is $F(x_1(t), ..., x_n(t))$ for any t s.t.

$$(x_1(t), \ldots, x_n(t)) \in U.$$

□

5.5.1 Sufficient general existence conditions

We obtain the following main results on the existence of formal power series invariants for systems S, described as in Eq. (5.2.2).

Theorem 24. *Let A be the Jacobian matrix at zero of the polynomial*

$$P = (P_1, ..., P_n)$$

defining the system S. Its expression is:

$$(\partial_i P_j(0, ..., 0), i, j \in [1, n]^2).$$

Let $P_k(0, .., 0) = 0$. If A is triangularizable with eigenvalues

$$\lambda_1 \leq ... \leq \lambda_n,$$

then there exists a λ-invariant formal power series for S when all eigenvalues are positive, or are all negative, with

$$\lambda = \lambda_1.$$

□

Demonstração. Up to a linear change of variables, we can assume that matrix A is triangular with diagonal terms

$$\lambda_1 \leq ... \leq \lambda_n.$$

We know that matrix $M_{k,k}$ has the form described in Lemma 5. As A is triangular, so is $M_{k,k}$, and its diagonal terms are the real numbers

$$i_1\lambda_1 + \cdots + i_n\lambda_n,$$

where

$$i_1 + \cdots + i_n = k.$$

Hence, the diagonal terms of

$$M_{k,k} - \lambda I_{k+1}$$

are

$$0 \leq \lambda_2 - \lambda ... \leq \lambda_n - \lambda$$

when $k = 1$. Also, it has a nonzero kernel, and so we can chose a nonzero F_1, such that

$$(M_{1,1} - \lambda I_2)F_1 = 0.$$

For $k \geq 2$ and $i_1 + \cdots + i_n = k$, the diagonal terms

$$i_1\lambda_1 + \cdots + i_n\lambda_n - \lambda$$

of the triangular matrix

$$M_{k,k} - \lambda I_{k+1}$$

are greater than

$$i_1\lambda_1 + \cdots + i_n\lambda_n - \lambda = k\lambda - \lambda > \lambda > 0.$$

So, $M_{k,k} - \lambda I_{k+1}$ is invertible.

Hence we can choose:

$$F_2 = -(M_{2,2} - \lambda I_3)^{-1} M_{1,2} F_1,$$

and then

$$F_3 = -(M_{3,3} - \lambda I_4)^{-1}(M_{1,3}F_1 + M_{2,3}F_2),$$

and recursively,

$$F_k = -(M_{k,k} - \lambda I_{k+1})^{-1}(M_{k-min(k,m)+1,k}F_{k-min(k,m)+1} + \cdots + M_{k-1,k}F_{k-1}).$$

Then,

$$(F_1, F_2, \dots)$$

is a nonzero solution of the system and the formal power series

$$\sum_i F_i X^i$$

is a λ-invariant. □

The proof of the preceding important theorem, also describe a method for the resolution of the triangular matrix system.

We are then able to generate nonzero formal power series

$$\sum_i F_i X^i$$

which are λ-invariants associated to the nonzero solution

$$(F_1, F_2, \dots).$$

In the examples we used Maple to compute the matrix products necessary to obtain F_k in its symbolic form.

We treat the case when all eigenvalues are negative in a similar way. That is, with $\lambda = \lambda_n$ ..., λ will be the eigenvalue with the minimum absolute value. Also, we recall that triangularizable matrices of $M_n(\mathbb{R})$ form a dense open subset of total measure of $M_n(\mathbb{R})$.

5.5.2 Inductive invariants and initial conditions

We state the following important result.

Theorem 25. *Let A be the Jacobian matrix at zero of the polynomial*

$$P = (P_1, ..., P_n)$$

defining a system S, as in Eq. (5.2.2), and whose expression is

$$(\partial_i P_j(0, ..., 0),\ i, j \in [1, n]^2).$$

Assume, further, that $P_k(0, .., 0) = 0$.

Suppose that A is triangularizable with eigenvalues

$$\lambda_1 \leq ... \leq \lambda_n.$$

Denote λ_1 by λ and assume that the eigenspace associated with λ is of dimension at least 2. Let F_1 and F_2 be two independent λ-invariants.

If there is an open subset U of $\mathbb{R}^n$, over which F_1 and F_2 define two normally convergent power series then, for any initial value

$$(x_{1,0}, \ldots, x_{n,0}),$$

the power series

$$F_2(x_{1,0}, \ldots, x_{n,0})F_1 - F_1(x_{1,0}, \ldots, x_{n,0})F_2$$

defines an inductive invariant on U for the solution of S with initial conditions

$$x_1(0) = x_{1,0}, \ldots, x_n(0) = x_{n,0}.$$

□

Demonstração. Both F_1 and F_2 converge to a solution

$$(x_1(t), \ldots, x_n(t))$$

with initial values $(x_{1,0}, \ldots, x_{n,0})$ in U.

Hence, it must stay in U for small t. Moreover, since F_1 and F_2 are independent,

$$F = F_2(x_{1,0}, \ldots, x_{n,0})F_1 - F_1(x_{1,0}, \ldots, x_{n,0})F_2$$

is a nonzero λ-invariant which vanishes at $(x_{1,0}, .., x_{n,0})$.

So, according to Theorem 23, F is an inductive invariant □

The methods presented so far automatically generate bases for non trivial multivariate formal power series invariants for each differential rule associated to locations in the hybrid automaton.

In order to handle the discrete transition relations when generating global invariants we can use the methods proposed in our previous works [83, 88, 81] which were presented in the previous Chapter. They generate finite polynomial invariant for such hybrid systems.

5.6 Triangularizable systems

Now we show how to treat the following general system with parameters a, b, c, $a_{1,1}$, $a_{1,2}$, $a_{2,2}$, $b_{1,1}$,$b_{1,2}$, $b_{2,2}$ in V, and variables x, y in V_t:

$$\begin{cases} \dot{x}(t) &= ax(t) + by(t) + a_{1,1}x^2(t) + a_{1,2}x(t)y(t) + a_{2,2}y^2(t) \\ \dot{y}(t) &= cy(t) + b_{1,1}x^2(t) + b_{1,2}x(t)y(t) + b_{2,2}y^2(t) \end{cases}$$

The Jacobian matrix at zero of the polynomials defining the system is

$$\begin{pmatrix} a & 0 \\ b & c \end{pmatrix}.$$

From Theorem 24, we already know how to find a formal power series F which is an a-invariant. Looking more closely at the coefficients of such a series we will show that it must converge in some appropriate neighborhood of 0. In this section it is not necessary to assume that $a > c$ or $a < c$, since we will choose $\lambda = min\{a, c\}$.

In Subsections 5.6.1, 5.6.2 and 5.6.3 we show how to generate λ-invariants which take initial condition into account, by applying Theorem 24. We stated $a = c$ only at subsection 5.6.4 to show that when Theorem 25 apply, the λ-invariants hold whatever the initial conditions are.

5.6.1 The matrices $M_{p-k,p}$

Using our notation, we have $P_1^i = 0$ and $P_2^i = 0$ for all $i > 0$. Then $M_{p-k,p}$ is the matrix whose l-th column is the vector corresponding to the decomposition of the polynomial

$$\begin{array}{l} \partial_1[(0,...,0, \underbrace{1}_{l-th\ position}, 0,...,0)X^{p-k}]P_{k+1}^1 X^{k+1} \\ +\partial_2[(0,...,0, \underbrace{1}_{l-th\ position}, 0,...,0)X^{p-k}]P_{k+1}^2 X^{k+1} \end{array}.$$

in the ordered canonical basis of $\mathbb{R}_p[x, y]$.

Here, the polynomial

$$(0, .., 0, \underbrace{1}_{l-th\ place}, 0, .., 0)X^{p-k}$$

is the l-th monomial of the canonical basis of $\mathbb{R}_{p-k}[x, y]$.

Therefore, the matrices $M_{p-k,p}$ are zero unless $k = 0$ or $k = 1$. When $k = 0$, the general form of $M_{p,p}$ is given in Section 5.4 and, in our particular case, it is

$$\begin{pmatrix} pa & & & & & \\ p.b & (p-1)a+c & & & & \\ & (p-1)b & (p-2)a+2c & & & \\ & & \ddots & \ddots & & \\ & & & 2b & a+(p-1)c & \\ & & & & b & pc \end{pmatrix}.$$

Note that $p+1$ is actually the dimension of $\mathbb{R}_p[x,y]$ and so $M_{p-1,p}$ is rectangular with $p+1$ rows and p columns. Here, the l-th monomial in the basis of $\mathbb{R}_{p-1}[x,y]$ is $x^{p-l-1}y^l$. Also, the polynomial $P_2^1X^2$ is

$$a_{1,1}x^2 + a_{1,2}xy + a_{2,2}y^2$$

and the polynomial $P_2^2X^2$ is

$$b_{1,1}x^2 + b_{1,2}xy + b_{2,2}y^2.$$

Hence, matrix $M_{p-1,p}$ can be written as:

$$\begin{pmatrix} (p-1)a_{1,1} & b_{1,1} & & & & \\ (p-1)a_{1,2} & (p-2)a_{1,1}+b_{1,2} & 2b_{1,1} & & & \\ (p-1)a_{2,2} & (p-2)a_{1,2}+b_{2,2} & (p-3)a_{1,1}+2b_{1,2} & 3b_{1,1} & & \\ & \ddots & \ddots & \ddots & \ddots & \\ & & 3a_{2,2} & 2a_{1,2}+(p-3)b_{1,2} & a_{1,1}+(p-2)b_{1,2} & (p-1)b_{1,1} \\ & & & 2a_{2,2} & a_{1,2}+(p-2)b_{2,2} & (p-1)b_{1,2} \\ & & & & a_{2,2} & (p-1)b_{2,2} \end{pmatrix}.$$

5.6.2 Resolution of the infinite system

We are looking for λ-scale invariants and we know that we can choose $\lambda = min\{a,c\}$.

Then, the system to solve is

$$\begin{cases} (M_{1,1}-\lambda I_2)F_1 = 0 \\ M_{1,2}F_1 + (M_{2,2}-\lambda I_3)F_2 = 0 \\ M_{2,3}F_2 + (M_{3,3}-\lambda I_4)F_3 = 0 \\ \vdots \\ M_{k-1,k}F_{k-1} + (M_{k,k}-\lambda I_{k+1})F_k = 0 \\ \vdots \end{cases} \tag{5.4}$$

It can be written as:

$$\begin{cases} (M_{1,1}-\lambda I_2)F_1 = 0 \\ F_2 = -(M_{2,2}-\lambda I_3)^{-1}M_{1,2}F_1 \\ F_3 = -(M_{3,3}-\lambda I_4)^{-1}M_{2,3}F_2 \\ \vdots \\ F_k = -(M_{k,k}-\lambda I_{k+1})^{-1}M_{k-1,k}F_{k-1} \\ \vdots \end{cases} \tag{5.5}$$

One can choose any F_1, and then let

$$F_k = (-1)^{k+1}U_k(F_1),$$

where U_k is the matrix with $k+1$ rows and 2 columns:

$$\begin{array}{l} [\ (M_{k,k}-\lambda I_{k+1})^{-1}M_{k-1,k}] \cdot [(M_{k-1,k-1}-\lambda I_k)^{-1}M_{k-2,k-1}] \\ \cdots\cdots [(M_{3,3}-\lambda I_4)^{-1}M_{2,3}] \cdot [(M_{2,2}-\lambda I_3)^{-1}M_{1,2}\ ] \end{array} \tag{5.6}$$

Then, $M_{k,k}-\lambda I_{k+1}$ is

$$\begin{pmatrix} ka-\lambda & & & & \\ k.b & (k-1)a+c-\lambda & & & \\ & (k-1)b & (k-2)a+2c-\lambda & & \\ & \ddots & \ddots & & \\ & 2b & a+(k-1)c-\lambda & & \\ & & b & kc-\lambda \end{pmatrix}$$

which can be decomposed as the product DT:

$$\begin{pmatrix} d_1 & & & & & \\ & d_2 & & & & \\ & & d_3 & & & \\ & & & \ddots & & \\ & & & & d_k & \\ & & & & & d_{k+1} \end{pmatrix} \begin{pmatrix} 1 & & & & & \\ t_2 & 1 & & & & \\ & t_3 & 1 & & & \\ & & \ddots & \ddots & & \\ & & & t_k & 1 & \\ & & & & t_{k+1} & 1 \end{pmatrix},$$

where

$$d_i = (k+1-i)a + (i-1)c - \lambda$$

and

$$t_j = (k+2-j)b/d_j.$$

So,

$$(M_{k,k}-\lambda I_{k+1})^{-1} = T^{-1}D^{-1},$$

where D^{-1} has the obvious form and T^{-1} is

$$\begin{pmatrix} 1 & & & & & \\ -t_2 & 1 & & & & \\ t_2t_3 & -t_3 & 1 & & & \\ -t_2t_3t_4 & t_3t_4 & -t_4 & 1 & & \\ \star & \star & \star & \star & \star & \\ (-1)^k t_2 \dots t_{k+1} & (-1)^{k-1} t_3 \dots t_{k+1} & \dots & t_k t_{k+1} & -t_{k+1} & 1 \end{pmatrix}.$$

5.6.3 Convergence of the λ-invariant

We want to show that if $\lambda > 2b$, the coefficients of the F_i vectors decrease quickly enough so that the invariant F converges in a neighborhood of zero.

Let us first recall some basic properties of norms in finite dimension real vector spaces, as well as the associated matrix norms. If v, with coordinates v_i, belongs to $\mathbb{R}^n$, we denote by $|v|_\infty$ the value $\max_{i=1,\dots,n} |v_i|$. If A is a matrix with m rows and n columns, representing a morphism from $(\mathbb{R}^n, |.|_\infty)$ to $(\mathbb{R}^m, |.|_\infty)$ in the canonical basis, it is well-known that associated with the norm $|.|_\infty$ is the matricial norm $||.||$ on $M_{m,n}(\mathbb{R})$, where

$$||A|| = \max_{i=1,\dots,m} (\sum_{j=1}^{n} |A_{i,j}|).$$

Moreover, using this norm, if $v \in \mathbb{R}^n$ then one has that

$$|Av|_\infty \leq ||A||.|v|_\infty.$$

This implies that if A and B are two matrices belonging, respectively, to $M_{m,n}(\mathbb{R})$ and $M_{n,p}(\mathbb{R})$, then one has that

$$||AB|| \leq ||A|| \cdot ||B||.$$

In particular,

$$||U_k|| \leq ||M_{k,k} - \lambda I_{k+1}| \cdot |||M_{k-1,k}|| \dots ||M_{2,2} - aI_3|| \cdot ||M_{1,2}||.$$

But, from the expressions for the $M_{k-1,k}$ matrices, we have that

$$||M_{k-1,k}|| \leq f(k-1),$$

where

$$f = 4 \cdot \max(|a_{i,j}|, |b_{i',j'}|).$$

From the preceding paragraph again, we deduce that

$$||(M_{k,k} - \lambda I_{k+1})^{-1}|| \leq ||D^{-1}|| \cdot ||T^{-1}||.$$

But

$$||D^{-1}|| = \max_i(d_i^{-1}) \leq [(k-1)\lambda]^{-1},$$

because $\lambda = \min(a, c)$, and so

$$||T^{-1}|| = \max_i(1 + t_i + t_{i-1}t_i + \cdots + t_2t_3 \ldots t_{i-1}t_i).$$

But each t_j is less than

$$(k+2-j)b/d_j \leq kb/[(k-1)\lambda] \leq 2b/\lambda.$$

Suppose now that $\lambda > 2b$. Then

$$||T^{-1}|| \leq 1 + 2b/\lambda + \cdots + (2b/\lambda)^k \leq 1/(1-2b/\lambda).$$

By letting e be the constant

$$1/(1-2b/\lambda),$$

we can write

$$||(M_{k,k} - \lambda I_{k+1})^{-1}|| \leq e/(k-1)\lambda.$$

Finally, $||U_k||$ is less than

$$(ef/\lambda)^{k-2} = r^{k-2}.$$

Eventually,

$$|F_k|_\infty = |U_k(F_1)|_\infty \leq ||U_k||.|F_1|_\infty \leq r^{k-2}|F_1|_\infty.$$

Let t be $\max\{|x|, |y|\}$. Then,

$$|F(x,y)| \leq |F_1X^1| + |F_2X^2| \cdots + |F_kX^k| + \cdots \leq 2|F_1|_\infty t + 3|F_2|_\infty t^2 + \cdots + (k+1)|F_k|_\infty t^k + \ldots$$

The right part of the inequality is itself inferior to

$$1/r^2|F_1|_\infty[2(rt) + 3(rt)^2 + \cdots + (k+1)(rt)^k + \ldots],$$

which, from the classical theory of one variable power series, is convergent in the open disk centered at zero and of radius $1/r$. Hence, we have proved the following.

Proposition 2. *Consider the system described at the beginning of Section 5.6 with a and c positive and strictly greater than $2b$. Let λ be the minimum between a and c. Then there exists a λ-invariant, obtained as described in Theorem 24, and which always converges in a neighborhood of zero.* □

5.6.4 The case of eigenspaces with dimension 2

Now, suppose that the eigenspace corresponding to λ has multiplicity 2, *i.e.* $a = c = \lambda > 0$ and $b = 0$.

We know, from the previous subsection, that any λ-invariant will converge in a ball of radius $1/r$ and centered at zero. Moreover, according to Theorem 25, this will give an inductive invariant for the system, for any initial solutions within this ball.

More precisely, by letting

$$F_1^1 = (1, 0)^\top$$

and

$$F_1^2 = (0, 1)^\top,$$

we get a basis $F^1(x, y)$ and $F^2(x, y)$ of λ-invariants that converge in the open $|.|_\infty$-disk of radius $1/r$ and centered at zero.

Note that the monomial of degree one in the Taylor series of F^1 is x, and it is y in the Taylor series of F^2. In other words, if we take the first coefficient of F as $(1, 0)^\top$, we obtain a λ-invariant $F = F^1(x, y)$ and, similarly, if we take the second coefficient of F as $(0, 1)^\top$, we obtain another λ-invariant $F = F^2(x, y)$. Moreover, these two invariants form a basis for invariants that converge in the open $|.|_\infty$-disk of radius $1/r$ and centered at zero.

Assume now that we are given initial values, $x(0) = x_0$ and $y(0) = y_0$, as solutions in this open disk. Then, there will always exist two real numbers, λ and μ, such that

$$\lambda(x_0, y_0)F^1(x_0, y_0) + \mu(x_0, y_0)F^2(x_0, y_0) = 0,$$

where

$$\lambda(x_0, y_0) = F^2(x_0, y_0)$$

and

$$\mu(x_0, y_0) = -F^1(x_0, y_0).$$

Then,

$$\lambda(x_0, y_0)F^1 + \mu(x_0, y_0)F^2$$

is an invariant for the solution corresponding to the initial condition (x_0, y_0).

So, given (x_0, y_0) in the $|.|_\infty$-disk of radius $1/r$ and centered at zero, the invariant depends smoothly on the initial condition.

5.7 Running example

In this section, we discuss a running example and explain how the sufficient conditions for invariance are used, and how a basis for invariant ideals are automatically obtained. We treat sub-classes of non linear differential rules that we often find in local continuous modes in hybrid systems.

More specifically, we show how our method applies to the systems:

$$\begin{bmatrix} \dot{x}(t) = ax(t) + bx(t)y(t) \\ \dot{y}(t) = ay(t) + dx(t)y(t) \end{bmatrix}.$$

In the sequel, we proceed trace, step by step, how to generate invariant ideals.

The Jacobian matrix at zero for the system is

$$\begin{pmatrix} a & 0 \\ 0 & a \end{pmatrix}.$$

Hence, from Theorem 24, we already know that we can find a formal power series F which is an a-invariant. We will show that it must converge in some neighborhood of 0.

5.7.1 The matrices $M_{p-k,p}$

The coefficient vectors P_i are zero, for $i \geq 2$. So, $M_{p-k,p}$ is the matrix whose l-th column is the vector corresponding to the decomposition of the polynomial

$$\begin{aligned} &\partial_1[(0,\dots,0,\underbrace{1}_{l-th\ position},0,\dots,0)X^{p-k}]P^1_{k+1}X^{k+1} \\ +&\partial_2[(0,\dots,0,\underbrace{1}_{l-th\ position},0,\dots,0)X^{p-k}]P^2_{k+1}X^{k+1} \end{aligned}$$

in the ordered canonical basis of $\mathbb{R}_p[x,y]$. Hence, in this case, the matrices $M_{p-k,p}$ are zero unless $k=0$ or $k=1$. When $k=0$, the general form of $M_{p,p}$, as detailed in Section 5.4, is given by

$$paI_{p+1}.$$

But $p+1$ is actually the dimension of $\mathbb{R}_p[x,y]$ and so $M_{p-1,p}$ is rectangular with $p+1$ rows, and p columns. Then the l-th monomial of the basis of $\mathbb{R}_{p-1}[x,y]$ is $x^{p-l-1}y^l$. Then, the polynomial $P^1_2X^2$ is bxy, and the polynomial $P^2_2X^2$ is dxy.

Hence, $\partial_1[(0,..,0,\overbrace{1}^{l-th\ position},0,..,0)X^{p-1}]P^1_2X^2+$

$$\partial_2[(0,..,0,\overbrace{1}^{l-th\ position},0,..,0)X^{p-1}]P^2_2X^2$$

reduces to

$$b(p-l-1)x^{p-l-1}y^{l+1} + dlx^{p-l}y^l.$$

Eventually, it can be seen that the matrix can be written as:

$$M_{p-1,p} = \begin{pmatrix} 0 & & & & & & \\ (p-1)b & d & & & & & \\ & (p-2)b & 2d & & & & \\ & & \ddots & \ddots & & & \\ & & & 2b & (p-2)d & & \\ & & & & b & (p-1)d & \\ & & & & & 0 & \end{pmatrix}.$$

5.7.2 Resolution of the infinite system

When looking for λ-scale invariants, we already know that we must choose $\lambda = a$. Then, we need to solve the following:

$$\begin{cases} (M_{1,1} - aI_2)F_1 = 0 \\ M_{1,2}F_1 + (M_{2,2} - aI_2)F_2 = 0 \\ M_{2,3}F_2 + (M_{3,3} - aI_3)F_3 = 0 \\ \vdots \\ M_{k-1,k}F_{k-1} + (M_{k,k} - aI_k)F_k = 0 \\ \vdots \end{cases} \tag{5.7}$$

As the matrix $M_{k,k}$ is equal to kaI_{k+1}, the system becomes:

$$\begin{cases} 0 \cdot F_1 = 0 \\ F_2 = -a^{-1}M_{1,2}F_1 \\ F_3 = -(2a)^{-1}M_{2,3}F_2 \\ \vdots \\ F_k = -[(k-1)a]^{-1}M_{k-1,k}F_{k-1} \\ \vdots \end{cases} \tag{5.8}$$

This means that one can choose any F_1, and then choose F_k as

$$(-1)^{k+1}a^{-k+1}U_k(F_1),$$

where U_k is the matrix with $k+1$ rows and 2 columns given by the product

$$[1/(k-1)M_{k-1,k}] \cdot [1/(k-2)M_{k-2,k-1}] \ldots [1/2M_{2,3}]M_{1,2}.$$

5.7.3 Convergence of the invariant

Now we show that the invariant F converges in a neighborhood of zero.

In particular, the norm $||U_k||$ is less than or equal to the product

$$\frac{1}{(k-1)!}||M_{k-1,k}||\dots||M_{1,2}||.$$

The expression of $M_{k-1,k}$ gives

$$||M_{k-1,k}|| \leq ck,$$

where $c = \max\{|b|, |d|\}$. Hence,

$$||U_k|| \leq ck!/(k-1)! = ck.$$

Eventually, we get

$$|F_k|_\infty = a^{-k+1}|U_k(F_1)|_\infty \leq a^{-k+1}||U_k||.|F_1|_\infty \leq \frac{ck}{a^{k-1}}|F_1|_\infty.$$

Let t be $\max\{|x|, |y|\}$. Then,

$$\begin{aligned}|F(x,y)| &\leq |F_1X^1| + |F_2X^2| \dots + |F_kX^k| + \dots \\ &\leq 2|F_1|_\infty t + 3|F_2|_\infty t^2 + \dots + (k+1)|F_k|_\infty t^k + \dots\end{aligned}$$

The right side of the inequality is inferior to

$$ac|F_1|_\infty[2(\frac{t}{a}) + 3.2(\frac{t}{a})^2 + \dots + (k+1)k(\frac{t}{a})^k + \dots].$$

From the classical theory of one variable power series, it must converge in the open disk of radius a and centered at zero.

More precisely, taking F^1 and F^2, respectively, as $(1,0)^\top$ and $(0,1)^\top$, we get a basis $F^1(x,y)$ and $F^2(x,y)$ for a-invariants of the system, and which converge in the open $|.|_\infty$-disk of radius a and centered at zero.

Assume now that we are given initial values $x(0) = x_0$ and $y(0) = y_0$ for solutions of the system within this open disk.

Then, there will always exist two real numbers λ and μ, such that

$$\lambda(x_0,y_0)F^1(x_0,y_0) + \mu(x_0,y_0)F^2(x_0,y_0) = 0,$$

where

$$\lambda(x_0,y_0) = F^2(x_0,y_0)$$

and

$$\mu(x_0,y_0) = -F^1(x_0,y_0).$$

Then,

$$\lambda(x_0, y_0)F^1 + \mu(x_0, y_0)F^2 = 0$$

is an invariant corresponding to the initial condition (x_0, y_0). It is also clear that, for (x_0, y_0) in the $|.|_\infty$-disk of radius a and center at zero, it depends smoothly on the initial condition.

Note that, for these classes of systems we obtained a larger region of convergence than the one found in the previous Section.

5.8 Transcendental invariants generation in closed-form

Here is an example where our method exhibits a transcendental invariant. Most importantly, note that this kind of results can not be obtained via the classical constant, polynomial or fractional scale methods. Moreover, the invariant obtained converge everywhere.

The formal power series invariant generated are often composed by expansion of some well-known transcendental function and hence has an analysable closed form. Moreover, being able of computing closed forms for the invariants allows us to reason symbolically about formal power series. This facilitates the use of the invariants to verify properties.

Consider the system

$$\begin{cases} \dot{x(t)} = ax(t) \\ \dot{y(t)} = ay(t) + bx(t)y(t). \end{cases} \tag{5.9}$$

According to previous Section 5.6, we start by computing the matrix $U_k(F_1)$, which gives

$$1/[(k-1)!]M_{k-1,k} \dots M_{1,2}(F_1)$$

when F_1 is $(0, 1)$ and $(1, 0)$. One easily checks that for such a system, $U_k((1, 0))$ is the zero vector for $k \geq 2$, and $U_k((0, 1))$ is equal to

$$1/[(k-1)!](0, d^{k-1}, 0, \dots, 0)$$

for $k \geq 2$. Hence F_k is zero when $k \geq 2$ and F_k is

$$(-d/a)^{k-1}/[(k-1)!](0, 1, 0, \dots, 0).$$

Hence (Section 5.6.3), the power series

$$F^1(x, y) = x$$

and

$$F^2(x,y)=\sum_{k\geq 1}(-d/a)^{k-1}/[(k-1)!]x^{k-1}y=e^{-dx/a}y$$

form a basis of the vector space of a-invariants.

Finally, for any given initial value (x_0, y_0), the following assertion

$$e^{-dx_0/a}y_0x-x_0e^{-dx/a}y=0$$

is an inductive invariant whatever are the initial conditions, i.e. for all x_0 and y_0. Clearly it depends smoothly on the initial value and is convergent everywhere.

5.9 Chapter Discussions: Performances and Limitations

Invariant generation problems for the continuous time state evolution is the most challenging step in static analysis and verification of hybrid systems. We know that, in order to verify safety properties expressed with transcendental functions and to reason symbolically about formal power series, one need first to be able to generate formal power series invariants. So, before looking for automated reasoning techniques for transcendental functions one need to know how to generate inductive invariants expressed by formal power series. In this Chapter, we presented methods which generate bases of *multivariate formal power series and transcendental invariants* for hybrid systems with non-linear behavior.

As for originality, the problem of generating power series invariants and the results are clearly novel. Importantly, there is no other known methods that generate this type of invariants.

As for efficiency, we used linear algebra methods which do not require several Gröbner Basis computation or quantifier eliminations. To show the strength and the performance of our results we recall that we generated transcendental invariants for some non-linear systems while we can prove that these non-linear systems do not have "finite"polynomial invariants, i.e. beyond recent approaches limits. Also, we remark that all the elements of the generated basis are not essentially infinite transcendental invariants and some of them are simple finite polynomials. In other words, one could also generate finite invariants for systems that could not be treated by polynomial or fractional scaling consecution encodings.

We limited our work to the development of formal methods that can generate formal power series invariants for non-linear hybrid systems. These invariants are very precise and would allow precise reachability analysis and safety verification of properties expressed using transcendental functions. But we need to associate new automatic reasoning techniques to be able to reason symbolically about formal power series and to check for safety properties of this kind. One could then extend this words investigating several existing computational algebraic techniques and tools, like Maple or Mathematica, that manipulate formal power series. Also, we saw that the formal power series obtained using our methods are expressed in a symbolic way, but the identification of closed-forms will depend on the limits of orthogonal techniques used by the tools to be considered.

Capítulo 6

New domain of applications

The contributions exposed in this chapter also appear in our articles [93, 108, 95].

Abstract: In our days, substential social infrastructures rely on computer security and privacy: a malicious intent to a computer is a threat to society. In this chapter we provide powerful a *theoretical basis* for the design of static and dynamic platforms that can exhibit a suitable architecture for automatic in-depth malware analysis.

We show how formal methods involving programs static and dynamic analysis can be used to build such architectures. We propose *automatic semantic aware* detection, identification and model extraction methods, hereby circumventing difficulties met by other recent approaches to malware detection.

We propose a new approach to detect and identify malware by automatically generating invariants directly from the specified malware code. Such invariants that we call *malware-invariants*, can then be used as *semantic aware* signatures. Importantly, these invariants would remain unchanged in most of the *obfuscated* versions of the code.

Then, we propose *host-based intrusion detection systems*, using automatically tools, where system calls are guarded by pre-computed invariants. In this way, we can report any deviations observed during the execution of the application. Our methods also provides techniques for the detection of logic bugs and vulnerabilities in applications.

We prove that any static analysis based malware or intrusion detection system will be strongly re-enforced by the presence of pre-computed invariants, and will also be weakened by their absence.

6.1 Introduction

Invariant properties are assertions, expressed in a specified logic, that hold true on every possible run of a system. A *malware* is a program that has malicious intent. Examples of such programs include viruses, Trojans horses, and worms. Malicious intent to computers can be virulent threats to society. We deeply need to understand *malicious behaviors* in more detail.

All present security systems, like anti-virus and other detection systems, suffer from a lack of automation in their malware analysis. In order to provide automatic in-depth malware analysis and precise detection systems, one needs to be able to automatically extract the malicious behaviors, and not just its syntactic signature.

Current malware detectors are "*signature-based*": the presence of the malicious behavior is detected if the malicious code matches some byte-signature. Such current malware detectors are based on sound methods, with byte-signatures located in a database of regular expressions which specify byte or instructions sequences. But the main problem is that malware writers can then use *Obfuscation* [97] to evade current detectors. To evade detection, hackers frequently use obfuscation to morph malware. By so doing, they can evade detection by injecting code into malwares in a way that preserves malicious behavior and makes the previous signature irrelevant. Further, current intrusion detection systems are based on too coarse abstraction methods.

The number of malwares variants that use obfuscation increases exponentially each time a new malware type appear. Malware writers can easily generate new undetectable viruses and, then, the anti-virus code has to update its signature database very frequently to be able to catch the new virus. The main difficulty remain in the update procedures, because the new malware need to be analyzed precisely and the new signatures need to be created and distributed as soon as possible in order to control the propagation.

We propose a new approach to detect and identify malware by automatically generating invariants directly from the specified malware code that we call *malware-invariants*, and use them as *semantic aware signatures.* Malware invariants are propertiesthat hold true on any possible behaviors, execution of considered malware. In order to do so, one needs to adapt *formal methods* currently used to verify and prove systems correctness. Importantly, malware invariant remains unchanged for most of obfuscated version of the code. In other words, most of the new versions of the malicious piece of code, obtained by obfuscation techniques, would still share the same malware invariants. Thus, using such platforms, one would have only a few semantic aware signatures for each family of viruses.

Using such static analysis platforms and automatically generated invariants we show how to construct a suitable architecture for *Host-based intrusion detection systems.* Host-based intrusion detection systems that monitor an application execution report any deviation from its statically built model. They have seen a tremendous progress in recent years. We could capture the main difficulties met by recent host-based intrusion detection systems by saying that they use models that reflect only the control flow structure of the programs. The lack of precision of these models explain why they are subject to *mimicry attacks* [129] and *non-control-data attacks* [23].

We propose a new model addressing various deficiencies of the state-of-the-art of other recent intrusion detection systems. Our model captures the control flow structure and the data flow characteristics of a program. It is automatically generated and has a very low monitoring overhead. Further, using our methods, one can mathematically prove that any intrusion once the violation of an application invariant is observed during the execution of the application. Being rigorous, they allow for no false alarms. Our model does not only prevent mimicry and non-data-control flow attacks, but can also be used to detect logic bugs and other vulnerabilities in the application.

As the main contribution (see Section 1.2.4) , we prove that any static analysis based malware or intrusion detection system will be strongly re-enforced by the presence of pre-computed invariants, and will also be weakened by their absence. In Section 6.2 we discuss a theoretical basis that supports the design of malwares analysis platforms using formal methods. In Section 6.3 we present guarded monitor generation for intrusion detection and vulnerability auditing.

6.2 Malware Invariant Generation

6.2.1 Malware and Virus Characterisation

A *malware* is a program that has malicious intent. Examples of such programs include viruses, Trojans horses, and worms. These malicious intent display the following behavior, by:

1. following *infection strategies*,
2. executing a set of malicious actions, *payloads*,
3. evaluating some boolean control conditions, called *triggers*, to determine when a *payload* will be activated.

A classification of malware with respect to their effects and propagation methods is proposed in G. McGraw and G. Morrisett [90]. Also, L. M. Adelman [2] and F.B. Cohen [30] show that the research security community will deeply need a mathematical

Type	Active Propa.	Pop. Evolution	Context-free
Virus	yes	> 0	no
Worm	yes	> 0	yes
Spyware	no	0	yes
Adware	no	0	yes
Trojan	no	0	no
Back Door	no	0	partially
Rabbit	yes	0	yes
Logic Bomb	no	0	partially

Tabela 6.1: Characterisation for malware types.

formalism that could serve as a scientific basis for their classification. We can distinguish three properties that characterise a class of malware:

- *Active propagation*: A malware can propagate passively or actively using self-instance replication or self-modification.
- *Population evolution*: A malware can be characterized by the evolution of the number of malware instances.
- *Context dependence*: in order to perform its malicious intent, a malware can depend on external context, *e.g.* it could require other executable code or a pre-compilation step.

We give the classification of some types of malwares using these three properties in Table 6.1. Other hybrid types and network-based denial-of-service types like *botnets* and *zombie net-works*, ... could be also considered by combining and extending these properties. In Subsection 6.2 we will encounter more specific characterisation properties, like obfuscation and encryptions techniques, that we shall consider in our approaches for automated malware analysis.

6.2.2 Identifying Malware Concealment Behaviours

Current malware detectors are "*signature-based*" and equipped with a data base. These malware detectors are based on sound methods, that is, if the executable matches byte-signatures, then they guarantee the presence of the malicious behavior. They are equipped with a database of regular expressions that specify byte or instructions sequences that are considered malicious. But the main problem resides in the fact that these methods have a low degree of completeness.

Malware writers can then evade detection by current syntactic signatures and pattern matching approach by injecting code into malwares in such a way that it preserves malicious behavior and makes the previous signature irrelevant, because the regular expressions would not match the modified binaries. These techniques are called *obfuscation* [97]. Hackers frequently use obfuscation techniques to morph malware, as they are easy to write and very challenging to detect. Malware detectors, commercial anti-virus and scanners are susceptible to these techniques.

As common obfuscation techniques, one could cite *polymorphism* and *metamorphism*. In these methods, a virus encrypts, or obfuscates, its malicious payload and decrypts, or deobfuscates, it during execution. By doing so, a virus can succeed in morphing itself. Here the words "encryption"and "decryption"do not refer to cryptography, they are better thought of as obfuscation and deobfuscation strategies.

Once a new type of malware appears, the number of derivative malwares generated by obfuscation increases exponentially. Malware writers can easily generate new undetected viruses. As a consequence, the anti-virus code has to update its signature database very frequently in order to be able to catch the new virus the next time it appears.

6.2.3 Obfuscation Strategies for Infection Mechanisms, Triggers and Payloads

Malware can be obfuscated, i.e. its three body parts, namely Infection Mechanisms, Triggers and Payloads, can be first set in an encrypted form to avoid detection.

But it can not be entirely encrypted to be executable. It needs a *decryptor loop*, which deobfuscates its body parts in a specific writable memory location. Also, the deobfuscation mechanism can use a random key, which vary at any iteration, or it could use encryption library, which contains cryptographic algorithms. The encrypted part would be very difficult to detect, and that is why anti-virus, and polymorphic malware detectors concentrate on detecting the decryptor loop. The unchanging part of an encrypted virus, which can randomly modify its key for each instance, is the decryptor loop.

To avoid detection, a *polymorphic* virus modify their decryptor loop at each instance using several automatic transformation, since a virus could easily generate billion of loop versions [49]. And *metamorphic* viruses change their body parts using a variety of obfuscation techniques when they replicate.

To modify the loop, or to obfuscate a body part, a malware uses a *mutation engine* [97] which can rewrite the loop with other semantically equivalent sequences of instructions [31] and rename register or memory locations. This transforms the original sequence of instructions into an equivalent one. In general this is done by using unconditional jumps, inlining and outlining the body of function codes, using new function calls, inserting junk code, using threaded versions, ... and a host of similar trirs.

Example 27. *Let's consider the following piece of pseudo-code:*

```
call Function_0
...
Function_0:
r5=12
r1=12
r6=r3+r2
r2=34
r4=r5+r6
r3=r1+r2
return
```

It could be rewritten in the form:

```
call Function_1
call Function_2
...
Function_1:
rl = 12
r2 = r3+r2
r4 = rl+r2
return
Function_2:
rl = 12
r2 = 34
rS = rl+r2
return
```

without changing its semantic behavior.

But these decryptor loops and the derived ones still share some common invariants, and, these invariants are more difficult to morph in an automated fashion.

6.2.4 Malware Invariant as Semantic Aware Signature

To be able to reason directly from unknown vulnerable binary code, one needs an intermediate representation [58, 19] for the binary code.

Example 28. *Intermediate representation.*

```
...              //...
dec ecx          //ecx <-- ecx -1
jnz 004010 B7 //If (! ecx =0 ) goto 004010 B7
mov ecx , eax //ecx <-- eax
shl eax , 8    //eax <-- eax < < 8
...              //...
```

The rightmost column shows the semantics of each x86 Executable instructions expressed in a C-like notation [58, 19].

In [26, 25, 40], malware-detection algorithms try to incorporate instruction semantics in order to detect malicious behavior. Semantic properties are more difficult to morph in an automated fashion than syntactic properties. But the main problem of these approaches is that they relay on too coarse abstracted semantic information e.g. *def-use* information. Instead of dealing with regular expressions, they try to match a control flow graph, enriched with def-use informations to the vulnerable binary code. Those methods eliminate a few simple techniques of obfuscation as it is very simple to obfuscate def-use information by adding any junk code or by reordering operations that would either redefine or use the variables present in the def-use properties.

Our approach to the problem of malware detection generates quasi-static invariants directly from the specified malware code, and use it as *semantic-aware* signatures that we call *malware-invariants*. Now consider a suspicious code. We could check if there is one assertion in the malware-invariant data base that holds in one of the reachable program states. In order to do so, we can use our formal methods for invariant generation and assertion checking. This will complicate, make it much more for a hacker to evade the detection using common obfuscation techniques. Thus, for each family of viruses we would have few semantic aware signatures. We propose to use, combine and compose many static and dynamic tools in order to automatically generates invariants which are semantic-aware signatures of malwares, i.e. they are malware-invariant.

6.2.5 Automatic Generation of Malware Invariants

Static Analysis for Detection and Identification

We say that an analysis is *static* when it does not run the application. Anti-virus use a data base of signatures and a *scanning* algorithms to look efficiently for several patterns at a time. Each of these patterns represents several different signatures. As we saw in the previous sections, present malware writers can evade such pattern-matching techniques.

Malware invariants could be computed directly using our invariant generations methods, together with any other invariants computed using different techniques, such as based on a complementary logic. Several tools could be made available though a *communications framework*. We provide a theoretical basis for the construction of static analysis platforms that can form a suitable architecture for the extraction and identification of possible malicious behaviours. We propose to consider several invariant generations techniques where the automatically computed invariants would be express in specific logic. Considering the methods described in Chapter 3, one can handle logic with non-linear arithmetic and inequalities. Using them one can generate non-linear invariants in several guises like assertions, formulae over inter-relationship values of registers, memory locations, variables, system call attributes, and similar others.

Example 29. *Consider the following piece of code expressed in an intermediate representation after a transformation process.*

```
integer R_1, R_2, eax, ebx, ecx, OxN;
...
where (eax=R_1 && ebx=ecx=R_2 && OxN=0)
 ...
while (eax != ebx){
  while (eax > ebx){
    eax = eax - ebx;
    y_4 = OxN + ecx;
  }
  while (ebx > eax){
    ebx = ebx - eax;
    ecx = ecx + 0xN;
  }
}
 ...
```

We can, for instance generate the following invariant

$$eax * ecx + ebx * OxN - R\_1 * R\_2 = 0.$$

□

In order to handle systems and function calls effectively, we adapted the techniques described in A. Tiwari et al. [62, 60] which provide invariants over a logic with uninterpreted function and inequality. This theory uses several level of Abstract Interpretation [36] and is based on Unification Theory [6]. Moreover the implementation also requires a modifications to adapt it of the CIL tool [99].

Example 30. *Consider the following piece of code with uninterpreted* system calls. *Note that it could be any other systems calls with a similar signature.*

```
  void
  F_D(int fd_1,int fd_2){
  ...
  int t1,t2,t3,t4,t5,a;

  if (fd_1==fd_2) {
      t1=dup(fd_1-fd_2);
      t2=dup(dup(0));
      }
  else {
      t2=dup(t1);
      }
  t3=dup(t1);
  a=fd_2-1;
  if (fd_1==a){
      t4=dup2(fd_1,2*fd_1+1);
      t5=dup2(fd_1,dup2(fd_1,2*fd_2-1));
      }
  else {
      t5 = dup2(fd_1, t4);
       }
  }
```

As an example, we were able to generate the following invariant

```
Invariant Generation [Logic: Uninterpreted Function]
t3 = t2
fd_2 + -1.000000*a = 1.000000
t2 = dup(t1)
t5 = dup2(fd_1, t4)
```

As we can see it yields inter-relationships between system calls returns values and attributes. □

We could as well, consider any other logic which has an associated invariant generation techniques and tools. The main contribution here is that most of all obfuscation techniques presented in section 6.2.3 will not change the computed invariants.

Quasi-Static Malware Analysis

In contrast, we propose new concepts and a theoretical framework that can be used to compute exact invariants, i.e. inductive provable invariant, using hypotheses, i.e. likely invariants, that are true properties on all observed execution traces, in a certain training period. We could then generate likely invariants which are properties that hold at any program point in the observed execution trace, using test methods. We call this new view of program analysis "*Quasi*-Static Analysis". Figure 6.1 illustrates the architectural structure of this new theoretical framework. Then we could turn likely invariants into invariants using verification methods, like assertion checkers [11, 132].

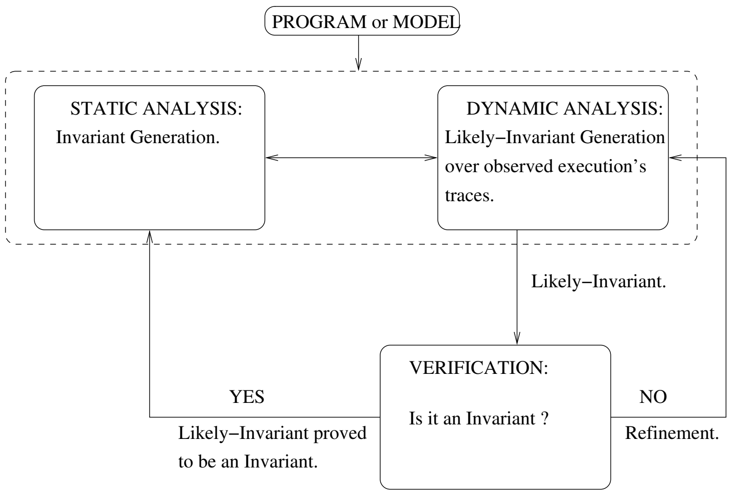


Figura 6.1: Quasi-static analysis framework

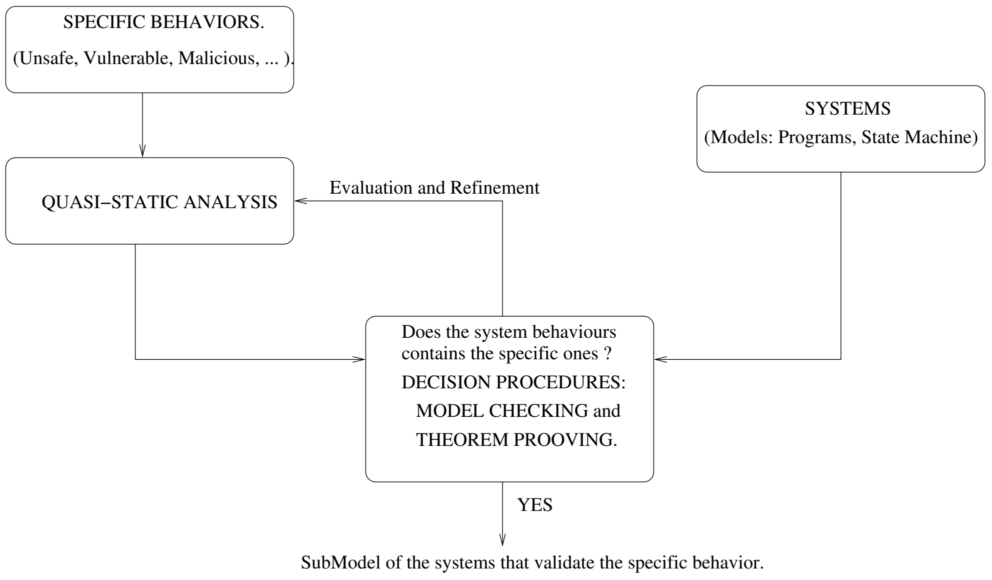

SPECIFIC BEHAVIORS.
(Unsafe, Vulnerable, Malicious, ...).
SYSTEMS
(Models: Programs, State Machine)
QUASI–STATIC ANALYSIS
Evaluation and Refinement
Does the system behaviours contains the specific ones ?
DECISION PROCEDURES:
MODEL CHECKING and
THEOREM PROOVING.
YES
SubModel of the systems that validate the specific behavior.

Figura 6.2: New formal methods to verify if the considered systems contains common behavior with another specified systems

Some of the likely-invariants computed by a Dynamic Analysis are real invariants. They hold in all possible executions of the program. Then, using theorem provers or assertions checkers, for instance one could check if the proposed properties during the Dynamic analysis are real invariants. These computed malware-(likely)invariants are then treated as signatures. One can, then, use pushdown model checking techniques and theorems proving methods, combined with program verification tools and methods, to detect the presence of malicious behavior described by our malware-invariants. If the malware-invariants describe reachable states in the verification process, then we can guarantee that the software displays the suspicious behavior. This point is illustrated in Figure 6.2

One can first generate a malware invariant ϕ_{sign} as described just above. Then one can use similar invariant generation methods, applied over the vulnerable executable code being inspected, and generate an invariant ψ_{Exe}. Then one could check if $\psi_{Exe} \Rightarrow \phi_{sign}$ using a theorem prover or other decision procedure. Alternatively, one can generate a push down system modeling the malicious behavior by combining control flow and data flow analysis. Those abstract state graph, that we call *malware PDS*, captures both the control and the data flow properties of the malicious behaviours. We can use and combine malware invariant and malware PDS to form *formal signature*. This point is illustrated in Figure 6.3.

One can, then, build formal signature data base. On the other hand we could extract invariant from the code being inspected with similar mentioned methods and check, using

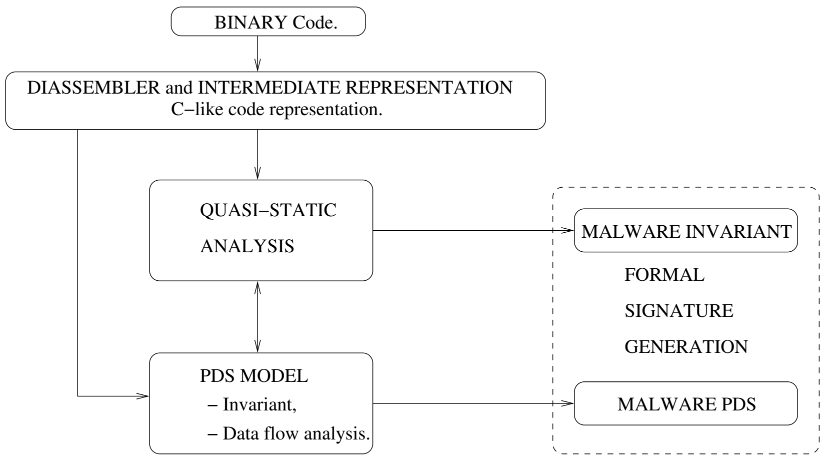


Figura 6.3: Formal Signature: Malware Invariant and Malware PDS Generation.

decision procedures, if it implies one of the malware invariant. Or, we can check if the code behavior allow a possible run in one of the malware PDS. The malware detector method described in this Section is sound with respect to the signature that is being considered as the representation of malicious behavior. This point is illustrated in Figure 6.4.

In Section 6.3, we build a working theoretical platform, and its associated Host-based intrusion detection systems, following similar a proof of concept. A semantic aware malware detector can then be built upon the intrusion detector system.

6.3 Guarded Monitors For Host-based Intrusion Detection Systems

Host-based intrusion detection systems monitor an application execution and report any deviation from its statically built model [128, 47, 57]. It has been shown [51] that finding anomalies in the *stream of system calls* issued by the execution of user applications is an effective host-based intrusion detection capability. In D. Wagned and D. Dean [128], the idea of using static analysis to design a model of application leading to a host-based intrusion detection system was mentioned for the first time. They first recognized the fact that such systems would have the following three advantages:

- a high degree of automation,
- protection against a broad class of attacks, and

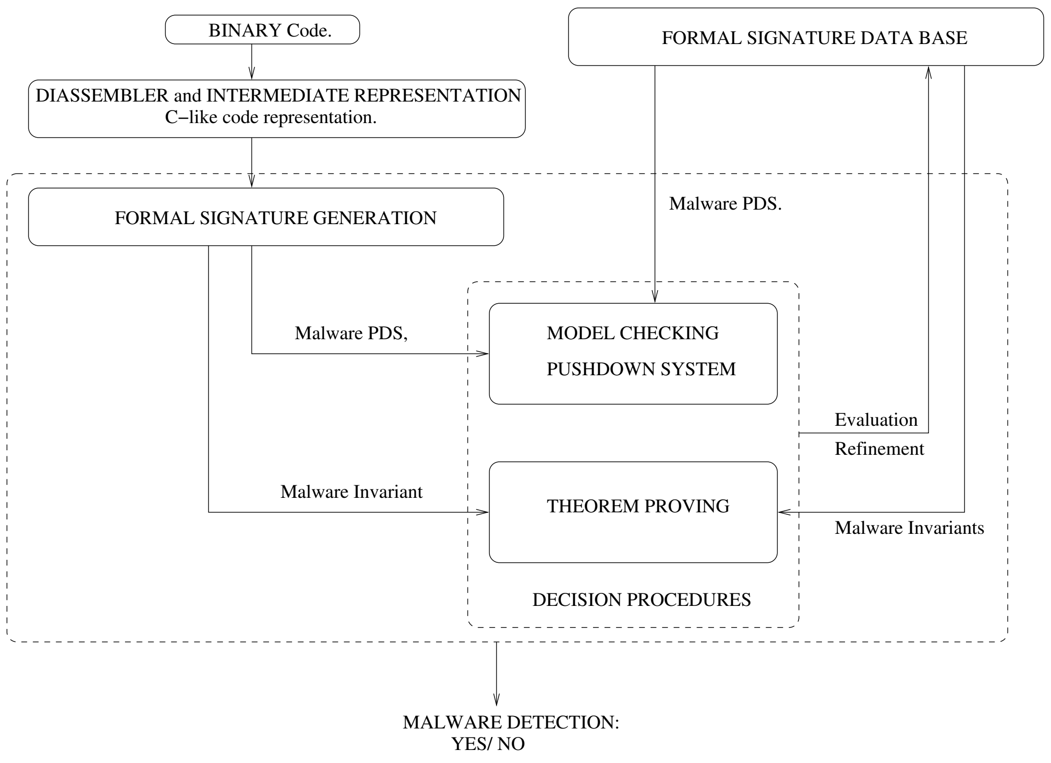


Figura 6.4: Formal Malware Intrusion Detector System

- the elimination of false alarms.

Since then, various models capturing the sequences of legitimate system calls have been proposed [128, 57, 47, 55]. The *degree of precision* of the model is the main issue that current intrusion detection systems approaches struggle with. The more precise the model is, the less attacks are possible.

The weakness of these systems is that they often rely on overly abstracted models that reflect only the control flow structure of programs and, therefore, are subject to the so-called *mimicry attacks* [129, 75] that produce a sequence of system calls that conform to the model but allow the execution of malicious code. In other words, the weakness of current host-based intrusion detection systems can be traces to the lack of any significant data flow analysis.

We propose to use automatically generate invariants to guard system calls. By combining control flow and data flow analysis, our models are not only able to detect mimicry attacks, but they also tackle the ever increasingly threatening *non-control-data flow attacks* [23]. We note that more data flow information about the program is necessary in order to prevent such attacks.

These attacks are all detectable by our model because it captures in a very precise manner the semantics of the application being protected. During the execution of the application, the associated model simulation will detect any *mimicry* and *non-control-data* attacks, as they inevitably violate an invariant specified in the model. Our model makes sure that system calls not only occur in the order specified by the model, *i.e.* they maintain *control flow integrity*, but also guarantee that each system call is preceded by a check that some program invariant generated by static analysis holds, *i.e.* they assure *data flow integrity.* Such invariants capture crucial properties about system call arguments, system call return values, input variables, and about the values of branch predicates at all control locations of the program. Some very weak versions of these properties are generated manually in J. Giffin et al. [55], or learned from runs of the application in an attack free environment in R. Sekar et al [12]. Thus, in these cases, they form an under-approximation of the application behavior, which leads to an intrusion detection approach that generates many false alarms.

We use invariant generation and assertion checking techniques to build our model which can be described as a control flow graph in which system call locations are annotated with a set of predicates and logic assertions that have to remain true at those system call locations. It is an *abstract state graph* [59] that captures both the control and the data flow properties of a program. The model is statically build using a combination of predicate abstraction, invariant generation techniques and inter-procedural invariant propagation.

We use these statically built model to *monitor* an application execution and report any deviation from it i.e. we report any behaviors that are not possible to simulate by the model. These monitors are Visibly Pushdown Automaton [5], in which, invariants are checked against the image of the process before any system call. As we discuss in R. Rebiha and H. Saidi [108], our method is much more secure, while the overhead is reduced drastically compared to Dyck models [56], since the latter needs to add null system calls at any function call site. Our monitor is automatically generated by combining control flow and data flow analysis using state-of-the-art tools for automatic generation and propagation of invariants.

Further, our methods do not yield false alarm because if an invariant is violated we have the proof of an abnormal behavior issued by an intrusion. In many cases, our model is precise enough so that it is possible to automatically check the application for the presence of application logic bugs and vulnerabilities. Such feature is absent from any other intrusion detection approach.

6.3.1 Systems Calls Guarded by Invariants

Invariants that are true at control locations where system calls are invoked become guards that should evaluate true at run-time, before allowing for a system call. This ensures both control flow and data flow integrity.

First we propose an intra-procedural analysis to define model a procedure. This model is the abstract state graph of the procedure. It is a much more precise version of the control flow graph of the procedure.

Definition 28. ***ASG: Abstract State Graph***. *Let P be a procedure, and let $p_1, \dots, p_k$ be a set of arbitrary predicates over P's program variables. An ASG of P using the predicates $p_1, \dots, p_k$ is the intraprocedural control flow graph*

$$G = \langle V, V_a, Enter_p, Exit_P, \delta_a \rangle$$

associated with the program source code of P where:

- *V is the finite set of program locations,*
- *$Enter_P \in V$ is the entry point of P*
- *$Exit_P \subseteq V$ is the finite set of P's exit points.*
- *V_a is the finite set of* abstract states. *Each such abstract state is a pair (v, e) where v is a program control location and e a valuation of the predicates $p_1, \dots, p_k$. For instance for $k = 3$, a node $pc1, (0, 1, 1)$ indicates that at location $pc1$, the predicate $P_1 \wedge \neg P_2 \wedge \neg P_3$ is true.*
- *$\delta_a \subseteq V \times V$ is a transition relation.* □

Example 31. *Consider the following program:*

```
x=0;
While (x <= 10){
   x=x+1;
}
```

Consider the predicate $\mathscr{B} \equiv (x \leq 10)$*, its abstract state graph is given below:*

$$\begin{tikzcd}
\langle Enter_P, \top \rangle \arrow[r, "\tau_1\ :\ x = 0;"] & \langle l_2, \mathscr{B} \rangle \arrow[loop above, "\tau_2\ :\ x = x + 1;"] \arrow[r, "\tau_3\ :\ x = x + 1;"] & \langle l_4, \neg\mathscr{B} \rangle \arrow[r, "\tau_4"] & \langle Ex_P, \neg\mathscr{B} \rangle
\end{tikzcd}$$

with $V = \{Enter_P, l_2, l_4, Ex_P\}$*,* $Exit_P = \{Ex_P\}$*, and* $\mathcal{T} = \{\tau_1, \tau_2, \tau_3, \tau_4\}$. □

The ASG of a procedure is turned into a Guarded Control Flow Graph by guarding transitions originating from an abstract state (v, e) by the Boolean combination of the predicates $p_1, \ldots, p_k$ given by the valuation e. This Boolean combination is an invariant of the program at location v.

Definition 29. ***(GCFG: Guarded Control Flow Graph)*** *A GCFG of a procedure* P *is the* intraprocedural *control flow graph* $G_P = \langle V, v_0, Exit_P, \mathcal{G}, \delta_{lP} \rangle$ *associated with the program source code of* P *where:*

- V *is the finite set of program locations.*
- $Enter_P \in V$ *is the entry point of* P*.*
- $Exit_P \subseteq V$ *is the finite set of* P*'s exit points.*
- $\delta_P(V \times \mathcal{G}) \subseteq 2^{\mathcal{I}} \times V$ *is a transition function where a block of instructions in* $\mathcal{I}$ *is guarded by an invariant in the set* $\mathcal{G}$ *.* □

Since we monitor system calls, our model abstracts away from the GCFG any transitions and instruction blocks that do not refer to a system call or to function call.

The result is a guarded call graph.

Definition 30. ***(GCG: Guarded Call Graph)*** *Let P be a procedure and let its GCFG be*

$$G_P = \langle V, Enter_P, Exit_P, \mathcal{G}, \delta_{lP} \rangle.$$

A guarded call graph (GCG) of P is a tuple

$$G = \langle \mathcal{S}, Enter_P, Exit_P, \delta_P, \lambda, FuncC, SysC, Pred_{site} \rangle$$

where :

- *$\mathcal{S}$ is a finite set of states.*
- *$FuncC$ is the finite set of* function call *that appear in P.*
- *$SysC$ is the finite set of* system call *that appear in P.*
- *$Pred_{site}$ is a finite set of predicates associated to function call sites.*
- *$\lambda : \mathcal{S} \to 2^V$ labels the state with a set of consecutive program points that identify a basic block, i.e. straight-line pieces of code without any jumps nor system or function calls.*
- *Finally,*

  $$\delta_P \subseteq (\mathcal{S} \times \mathcal{G} \times FuncC \times \mathcal{S}) \cup (\mathcal{S} \times \mathcal{G} \times SysC \times \mathcal{S})$$

  is a transition relation. □

Intuitively, these action guarded transitions are labeled by a system ($SysC$) or a function ($FuncC$) call that occur at the last program point labeled in the current state (*basic block*). They should satisfy:

- If there is a function call $F \in FuncC$ at a program point $v \in V$ then $\exists s, \exists s' \in \mathcal{S}$ and $\exists g \in \mathcal{G}$ s.t. $v \in \lambda(s)$, and $\delta_P(s, g) = (f, s')$. Also the predicate $P(F, s)$ is in $Pred_{site}$.
- If there is a system call $C \in SysC$ at a program point $v \in V$ then $\exists s, \exists s' \in \mathcal{S}$ and $\exists g \in \mathcal{G}$ s.t. $v \in \lambda(s)$, and $\delta_P(s, g) = (c, a, s')$.

Since we monitor only system calls, it is necessary to replace function calls by a jump to the model of the called function, while retaining information about the control location to which the application should return after the call to the function is completed.

Definition 31. ***(GSCG: Guarded System Call Graph)*** *Let P be a procedure and let guarded call graph (GCG) be*

$$G = \langle \mathcal{S}, Enter_P, Exit_P, \delta_{SC}, \lambda, FuncC, SysC, Pred_{site} \rangle.$$

Then G is said to be a Guarded System Call Graph *if $FuncC = \emptyset$. In other words, G does not generates function calls. Furthermore, the transition relation δ_{SC} is defined as*

$$\delta_{SC} \subseteq (\mathcal{S} \times \mathcal{G} \times SysC \times (A \cup \{epsilon\}) \times \mathcal{S})$$

where the finite set of actions A is given by the grammar

$$a \leftarrow P := \texttt{True} | P := \texttt{False} | \epsilon,$$

where P is a predicate in $Pred_{site}$. □

Intuitively, by eliminating function calls, the transition relation only refers to system calls. However, for the purpose of maintain control flow integrity, it is crucial to maintaining information related to function call sites and return sites. This is achieved by augmenting the transition relation with assignments of the form $P(s, F) :=$ `True`, or $P(s, F) := 1$, that indicate that a function F has been called at control point s, or,in other words, the call was done at a location v such that $\lambda(v) = s$. An assignment of the form $P(s, F) :=$ `False`, or $P(s, F) := 0$, where $P(s, F)$ is a predicate in $Pred_{site}$, indicates a return from a call to the function F.

In what follows, we propose to extend the previous model with a stack, which will allow us to deal with recursive functions.

Definition 32. ***E_GSCG: Extended GSCG*** *Let P be a procedure and let its guarded call graph (GSCG) be*

$$G = \langle \mathcal{S}, Enter_P, Exit_P, \delta_P, \lambda, FuncC, SysC, Pred_{site} \rangle.$$

The E_GSCG E_G extends G with a finite stack alphabet

$$\Gamma \subseteq \{Pred_{site} \times \{0, 1\}\},$$

where $\gamma_0 \in \Gamma^*$ the initial stack configuration. We define the transition system δ_E with the set of actions A_E as follow:

- *A_E is the finite set of actions given by the following grammar:*

  $$a \leftarrow \mathsf{Push}(\gamma) | \mathsf{Pop}(\gamma) | B := 0 | B := 1 | \epsilon,$$

  where $B \in Pred_{site}$, $\gamma \in \Gamma$ and $\mathsf{Push}(\gamma)$, $\mathsf{Pop}(\gamma)$ are the two common operations that pop and push γ into the stack.

- *Finally,*

$$\delta_E \subseteq \cup(\mathcal{S} \times \Gamma \times \mathcal{G} \times SysC \times (A_E \cup \{\epsilon\}) \times \mathcal{S} \times \Gamma).$$

$\square$

The semantics of the transition relation δ_E identifies valid sequences of system calls during a program execution. Its semantics is given by the rules described below.

Definition 33. ***Run of an E_GSCG****.*
For sequence of system calls, i.e. a word

$$w = a_1, ..., a_k \in Syc_{call}$$

a run of G_P on w is a sequence

$$\rho = (s_0 \rho_0), \ldots, (s_k, \rho_k),$$

where for every $s_i \in \mathcal{S}, \rho_i \in \Gamma^*$, $\rho_0 = \epsilon$ and for every $1 \leq i \leq k$ the following rules apply:

- Making a system call*:*
  If $(s_i, \epsilon, g, a_i, \epsilon, s_{i+1}, \epsilon) \in \delta$ then the system call a_i is allowed by our model when transitioning from s_i to s_{i+1} if the guard $g \in \mathcal{G}$ evaluates to True *at s_i and there is no stack activity ($\rho_{i+1} = \rho_i$).*

- Call to a non-recursive function*:*
  If $(s_i, \epsilon, g, a_i, (P(s_i, F) := \texttt{True}), s_{i+1}, \epsilon) \in \delta_E$ the system call a_i is allowed by our model when transitioning from s_i to s_{i+1} if the guard $g \in \mathcal{G}$ is evaluate to True *at s_i. The action of the transition sets the predicate $P(s_i, F) \in Pred_{site}$ to* True*, meaning that a call to a function F has been done at s_i and a_i is the first system call induced by F for the current context. Also, there is no stack activity.*

- Return from a non-recursive function*:*
  If $(s_i, \epsilon, (P(s_i, F) == \texttt{True}) \wedge g), a_i, (P(s_{i+1}, F) := \texttt{True}), s_{i+1}, \epsilon) \in \delta_E$ then the system call a_i is allowed *by our model when transitioning from s_i to s_{i+1} if the guard $(P(s_i, F) == \texttt{True}) \wedge g$ evaluates to* True *at s_i. The action of the transition assigns the value of the predicate $P(s_{i+1}, F) \in Pred_{site}$ to* False*, meaning that s_{i+1} is the return address of the non-recursive function F. In the case where there are many call sites to F, the guard checks also if it is the correct return address $(P(s_i, F) == \texttt{True})$. Also, there is no stack activity.*

- Call to a recursive function*:*
  If $(s_i, \epsilon, g, a_i, \mathsf{Push}(\gamma), s_{i+1}, \gamma) \in \delta_E$ then the system call a_i is allowed by our model

when transitioning from s_i to s_{i+1} if the guard g evaluates to True *at s_i. In that case, γ is* pushed *onto the predicate stacks, meaning that $\rho_{i+1} = \gamma \cdot \rho_i$, with s_i interpreting the function call site mentioned in γ, and where "$\cdot$" is the operator of concatenation.*

- Return from a recursive function*:*
  If $(s_i, \gamma, (P(s_{i+1}, F) == \textit{True}) \wedge g, a_i, \textit{Pop}(\gamma), s_{i+1}, \epsilon) \in \delta_E$ *and* $\gamma = (P(s_{i+1}, F), 1)$. *The system call is allowed if g and the test at the* ***Top*** *of the predicate stack* $(P(s_{i+1}, F) == \textit{True})$ *are* True. *It means that the program returns from a function call and pops γ from the stack under the condition that γ is at the top of the stack.*

$\square$

Existing models that use a Pushdown System to monitor sequences of system calls are *visibly pushdown automaton* [5]. Intuitively, the stack alphabet is a finite set of return addresses specifying to where a function call returns. The visibly pushdown automaton pushes onto the stack an element when it reads a function call, it pops the stack only when returning, and it does not use the stack when it reads allowed system calls. Using a visibly pushdown automaton requires code instrumentation to trigger the push and pop operations based on the observed inputs to the automaton.

Our model is more efficient at monitoring time because we already encoded these operation using the predicates in $Pred_{site}$. It is a generalisation of *visibly pushdown automaton* with a *stack of predicates* controlling the return of a function called at the right address. It achieves stack determinism and can be simulated by a Dyck model. But the latter generates more overhead during monitoring, by adding extra system calls at each function call and return sites. It can not handle recursive functions or cycles in function calls. We show that recursive functions can be handled without generating extra system calls. We, thus, succeeded in building a model which does not need to monitor the program counter, unlike in D. Dean et al. [128], nor does it introduce extra null system calls unlike in J. Giffin et al. [47, 55].

6.3.2 Construction of our model

Our invariants are generated by using a combination of several methods and tools. Assertions are generated from observed runs during a training period in an attack-free environment and using Daikon [45]. Daikon generates *likely invariants* from execution traces. That is, assertions or relationships between program variables that are true at some control location during the observed runs, but may or may not be true for all possible runs of the program. We use the BLAST assertion checker for C programs [11, 65] to try to prove or disprove the fact that the likely invariants hold at the locations indicated by Daikon.

We also automatically generate linear invariants expressing linear relationships among integer program variables that hold at all control locations of a program using the Inv-Gen tool described in A. Tiwari et al. [61]. In particular, we generate invariants that hold at entry and exit points of every procedure in the application. We use the linear invariants as inputs to BLAST, as additional information about the program, so that it maximizes BLAST's chances of proving or disproving likely invariants. BLAST proves assertions using predicate abstraction [59]. That is in order to obtain a finite representation of the program, it partitions the reachable states of the program using a set of predicates appearing in the assertion to prove, the conditional branches and assignments as well as internally generated predicates [68]. BLAST's assertion checking process amounts to evaluating the set of predicates in every control location, effectively propagating the predicates throughout the control graph of the program. The propagation works as follows: if an assignment instruction does not modify the variables in a predicate, then the predicate holds after the assignment. If however the program variables in the predicate are modified by the instruction, it is necessary to either prove that the post condition of the assignment implies that the predicate still holds, or its negation holds. If neither implication is proved, then the successor control node is split into a node where the predicate holds and a node where the negation of the predicate holds, introducing therefore some non-determinism in the control flow graph. The result of our invariant generation and assertion checking process is a control flow graph where each program control point is annotated with a set of predicates that are true at that location. That is, an *abstract state graph* that captures both the control and the data flow properties of a program.

To review, in order to build our model:

- We adapted the Invariant Generation methods of A. Tiwari [60] to handle inter-procedural codes with system calls.

- We run the application using Daikon [45] and generate a set of *likely invariants*: assertions generated from observed runs during a training period in an attack-free environment. These are, assertions or relationships between program variables that are true at some control location during the observed runs, but may or may not be true in all possible runs of the program.

- We turn every likely invariant generated by Daikon into an assert statement at the line of code indicated by Daikon, and we augment the application code with these assert statements.

- We run the Blast [11] assertion checker and check all assert statements in the augmented application code, making use of the invariants generated by Inv-Gen as aids in the assertions checking process.

- We filter the set of likely invariants into three sets: (i) likely invariants for which Blast can prove that the corresponding assert holds; (ii) likely invariants for which Blast generates execution traces that violate the corresponding asserts, and (iii) likely invariants for which Blast either runs out of memory, crashes, or returns with an inconclusive answer.

- We generate the control flow graph of each function, in such a way that each node is decorated by the set of predicates that hold at that node.

- Finally, we minimize the abstract state graph by applying a minimization algorithm, based on observational equivalence proposed by R. Milner [92] and implemented in J-C. Fernandez [48].

The call graph of each function is turned into a guarded call graph by guarding each system call and function call by the invariant that holds at the control location where the call is invoked. The minimization step does not loose any information about the collected invariants at the control nodes that have been collapsed, since the construction of the abstract state graph propagates every invariant to every node location where it holds and, in particular, at control locations where a function or a system call is invoked. While obtaining the guarded call graph for each function is simple once an abstract state graph is built, combining the guarded call graphs of all functions to build a global call graph can be done in different ways.

In J. Vitek et al. [57] inlining is used whenever a function call is made. The problem in this approach is that the same call graph is duplicated at each call site of a function without further analysis. In our approach, invariants that are true at the function call site are propagated inside the function to obtain a more refined control structure. That is, for every function call site, we might inline a different and more precise model of the function. In the case where the propagation of the predicates does not produce a more precise model of the function, we just transfer the control to the function's guarded call graph. Since we need to remember the call site to which the execution should return to after the function call, we add a new predicate that is set to true when a particular call is invoked at a particular control location. We update the specific predicate $P \in Pred_{site}$ at function call sites and return sites. We define first the set of predicates that we use. Then, $Pred_{site}$ is given by the following characteristic function:

$$Pred_{site}(\mathcal{S} \times FuncC) \subset \{\{0\}, \{1\}\}.$$

The predicate $Pred_{site}(s, F)$ is true if there is a function call to F at s.

Adding call site predicates amounts to combining the guarded call graph of all functions and producing a global guarded system call graph when there are no recursive functions, or an extended guarded system call otherwise. This is done by eliminating the function calls and deciding whether to apply inlining or not.

The algorithm, presented in figure 6.5, describes our process for producing the final global model, starting from the guarded call graph (GCG) of the main function.

6.3.3 Guarded monitors precision and properties

Our model allows less attacks than, for instance, a Dyck model [55]. Recall that a Dyck model instrumentation involves inserting distinct pre- and post-calls, denoted *Pre_Call_* and *Post_Call_* at each function call site. This ensures determinism in stack operations as each call site is identified by its pre-call. A Dyck stack records the *Pre_call_* and a path is deemed feasible if each return leads to *Post_Call_* that matches the last pre-call at the top of the Dyck stack.

Theorem 26. ***(Expressivity)*** *Let $\mathcal{P}$ be a program, and GSCG and E_GSCG be, respectively the guarded system call and and the extended system call graph of $\mathcal{P}$. We have:*

- *If $\mathcal{P}$ contains no recursive functions then any GSCG can be simulated by a Dyck Model.*
- *If $\mathcal{P}$ contains recursive functions then any E_GSCG can be simulated by a Dyck Model.* □

Theorem 26 shows that our Model is always more precise than the Dyck Model. Also, the level of stack determinism is reached without the overhead of instrumenting the binary code with null system calls. We also do not need to monitor the Program counter, as it is statically taken into account by the Abstract State Graph. By using updates over predicates in $Pred_{site}$, we eliminate the need to instrument the source or the binary code with the addition of system calls at each system call site. If $\mathcal{P}$ does not contain recursive functions, or cycles in sequence of function calls, then a stack is not mandatory, even if a function is called at different sites.

Following is a series of examples that illustrate the weaknesses of the various models proposed in the literature and how our model captures automatically dependencies between program variables and system calls in the form of powerful invariants that produce a dramatically more precise control flow graphs and limit the attacker's ability to launch mimicry attacks.

Let $T = (s, g, F, s') \in \delta_P$, a Function call to F is made at site s.

- If $FuncC_F = \emptyset$ then F does not make any function call.
  - In that case if $SysC_F \neq \emptyset$, that is F contains at least one system call C, then there exist $t_1, \dots t_m \in \delta_F$ such that
    $$t_k = (Entry_F, g_k, C_k, A_k, s_k)$$
    with $1 \leq k \leq m$ and $m \leq |SysC_F|$. We then connect the two graphs, without duplicating the graph of the called function, as follows:
    * we merge the states s and $Entry_F$,
    * we add the transition
      $$t_k = (s, \phi_{(g,g_k)}, C_k, (Pred_{site}(s, F) := 1), s_k)$$
      to δ_P, where $\phi_{(g,g_k)}$ are guards computed automatically using propagation techniques. The predicate $Pred_{site}(s, F)$ is then set to true.
    * If there exists $r_1, \dots r_n$ in δ_F such that
      $$r_j = (s_j, g_j, C_j, A_j, s'_j) \wedge s'_j \in Exit_F$$
      with $1 \leq j \leq n$ and $n \leq |SysC_F|$, then we add the transition
      $$r_j = (s_j, \phi_{g,g_j} \wedge Pred_{site}(s, F), C, (Pred_{site}(s, F) := 0), s'$$
      to δ_P.
    * We remove T from δ_P and we repeat this process until δ_P contains no more function calls.
    * We apply this process to all functions that call F.
    * We remove all transitions t_k from δ_F with $1 \leq k \leq m$ and we remove all transition r_j from δ_F with $1 \leq k \leq m$.
  - If $SysC_F = \emptyset$.
    * We merge s and s'
    * We update the guards of all outgoing transitions of s' by propagating invariant of F, and we remove T.
- If $FuncC_F \neq \emptyset$ then F makes function calls. Then we apply the same algorithm to functions in $FuncC_F$.

Figura 6.5: Process for producing the final global monitor

Example 32. *Let's first consider the following program without any function calls, described in [47].*

```
1  char *str, *user;
2  ...
3  if (strncmp (user, "admin", 5)){
4     sys_1 ();}
5  else {
6     sys_2 ();}
7  /* user = ''gest''; */
8  strcpy (str, someinput);
9  if (strncmp (user, "admin", 5)){
10    sys_3 ();}
11 else {
12    sys_4 ();}
13 }
```

In line 3 and 9, the predicate strncmp (user,"admin", 5) *determines which branch of the* `if` *statement is executed. Only two system call sequences are possible:* ⟨*sys_1*, *sys_3*⟩ *if the predicate evaluates to* `True` *and* ⟨*sys_2*, *sys_4*⟩ *otherwise.*

Since recent models proposed in the literature like Dyck Model [47, 55] do not track automatically the values of branch predicates, they will allow the following four sequences ⟨*sys_1*, *sys_3*⟩, ⟨*sys_1*, *sys_4*⟩, ⟨*sys_2*, *sys_3*⟩ *and* ⟨*sys_2*, *sys_4*⟩ *as can be noticed on their following representation:*

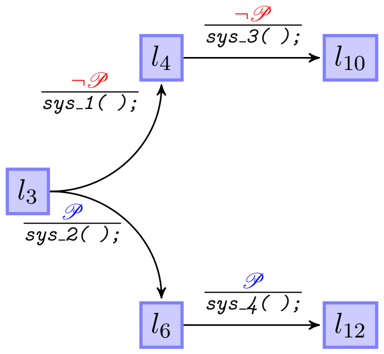
$$\begin{tikzcd}
\boxed{l_3} \arrow[r, bend left, "\text{sys\_1( );}"] \arrow[r, bend right, "\text{sys\_2( );}"'] & \boxed{l_9} \arrow[r, bend left, "\text{sys\_3( );}"] \arrow[r, bend right, "\text{sys\_4( );}"'] & \boxed{l_{13}}
\end{tikzcd}$$

An attack on these models uses a large someinput *in* strcpy *to overflow* str *and change the value "*guest*" of* user *to "*admin*". Then the illegal sequences* ⟨*sys_1*, *sys_4*⟩ *or* ⟨*sys_2*, *sys_3*⟩ *are executed without any detection.*

Below, we illustrate our guarded model, here it is simply the abstract state graph of the program built using the predicate

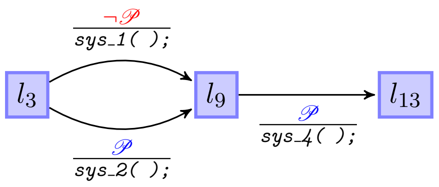
$$\mathscr{P} \equiv (user = admin)$$

appearing in the conditional branches.


l_3
l_4
l_6
l_{10}
l_{12}
$\frac{\neg\mathscr{P}}{sys\_1(\ );}$
$\frac{\mathscr{P}}{sys\_2(\ );}$
$\frac{\neg\mathscr{P}}{sys\_3(\ );}$
$\frac{\mathscr{P}}{sys\_4(\ );}$

Our model will first accept the system call **sys_1** *because the guard* $\mathscr{P} \equiv (user = admin)$ *is true, but will not accept system call* **sys_4**, *because* **sys_4** *is allowed only if the negation of* $\mathscr{P}$ *holds. The illegal sequences will violate the invariants* $\mathscr{P}$ *and* $\neg\mathscr{P}$ *that guard system calls* **sys_3** *and* **sys_4**.

Now, if we uncomment the instruction commented away in line 7, our model is precise enough to allow only the sequences $\langle$**sys_1**, **sys_3**$\rangle$ *and* $\langle$**sys_2**, **sys_3**$\rangle$ *as it is clearly shown in the following graph.*


l_3
l_9
l_{13}
$\frac{\neg\mathscr{P}}{sys\_1(\ );}$
$\frac{\mathscr{P}}{sys\_2(\ );}$
$\frac{\mathscr{P}}{sys\_4(\ );}$

□

Example 33. *Consider the following program:*

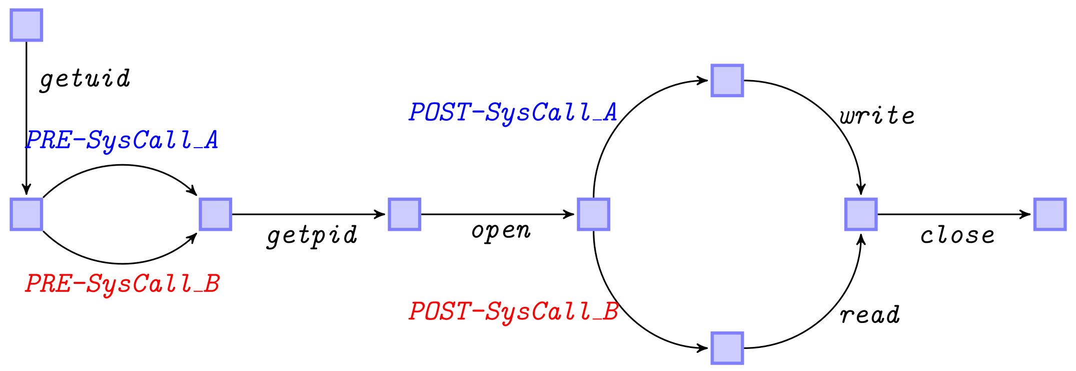
```
char* filename;
pid_t pid[2];
char buf[32];
int main(int argc, char *argv[]) {
  filename = argv[1];
  uid_t uid = getuid();
  int handle, ctrl_uid;
  ctrl_uid = ctrl(uid);
  if (ctrl_uid != 0){
    handle = prepare(1);
    read(handle, buf, 32);
  }
  else {
    handle = prepare(2);
    write(handle, buf, 32);
    close(handle);
  }
int prepare(int index){
  pid [index] = getpid();
  strcpy(buf, filename);
  return open(buf, O_RDWR);
}

int ctrl(int i1) {
  int t1,t2,t3;
  if (i1>0) {
    t1 = 2;
    t2 = 3;
  }
  else {
    t1 = 8;
    t2 = 9;
  }
  t3 = t2-t1;
  return t3;
}
```

In this example the program does a call to a function **`ctrl`** *which does not contain system calls. The behavior of this function affects the sequences of system calls that the program can generate. Existing Models abstract away in their control flow based models* *all function calls that do not contain system calls.*

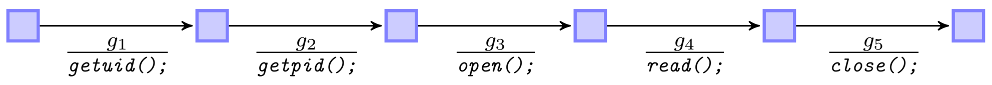
An attack on these models would force the process `pid[2]` *to* `write` *in the input file "filename", whereas our analysis says it can only read it. This attack will be detected by our model and accepted by others. The reason is that we were able to generate an invariant on the possible return value of the function* `ctrl`*, stating that it is always equal to 1 (in the form of the predicate $ctrl(x) = 1$) for all possible input value of x.*

This invariant is propagated to other control locations beyond the control location where `ctrl` *is invoked. This implies that we are able to eliminate the* `''else''` *branch from our model, and therefore detect any attack that tries to execute the* `write` *statement in line* 15.

Here, we compare our monitor to the corresponding Dyck model. First we represent the Dyck model for this program.



Our model obtained for this program is depicted in the figure below:



□

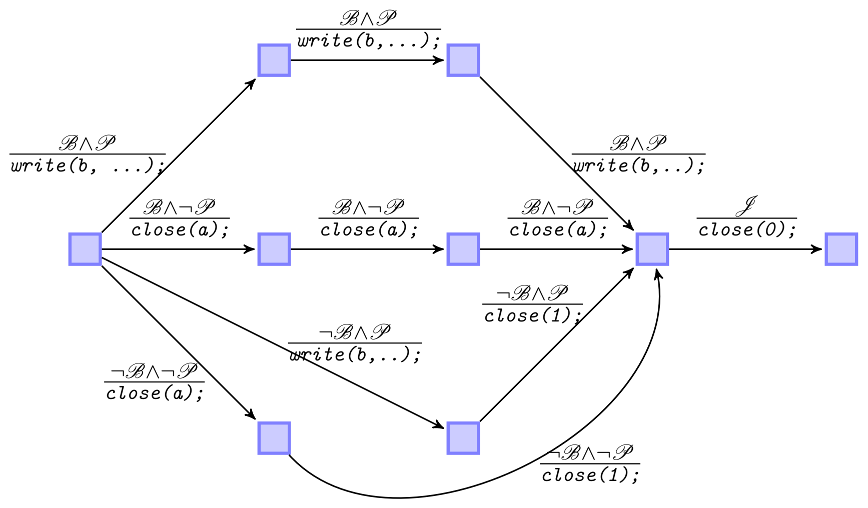
Example 34. *In our model, system calls arguments and return values are naturally captured by automatically generated invariants. Previous approaches [12] propose to learn dependency between an argument variable and a constant during a training period that does not guarantee an exhaustive listing of all possible dependencies. Consider the following example where F and G are two uninterpreted functions.*

```
 1  void File_des(int fd_1, int fd_2) {
 2    int t1,t2,t3,t4,t5,a;
 3    if (fd_1==fd_2) {
 4          t1 = F(fd_1-fd_2);
 5          t2 = F(F(0));
 6    }
 7    else {
 8          t2 = F(t1);
 9    }
10    t3 = F(t1);
11    a = fd_2 -1;
12  if (fd_1==a){
13        t4 = G(fd_1, 2*fd_1+1);
14        t5 = G(fd_1, G(fd_1, 2*fd_2 -1));
15  }
16  else
17    {
18         t5 = G(fd_1, t4);
19         close(1);
20    }
21   close(0);
22  }
```

First, we interpret F and G as being two common system calls dup() *and* dup2() *and we obtain the same program depicted in example 30. So, we obtained automatically the following invariant: $(t3 = t2) \wedge (fd\_2 + -1.000000 * a = 1.000000) \wedge (t2 = \mathsf{dup}(t1)) \wedge (t5 = dup2(fd_1, t4))$. Our invariants are strong enough to express dependencies between the arguments of the system calls* dup() *and* dup2() *and their return values. A possible exploit on this program will start by modifying the value of file descriptors $fd_1, fd_2, t_1, t_2, \cdots, t_5$ in order to grant the attacker access to critical files. As a first consequence, duplications performed by* dup()*,* dup2() *or any system and function calls referring to these file descriptors will be executed with the modified values. This is a mimicry attacks which can not be detected by existing models [47, 55, 12], because their techniques can not for instance see a dependency between the constant argument "0" line 5 in $F(F(0))$, and the arithmetic expression $fd_1 - fd_2$ in line 3 of F and between $fd_1 - fd_2$ and all the return values of system or function call F performed in lines 4 and 5. Since we employ powerful* *invariant generation and propagation tools, our invariants capture all the necessary dependencies and heir effect on the branch predicate $fd_1 == fd_2$ that decides which branch, and therefore, which system call is executed. Now, consider the same program, but instead of interpreting the functions F and G as system calls, we interpret F as ctrl, the function that does not contain system calls given in the program of example 33 (between lines* 24 *and* 36*), and G by the function, which contains system calls, described by the following program:*

```
int G(int a, int b){
  int ret;
  if (b < 15){
      write(b, .., ..);
      ret = b;
      }
  else {
      close(b, .., ..);
      ret = a;
      }
  return ret;
}
```

The figure below describes our generated model using invariant generation and propagation. We consider the predicates $\mathscr{B} \equiv (fd\_1 = fd\_2 - 1)$ and $\mathscr{P} \equiv (b < 15)$.


$\frac{\mathscr{B}\wedge\mathscr{P}}{write(b,...);}$
$\frac{\mathscr{B}\wedge\mathscr{P}}{write(b, ...);}$
$\frac{\mathscr{B}\wedge\mathscr{P}}{write(b,..);}$
$\frac{\mathscr{B}\wedge\neg\mathscr{P}}{close(a);}$
$\frac{\mathscr{B}\wedge\neg\mathscr{P}}{close(a);}$
$\frac{\mathscr{B}\wedge\neg\mathscr{P}}{close(a);}$
$\frac{\mathscr{J}}{close(0);}$
$\frac{\neg\mathscr{B}\wedge\mathscr{P}}{close(1);}$
$\frac{\neg\mathscr{B}\wedge\mathscr{P}}{write(b,..);}$
$\frac{\neg\mathscr{B}\wedge\neg\mathscr{P}}{close(a);}$
$\frac{\neg\mathscr{B}\wedge\neg\mathscr{P}}{close(1);}$

Any mimicry attacks that manipulates these files descriptors will not by detected by the other models. In particular, a Dyck mode [47, 55] will accept many sequences that could be executed non-deterministically whereas our model only allows in a deterministic way the four sequence of system calls described in the graph above. □

In Figure 6.6, we refer to published attacks in first column of the table. They are based on non-control flow manipulation described at the second column. All these attacks do not modify the sequences of system calls generated by the execution of the program but violate a trivial invariant that is generated and propagated along the control structure of the program and that is detected by our model. Notice, that attacks listed in S. chen et al. [23] are detected by data flow integrity mechanisms implemented in M. Castro et al. [22] and that connect read values to instructions that write those values. These are exactly the kind of dependencies that a tool based on runs, like Daikon, generates. Also, those are exactly the kinds of likely invariants that BLAST will manage to prove true and, therefore, will appear in our model.

Reference	Program	Attacks
[23]	ghttpd	overwrite filename
[23]	wu-ftpd	overwrites userid
[39]	rm.c	race condition overwrites path name
[102, 123]	ssh	overwrite authentificated flag to non-zero

Figura 6.6: Non-control flow attacks detected by our methods.

To illustrate the effectiveness of our monitor, it detects any attacks generated automatically by the automated mimicry attacks generator described in C. Kruegel et al. [75].

Theorem 27. ***(Effectiveness)*** *Let $\mathcal{P}$ be a program, and the GSCG the guarded system call graph of $\mathcal{P}$. Any attack on $\mathcal{P}$ automatically generated by the framework described in C. Kruegel et al. [75] is going to be detected by GSCG.* □

6.3.4 Vulnerability Auditing

In some cases our model detect application logic bugs efficiently.

Theorem 28. ***(Precision)*** *Let $\mathcal{P}$ be a program, and let GSCG be the guarded system call graph of $\mathcal{P}$. If GSCG is build from deterministic abstract state graphs, then the sequences of systems calls accepted by GSCG are exactly those accepted by the $\mathcal{P}$.* □

Theorem 28 states that if the abstract state graph is deterministic, then there is an equivalence between the application's code and its abstract state graph. That is, the model is not an overapproximation of the application behavior but is an exact abstraction. Therefore, not only a deviation from the model is an attack, but the model itself is a finite faithful representation of the original program that can be easily evaluated against a specification of application for logic bugs and vulnerabilities.

As we can see, the theoretical framework introduced in the previous sections has been established. Moreover, we note that any other assertion checker, invariant generation or dynamic tools can be considered in this framework.

6.4 Chapter Discussions: Performances and Limitations

Any intrusion or malware detection system analysis will be strongly re-enforced by the presence of these pre-computed invariants and will be weakened by their absence. Our platform is flexible as any invariant generation method could be incorporated. In other words, it is an open architecture, where any invariant generation tool can be connected into a *tool bus* where invariants, expressed in different logics, will help in the identification of malicious behavior or in the construction of a precision intrusion detection system.

We provide an extensible invariant-based formal platform for malware analysis. These invariants would concisely capture the semantic of the malicious behavior of this family of viruses and should be, then, associated to fewer semantic aware signatures. Following the lines of such theoretical frameworks, we proposed host-based intrusion detection systems using automatically generated models where system calls are guarded by pre-computed invariants in order to report any deviation observed during the execution of an application. Our methods also provide techniques for the detection of logic errors and vulnerabilities in applications. By analogy with proof carrying code [98] approaches, our methods and their automated proofs and inductive invariants generation provide use with a very precise intrusion detection and malware identification system.

In terms of limitations, we cannot guarantee that all obfuscation techniques would fail as the analysis depends on the formal methods that we have applied. It would be essential to build a tool bus where any formal methods for static and dynamic analysis could communicate their likely-invariants or inductive invariants. These communications between tools would augment the productivity of invariants as a tool could discover new invariants once it receives an invariant that it could not compute. In other words, this tool bus would not be only a tool chain but a communication system between formal methods handling different logics. Actually it is one of the main recent challenge for the formal methods community [112]. Also, most formal methods applies to source code, but since we would prefer static analysis tools that apply directly on object code, we note that this could imply tedious implementations in some cases.

Capítulo 7

Conclusions

Abstract: In this final Chapter, we offer our conclusions.

7.1 Final Conclusions

We proposed new computational methods that can automate the discovery and strengthening of non-linear interrelationships among the variables of a program that contains non-linear loops, that is, programs that display multivariate polynomial and fractional manipulations. Our methods automatically generate basis for non-trivial non-linear loop invariants ideals, providing an efficient treatment of complicated non-linear loop programs. We succeeded in reducing the non-linear loop invariant generation problem to the computation of eigenspaces of associated endomorphisms.

We described powerful computational methods, relying on linear algebraic methods, that can generate basis for non-linear invariant ideals of non-linear hybrid systems. To do so, we provided methods to generate non-trivial basis of provable invariants for local continuous evolution modes described by non-linear differential rules. These invariants can provide precise over-approximations of the set of reachable states in the continuous state space. As a consequence, they can determine which discrete transitions are possible and can also verify if a given property is fulfilled or not.

We first showed that the preconditions for discrete transitions and the Lie-derivatives for continuous evolution can be viewed as morphisms and can be suitably represented by matrices. The new relaxed consecution requirements, extended to fractional scaling, are also encoded as morphisms represented by matrices with terms that can be used to approximate the consecution conditions. Our invariant generation methods also embody strategies to estimate degree bounds, leading to the discovery of rich classes of inductive, provable invariants.

In order to verify safety properties expressed using transcendental functions, and to reason symbolically about formal power series, it is necessary to be able to generate formal power series invariants. We presented the first verification methods that automatically generate basis of invariants expressed by *multivariate formal power series* and *transcendental functions*. We also discussed their convergence over hybrid systems that exhibit non linear models. We provided resolution and convergence analysis for techniques that can generate non trivial bases of provable multivariate formal power series and can also generate transcendental invariants for each local continuous evolution rules. We reduced the invariant generation problem to linear algebraic matrix manipulations and presented an analysis of these matrices, from which we provide effective methods for solving the original problem. The formal power series invariants generated are often composed by the expansion of some well-known transcendental functions *e.g. log* and *exp*. They also have an analysable closed-form which facilitates the use of the invariants when verifying safety properties.

Our examples, dealing with non linear continuous evolution similar to those present today in many critical hybrid embedded systems, showed the strength of our results. In some of those examples we proved that they do not have “finite” polynomial invariants. These are results which truly extend the limits of any recent approach to the invariant generation problem. Moreover, for each invariant generation problem we deal with in this thesis, we succeeded in establishing very general sufficient conditions that guarantee the existence and allow for the computation of invariant ideals for situations that can not be treated by other present invariant generation approaches. These conditions could be directly used by any current or future constraint-based invariant generation methods, or by any methods based on over-approximations and reachability. Moreover, such linear algebraic methods have lower complexities than the mathematical foundations of the previous approaches that needed Gröbner basis computations, quantifier eliminations or cylindrical algebraic decompositions.

We also extended the domain of applications for invariant generation methods to the area of security. More precisely, we presented a *theoretical basis* for the design of static and dynamic analysis platforms which can lead to suitable architectures for automatic malware analysis. We showed how formal methods, involving program static and dynamic analysis, can be used to build such architectures. By doing so, we provided an extensible invariant-based platform for malware analysis. Further, in the line of such theoretical frameworks, we showed how we can detect the most virulent intrusions attacks using these invariants. We proved that any static analysis based malware or intrusion detection system will be strongly re-enforced by the presence of pre-computed invariants. Our platform is open and flexible, in such a way that any current and future invariant generation methods can be incorporated to it, in order to help in the identification of malicious behavior or in the construction of precision intrusion detection systems.

In summary, this thesis rigorously developed many mathematical techniques, based on linear algebraic tools and their associated algorithms, for the analysis of programs and hybrid systems. The novelty of our approaches provides the bases for future intensive studies both in the filed of invariant generation for static program analysis, as well as for the verification of safety and security properties.

7.2 Conclusões Finais

Propusemos novos métodos computacionais que podem automatizar a descoberta e fortalecimento das inter-relações não-lineares entre as variáveis de um programa que contém laços não-lineares, ou seja, programas que exibem manipulações polinomiais multivariadas e fracionárias. Nossos métodos geram automaticamente bases de ideais de invariantes não-triviais para laços não-lineares, proporcionando um tratamento eficiente de programas não-lineares complexos. Conseguimos reduzir o problema de geração de invariantes para laços não-lineares para o cálculo de "eigenspaces" de endomorfismos associados. Descrevemos métodos computacionais poderosos que podem gerar bases de ideais para invariantes não-lineares para sistemas híbridos não-lineares. Para tal, fornecemos métodos para gerar bases de invariantes não-triviais locais para modos de evolução contínua, descritos por regras diferenciais não-lineares. Estas invariantes podem fornecer sobre-aproximações precisas do conjunto de estados alcançáveis no espaço de estado contínuo. Como conseqüência, elas podem determinar quais das transições discretas são possíveis e também podem verificar se uma propriedade determinada é ou não verificada.

Primeiramente, mostramos que as condições prévias para transições discretas e as derivadas de Lie para evoluções contínuas podem ser vistas como morfismos e podem ser adequadamente representadas por matrizes. Os novos requisitos para consecução relaxada, estendidos para escalas fracionárias, também são codificados como morfismos representados por matrizes, com termos que podem ser usados para aproximar as condições de consecução. Nossos métodos de geração de invariantes também incorporam estratégias para estimar limites do grau dos polinômios, levando à descoberta de ricas classes de invariantes indutivas. Para verificar as propriedades de segurança expressas através de funções transcendentais, e raciocinar simbolicamente sobre séries de potência formais, é necessário a capacidade de gerar invariantes sobre séries de potência formais. Apresentamos os primeiros métodos de verificação que automaticamente geram bases de invariantes expressas por séries de potências multivariadas formais e por funções transcendentais. Também discutimos a convergência de tais séries geradas sobre sistemas híbridos, que exibem modelos não lineares. Fornecemos a análises de convergência para técnicas que podem gerar bases não triviais de séries de potência multivariadas formais e também podem gerar invariantes transcendentais para cada regra local de evolução contínua. Reduzimos o problema de geração de invariante para problemas envolvendo manipulações algébricas lineares de matrizes, e apresentamos uma análise destas matrizes, a partir do qual fornecemos métodos eficazes para resolver o problema original. As séries formais geradas são muitas vezes compostas pela expansão de algumas funções transcendentais bem conhecidos, tais como "*log*" e "*exp*". Elas também apresentam formas fechadas analisáveis que facilitam o uso de invariantes para verificar propriedades de segurança.

Os exemplos apresentados, lidando com evolução não linear contínua semelhantes àquelas presentes hoje em sistemas híbridos ou embarcados críticos, mostroram a força dos resultados. Alguns desses exemplos provam que os sistemas subjacentes não têm invariantes polinomiais "finitas". Estes são resultados que realmente estendem os limites de outras abordagem para o problema da geração de invariantes. Além disso, para cada problema de geração de invariante com que lidamos nesta tese conseguimos estabelecer condições suficientes muito gerais que garantem a existência e permitem o cálculo de ideais invariantes para situações que não podem ser tratadas por outras abordagens recentes de geração de invariantes. Estas condições podem ser diretamente usadas por quaisquer métodos de geração de invariantes baseados em restrições, atuais ou futuros, ou por quaisquer métodos de análise baseados em sobre-aproximações e acessibilidade. Além disso, tais métodos algébricos lineares apresentam complexidades computacionais menores do que outros cujos fundamentos matemáticos necessitam calcular bases de Gröbner, eliminar quantificador ou usar decomposições algébricas cilíndricas.

Também ampliamos o domínio das aplicações para métodos de geração de invariantes para a área de segurança. Mais precisamente, apresentamos uma base teórica para a concepção de plataformas de análise estática e dinâmica que podem levar arquiteturas adequadas para a análise automática de intrusões ("malwares"). Mostramos como os métodos formais, envolvendo a análise estática e dinâmica de programas, podem ser usados para construir tais arquiteturas. Fornecemos, assim, uma plataforma extensível baseada em invariantes para análise de malwares. Além disso, mostramos como podemos detectar os ataques mais virulentos usando essas invariantes. Provamos que qualquer análise estática de malwares ou sistemas de detecção de intrusão, será fortemente reforçada pela presença de invariantes pré-computadas. A plataforma propsota é aberta e flexível, de tal forma que métodos de geração de invariantes, atuais e futuros, podem ser incorporados a ela, auxiliando na identificação de comportamentos maliciosos ou na construção de sistemas de detecção de intrusão mais precisos.

Apêndice A

Proofs

In this Chapter we could find for each chapters, the collection of all theorem, lemma, corollary and their associated proofs.

A.1 Proofs of Chapter 2

A.1.1 Proofs of Section 3.4.1

Theorem 1
Consider a transition system corresponding to a loop τ as described in Eq. (3.4). A polynomial Q in $\mathbb{R}[X_1, .., X_n]$ is a λ-scale invariant for constant-scale consecution with *parametric constant* $\lambda \in \mathbb{R}$ for τ if and only if

$$Q(L_1(X_1, .., X_n), .., L_n(X_1, .., X_n)) = \lambda Q(X_1, .., X_n).$$

$\square$

Demonstração. if

$$Q(X'_1, .., X'_n) - \lambda Q(X_1, .., X_n)$$

belongs to the ideal I generated by the family

$$(X'_1 - L_1, \dots, X'_n - L_n),$$

then there exists a family

$$(A_1, \dots, A_n)$$

of polynomials in

$$\mathbb{R}[X'_1, .., X'_n, X_1, .., X_n]$$

such that

$$Q(X'_1, .., X'_n) - \lambda Q(X_1, .., X_n) = (X'_1 - L_1)A_1 + \cdots + (X'_n - L_n)A_n.$$

Letting

$$X'_i = L_i,$$

we obtain that

$$Q(L_1(X_1, ..., X_n), .., L_n(X_1, ..., X_n)) = \lambda Q(X_1, ..., X_n).$$

Conversely suppose

$$Q(L_1(X_1, \dots, X_n), .., L_n(X_1, \dots, X_n)) = \lambda Q(X_1, \dots, X_n),$$

then as $Q(X'_1, .., X'_n)$ is equal to $Q(L_1, .., L_n)$ modulo the ideal I, we get that

$$Q(X'_1, .., X'_n) = \lambda Q(X_1, \dots, X_n)$$

modulo I. $\square$

Theorem 2
A polynomial Q of $\mathbb{R}_r[X_1, .., X_n]$ is λ-invariant for constant-scale consecution if and only if there exists an eigenvalue λ of M such that Q belongs to the eigenspace corresponding to λ. $\square$

Demonstração. Let Q be a polynomial in $\mathbb{R}_r[X_1, .., X_n]$.

$$\begin{array}{cc} & (Q(L_1(X_1, ..., X_n), .., L_n(X_1, ..., X_n)) = \lambda Q(X_1, ..., X_n)) \\ \Leftrightarrow & (\mathscr{M}(Q) = \lambda Q) \\ \Leftrightarrow & (\mathscr{M}(Q) = \lambda Id(Q)) \\ \Leftrightarrow & ((\mathscr{M} - \lambda Id)(Q) = 0_{\mathbb{R}[X_1,..,X_n]}) \\ \Leftrightarrow & (Q \in Ker(M - \lambda I)), \end{array}$$

Using the definition of an invariant and theorem 1, we can see that Q will be a λ-scale invariant if and only if it belong to the eigenspace correspoinding to λ. $\square$

Corollary 1 Let M the matrix introduce in this section, departing from its charaterisitcs one could find several decidable classes for the problem of finding a *non-trivial λ-invariant*. For instance one can list the following *decidable* classes:

- M is block triangular (with 4×4 blocks or less) ,
- Eigenspace associated with eigenvalue 1 is of dimension greater than 1. $\square$

Demonstração. Suppose M is block triangular with blocks 4×4 or less, then it's characteristic polynomial will a product of polynomials of degree less than four, whose roots can be calculated by Lagrange's resolvent method [76].

For the second assertion, we already know that 1 is an eigenvalue, suppose that the corresponding eigenspace is of dimension exactly one, then the only vectors in that space are the constant polynomials. Whereas if it is of dimension two or more, than we get polynomials that are non trivial in the eigenspace. Looking at theorem 2 to come, we see that it is particularly interesting case. $\square$

A.1.2 Proofs of Section 3.4.2

Theorem 3
A polynomial Q in $\mathbb{R}_r[X_1, .., X_n]$ is an inductive invariant for the affine loop with initial values $(u_1, \ldots, u_n)$ if and only if there is an eigenvalue λ of M such that Q is in the intersection of the eigenspace of λ and the hyperplane $Q(u_1, \ldots, u_n) = 0$. $\square$

Demonstração. We first consider theorem 14.

The initiation step defines on $\mathbb{R}_r[x_1, \dots, x_n]$ a linear form on this space, namely, $I_u : P \mapsto P(u_1, ..., u_n)$. Hence, initial values correspond to a hyperplane of $\mathbb{R}_r[X_1, .., X_n]$ given by the kernel I_u, which is

$$\{Q \in \mathbb{R}_r[X_1, .., X_n] | Q(u_1, \dots, u_n) = 0\} .$$

If we add initial conditions of the form $(x_1(0) = u_1, \dots, x_n(0) = u_n)$, we are looking for a λ-scale invariant in $\mathbb{R}_r[x_1, \dots, x_n]$ that belongs to the hyperplane $P(u_1, \dots, u_n) = 0$, *i.e.*, we are looking for Q in

$$ker(M - \lambda I) \cap \{P \mid P(u_1, \dots, u_n) = 0\} .$$

□

Corollary 2
There will be a non-null invariant polynomial for any given initial values if and only if there exists an eigenspace of M with dimension at least 2. □

Demonstração.
(⇒) If there is a λ-scale invariant for any initial value. Then the corresponding eigenspace has dimension at least 2. Indeed, if the space was of dimension only 1 (which is at least necessary to have λ-invariants).Taking any nonzero vector Q in the eigenspace (i.e. a λ-invariant), Q should lie in any hyperplane of initial values,i.e. for every n-tuple $(u_1, \dots, u_n)$, one would have $Q(u_1, \dots, u_n) = 0$, i.e $Q = 0$, which is absurd.

(⇐) Any eigenspace of M with dimension at least 2 will intersect any space (semi-hyperplan, ...) given by any initial constraints. As any hyperplane is of codimension one in V_r, it must have a nonzero intersection with any subspace of dimension strictly greater than one. □

A.1.3 Proofs of Section 3.5.1

Theorem 4
A polynomial Q in $\mathbb{R}[X_1, .., X_n]$ is a T-scale discrete invariant for polynomial-scale consecution with *parametric polynomial* $T \in \mathbb{R}[X_1, ..., X_n]$ for τ if and only if

$$Q(P_1(X_1, .., X_n), .., P_n(X_1, .., X_n)) = T(X_1, .., X_n)Q(X_1, .., X_n).$$

□

Demonstração. If $Q(X'_1, .., X'_n) - TQ(X_1, .., X_n)$ belongs to the ideal I generated by the family

$$(X'_1 - P_1, \dots, X'_n - P_n),$$

then there exists a family

$$(A_1, \dots, A_n)$$

of polynomials in $\mathbb{R}[X'_1, .., X'_n, X_1, .., X_n]$ such that

$$Q(X'_1, .., X'_n) - \lambda Q(X_1, .., X_n) = (X'_1 - P_1)A_1 + \cdots + (X'_n - P_n)A_n.$$

Letting $X'_i = P_i$, we obtain that

$$Q(P_1(X_1, ..., X_n), ..., P_n(X_1, ..., X_n)) = TQ(X_1, ..., X_n).$$

Conversely suppose

$$Q(P_1(X_1, \dots, X_n), .., P_n(X_1, \dots, X_n)) = TQ(X_1, \dots, X_n),$$

then as $Q(X'_1, .., X'_n)$ is equal to $Q(P_1, .., P_n)$ modulo the ideal I, we get that $Q(X'_1, .., X'_n) = \lambda Q(X_1, \dots, X_n)$ modulo I. □

A.1.4 Proofs of Section 3.5.2

Theorem 5
Consider M as described above. Then, there will be a T-scale discrete invariant if and only if there exists a matrix L (corresponding to $P \mapsto TP$) such that $M - L$ has a nontrivial kernel. Further, any vector in the kernel of $M - L$ will give a T-scale invariant. □

Demonstração. Let Q be a polynomial in $\mathbb{R}[X_1, .., X_n]$. In fact, a polynomial Q is T-invariant if and only if

$$Q(P_1(X_1, .., X_n), .., P_n(X_1, .., X_n)) = T(X_1, .., X_n)Q(X_1, .., X_n),$$

i.e. if and only if

$$\mathscr{M}(Q) = \mathscr{L}(Q) \Leftrightarrow (\mathscr{M} - \mathscr{L})(Q) = 0_{\mathbb{R}[X_1, .., X_n]}.$$

Writing this in equivalent terms of matrices:

$$((M - L)Q = 0) \Leftrightarrow (Q \in Ker(M - L)),$$

we get the statement of the theorem. □

Theorem 6
There is a non trivial T-scale invariant if and only if the polynomials $(V_1,..,V_s)$ admit a common root, other than the trivial one $(0,\dots,0,1)$. □

Demonstração. From linear algebra, we know that $M-L$ with a non trivial kernel is equivalent to it having rank strictly less than the dimension $v(r)$ of $\mathbb{R}_r[x_1,\dots,x_n]$.

This is equivalent to the fact that each $v(r)\times v(r)$ sub-determinant of M_D-L_T is equal to zero.

Those determinants are polynomials with variables $(t_1,..,t_{v(d-1)})$, which we will denote by

$$V_1(t_1,...,t_{v(d-1)}),...,V_s(t_1,...,t_{v(d-1)}).$$

From the form of L, this is zero when

$$(t_1,...,t_{v(d-1)})=(0,...,0).$$

Hence, in this case, $M-L$ has its last column equal to zero, giving a common root for these polynomials, corresponding to the constant invariants. □

A.1.5 Proofs of Section 3.5.3

Theorem 7
Let Q be in $\mathbb{R}_r[X_1,..,X_n]$. Then Q is an inductive invariant for the transition system with initial values $(u_1,..,u_n)$ if and only if there exists a matrix $L\neq 0$ (the one of $P\mapsto TP$), corresponding to T in $\mathbb{R}_e[X_1,..,X_n]$, such that Q is in the intersection of $Ker(M-L)$ and the hyperplane given by the initial values $Q(u_1,\dots,u_n)=0$. The invariants correspond to vectors in the intersection □

Demonstração. We first consider theorem 5.

The initiation step defines on $\mathbb{R}_r[x_1,\dots,x_n]$ a linear form on this space, namely,

$$I_u : P\mapsto P(u_1,...,u_n).$$

Hence, initial values correspond to a hyperplane of $\mathbb{R}_r[X_1,..,X_n]$ given by the kernel I_u, which is

$$\{Q\in\mathbb{R}_r[X_1,..,X_n]|Q(u_1,\dots,u_n)=0\}.$$

If we add initial conditions of the form

$$(x_1(0)=u_1,\dots,x_n(0)=u_n),$$

we are looking for a T-scale differential invariant in $\mathbb{R}_r[x_1,\dots,x_n]$ that belongs to the hyperplane

$$P(u_1,\dots,u_n)=0,$$

i.e., we are looking for Q in

$$ker(M - L) \cap \{P \mid P(u_1, \dots, u_n) = 0\} .$$

$\square$

Corollary 3
There are non-trivial invariant for any given initial values if and only if there exists a matrix L such that $Ker(M - L)$ has dimension at least 2. The basis of $Ker(M - L)$ being a basis for non-trivial invariants. $\square$

Demonstração.

- ($\Rightarrow$) If there is a T-scale invariant for any initial value, then the corresponding eigenspace has dimension at least 2. Indeed, if the space was of dimension only 1 (which is at least necessary to have T-invariants), taking any non-zero vector Q in the eigenspace (i.e. a T-invariant), Q should lie in any hyperplane of initial values,i.e. for every n-tuple $(u_1, \dots, u_n)$, one would have $Q(u_1, \dots, u_n) = 0$,i.e $Q = 0$, which is absurd.

- ($\Leftarrow$) Any intersection between an eigenspace of $M_D - L_T$ with dimension at least 2 will intersect any space (semi-hyperplane, ...) given by any initial constraints.

$\square$

A.1.6 Proofs of Section 3.6

Corollary 8
Let $M \bar{-} L = U \cdot S \cdot V$ be the singular value decomposition of matrix $M \bar{-} L$ described just above. There will be a non trivial T-invariant for any given initial condition if and only if the number of non-zero elements in matrix S is less than $v(r) - 2$, where $v(r)$ is the dimension of $\mathbb{R}_r[x_1, \dots, x_n]$. Moreover, the orthonormal basis for the nullspace obtained from the decomposition directly gives an ideal for non-linear invariants. $\square$

Demonstração. The right singular vectors corresponding to vanishing singular values of $M \bar{-} L$ span the null space of $M \bar{-} L$. The left singular vectors corresponding to the non-zero singular values of $M \bar{-} L$ span the range of $M \bar{-} L$. As a consequence, the rank of $M \bar{-} L$ equals the number of non-zero singular values which is the same as the number of non-zero elements in the matrix S. $\square$

A.1.7 Proofs of Section 3.7

Theorem 8
A polynomial Q in $\mathbb{R}[X_1,..,X_n]$ is a F-scale invariant for fractional discrete scale consecution with a *parametric fractional* $F \in \mathbb{R}(X_1,..,X_n)$ for τ if and only if

$$Q\left(\frac{P_1}{Q_1},..,\frac{P_n}{Q_n}\right) = FQ.$$

□

Demonstração. If $Q(X'_1,..,X'_n) - FQ(X_1,..,X_n)$ belongs to the fractional ideal J generated by the family

$$(X'_1 - P_1/Q_1, \dots, X'_n - P_n/Q_n),$$

then there exists a family

$$(A_1, \dots, A_n)$$

of fractional functions in

$$\mathbb{R}(X'_1,..,X'_n,X_1,..,X_n)$$

such that

$$Q(X'_1,..,X'_n) - FQ(X_1,..,X_n) = (X'_1 - P_1/Q_1)A_1 + \cdots + (X'_n - P_n/Q_n)A_n.$$

Letting

$$X'_i = \frac{P_i}{Q_i},$$

we obtain that

$$Q(\frac{P_1}{Q_1},..,\frac{P_n}{Q_n}) = \lambda Q(X_1,\dots,X_n).$$

Conversely suppose

$$Q(\frac{P_1}{Q_1},..,\frac{P_n}{Q_n}) = FQ(X_1,..,X_n),$$

then as $Q(X'_1,..,X'_n)$ is equal to $Q(\frac{P_1}{Q_1},..,\frac{P_n}{Q_n})$ modulo the ideal J, we get that $Q(X'_1,..,X'_n) = FQ(X_1,..,X_n)$ modulo J. □

Theorem 9

Consider M and L as described above. Then, there exists F-scale invariants (where F is of the form T/Π^r) if and only if there exists a matrix L such that $Ker(M-L) \neq \emptyset$. In this situation, any vector in the kernel of $M-L$ will give a F-scale discrete invariant. □

Demonstração. Let Q be a polynomial in $\mathbb{R}[X_1,..,X_n]$. In fact, a polynomial Q is T/Π^r-invariant if and only if

$$Q(P_1(X_1,..,X_n),..,P_n(X_1,..,X_n)) = T/\Pi^r(X_1,..,X_n)Q(X_1,..,X_n),$$

which is equivalent to

$$\Pi^r Q(P_1(X_1,..,X_n),..,P_n(X_1,..,X_n)) = T(X_1,..,X_n)Q(X_1,..,X_n),$$

i.e. if and only if

$$(\mathscr{M}(Q) = \mathscr{L}(Q)) \Leftrightarrow ((\mathscr{M} - \mathscr{L})(Q) = 0_{\mathbb{R}[X_1,..,X_n]}$$

Writing this in equivalent terms of matrices:

$$((M - L)Q = 0) \Leftrightarrow (Q \in Ker(M - L)),$$

we get the statement of the theorem. □

Theorem 10
We have a non trivial invariant if and only if there exists a matrix L such that the intersection of the kernel of $M - L$ and the hyperplane given by the initial values is not zero, the invariants corresponding to vectors in the intersection. □

Demonstração. We first consider theorem 14.

The initiation step defines on $\mathbb{R}_r[x_1,\ldots,x_n]$ a linear form on this space, namely, $I_u : P \mapsto P(u_1,...,u_n)$. Hence, initial values correspond to a hyperplane of $\mathbb{R}_r[X_1,..,X_n]$ given by the kernel I_u, which is

$$\{Q \in \mathbb{R}_r[X_1,..,X_n] | Q(u_1,\ldots,u_n) = 0\}.$$

If we add initial conditions of the form $(x_1(0) = u_1,\ldots,x_n(0) = u_n)$, we are looking for a *strong*-scale differential invariant in $\mathbb{R}_r[x_1,\ldots,x_n]$ that belongs to the hyperplane $P(u_1,\ldots,u_n) = 0$, *i.e.*, we are looking for Q in $ker(M-L) \cap \{P \mid P(u_1,\ldots,u_n) = 0\}$. □

Corollary 5 We will have a non-trivial invariant for any non-trivial initial value if there exists a matrix L such that the dimension of $Ker(M - L)$ is at least 2. □

Demonstração.

- ($\Rightarrow$) If there is a non-trivial F-scale invariant for any initial value, then the corresponding eigenspace has dimension at least 2. Indeed, if the space was of dimension only 1 (which is at least necessary to have F-invariants), taking any non-zero vector Q in the eigenspace (i.e. a F-invariant), Q should lie in any hyperplane of initial values,i.e. for every n-tuple $(u_1, \ldots, u_n)$, one would have $Q(u_1, \ldots, u_n) = 0$,i.e $Q = 0$, which is absurd.

- ($\Leftarrow$) Any intersection between an eigenspace of M with dimension at least 2 will intersect any space (semi-hyperplane, ...) given by any initial constraints.

□

A.1.8 Proofs of Section 3.8

Theorem 11
Let $I = \{I_1, ..., I_k\}$ a set of ideals in $\mathbb{R}[X_1, ..., X_n]$ such that $I_j = (f^{(j)}{}_1, ..., f^{(j)}_{n_j})$ where $j \in [1, k]$. Let's $\otimes(I_1, ..., I_k) = \{\delta_1, ..., \delta_{n_1 n_2 \ldots n_k}\}$ such that all elements δ_i in $\otimes(I_1, ..., I_k)$ are formed by the product of one element from each ideal in I. Assume that all I_js are ideals of invariants for a loop at location l_j described by a transition τ_j. Now, if all l_j describe the same location/program point l, then we have several transitions *looping* at the same point. So we obtain an encoding of possible execution paths of a loop containing *conditional statements*. Then $\otimes(I_1, ..., I_k)$ is an ideal of non-trivial non-linear invariants for the entire loop located at l. □

Demonstração. Let $f^{(j)}_1, ..., f^{(j)}_{n_j} \in K[X_1, ..., X_n]$ such that $I_j = (f^{(j)}_1, ..., f^{(j)}_{n_j})$, forall j in $[1, k]$.

Let $\beta \in (\otimes(I_1, ..., I_k))$, then there exists $e_1, ..., e_{n_1 n_2 \ldots n_k} \in K[X_1, .., X_n]$ such that $\beta = e_1\delta_1 + ... + e_{n_1 n_2 \ldots n_k}\delta_{n_1 n_2 \ldots n_k}$. Also, by construction of $\otimes(I_1, ..., I_k)$ we know that: $\forall r \in [1, ..., n_1 n_2 ... n_k]$, $\delta_r \in \otimes(I_1, ..., I_k)$.

In other words, $\exists(\alpha^{(r)}_1, ..., \alpha^{(r)}_k) \in I_1 \times ... \times I_k$ such that $\delta_r = \Pi^k_{i=0}\alpha^{(r)}_i$.

Then we have $\beta = \sum^{n_1 n_2 \ldots n_k}_{j=1}[\lambda_j \Pi^k_{i=1}\alpha^{(j)}_i]$.

Now, for all m in $[1, k]$, if I_m correspond to a pre-computer inductive ideal of invariant associated to one of the transition τ_m at the location l, then $\forall j \in [1, n1n2..nk]$, $\alpha^{(j)}_m(X1, ..., Xn) = 0$. And so $\forall j \in [1, n1n2..nk]$, $\Pi^k_{i=1}\alpha^{(j)}_i = 0$.

Finally we obtain $\beta(X_1, ..., X_n) = 0$ for all m in $[1, n_1 n_2..n_k]$. In other words, $\beta(X_1, ..., X_n) = 0$ is an algebraic assertion true at any step of the iteration of the loop for any transition τ_m that could possibily taken. Then $(\beta(X_1, ..., X_n) = 0)$ is an inductive invariant and we can conclude that $(\otimes(I_1, ..., I_k))$ is an ideal of inductive invariant. □

A.2 Proofs of Chapter 3

A.2.1 Proofs of Section 4.3

Theorem 12
Let P be a continuous function and let

$$\mathcal{S} = \begin{bmatrix} \dot{X}_1(t) = P_1(X_1(t), .., X_n(t)) \\ \vdots \\ \dot{X}_n(t) = P_n(X_1(t), .., X_n(t) \end{bmatrix}$$

be a differential rule, with initial condition $(x_1, .., x_n)$. Any polynomial which is a P-scale differential invariant for these initial conditions is actually an inductive invariant. $\square$

Demonstração. Suppose $Q \in \mathbb{R}[X_1, .., X_n]$ is such an invariant. Then if $(X_1(t), .., X_n(t))$ is a solution of (S), by the definition of P-scale invariant one has

$$D_Q(P_1, .., P_n, X_1, .., X_n) = PQ(X_1, .., X_n).$$

Call $f(t)$ the function $Q(X_1(t), .., X_n(t))$. Then we get

$$\dot{f}(t) = P(X_1(t), .., X_n(t))f(t).$$

Call $R(t)$ an anti-derivative of $P(X_1(t), .., X_n(t))$.

Then f must be of the form

$$t \mapsto \lambda e^{R(t)}$$

for some scalar λ.

Now taking into account the initial conditions, if $Q(x_0, .., x_n) = 0 \Leftrightarrow f(0) = 0$, then λ must be zero.

Hence, $f(t) = Q(X_1(t), .., X_n(t))$ is the zero function, and Q is an invariant of (S). $\square$

Theorem 13
There exist a differential rule $\mathcal{S}$ such that its invariants are not Polynomial-scale differential invariant. Such systems are then counter-example for completeness. $\square$

Demonstração. Consider polynomial-scale consecution, the system

$$\begin{bmatrix} \dot{x} = ax(t) \\ \dot{y} = ay(t) + bx(t)y(t) \end{bmatrix}$$

could be cited as counter-example for completeness as its invariants are not P-scale differential invariant. $\square$

A.2.2 Proofs of Section 4.4.1

Lemma 1
Let $Q \in \mathbb{R}[X_1, .., X_n]$ such that $\mathscr{D}_Q(P_1, .., P_n, X_1, .., X_n) = 0$. Then Q is a *strong*-scale differential invariant. □

Demonstração. Let $Q \in \mathbb{R}[X_1, .., X_n]$ be a polynomial such that

$$D_Q(P_1(X_1, .., X_n), .., P_n(X_1, .., X_n), X_1, .., X_n) = 0$$

. then $dQ/dt = 0$ and Q is a strong invariant as $\frac{dX_i(t)}{dt} = P_i(X_1, ..., X_n)$ for all i in $[1, n]$ and by construction of D_Q. □

Theorem 14
A polynomial Q of $\mathbb{R}_r[X_1, .., X_n]$ is a *strong*-scale differential invariant for the differential system (4.1) if and only if it lies in the kernel of M_D. □

Demonstração. let $Q \in \mathcal{R}[X_1, .., X_n]$ be a polynomial. Then

$$\begin{aligned}(\mathscr{D}_Q(P_1, .., P_n, X_1, .., X_n) = 0) &\Leftrightarrow (D(Q) = 0_{K[X_1, .., X_n]})\\ &\Leftrightarrow (Q \in Ker(M_D)).\end{aligned}$$

Using the definition of an invariant and lemma 1, we can see that Q will be a strong-scale invariant if and only if it is in the kernel of M_D. □

Theorem 15 Let Q be in $\mathbb{R}_r[X_1, .., X_n]$. Then Q is an inductive invariant for the differential system with initial values $(u_1, .., u_n)$ if and only if Q is in the intersection of $Ker(M_D)$ and the hyperplane $Q(u_1, \dots, u_n) = 0$. □

Demonstração. We first consider theorem 14.

The initiation step defines on $\mathbb{R}_r[x_1, \dots, x_n]$ a linear form on this space, namely, $I_u : P \mapsto P(u_1, ..., u_n)$. Hence, initial values correspond to a hyperplane of $\mathbb{R}_r[X_1, .., X_n]$ given by the kernel I_u, which is

$$\{Q \in \mathbb{R}_r[X_1, .., X_n] | Q(u_1, \dots, u_n) = 0\}.$$

If we add initial conditions of the form $(x_1(0) = u_1, \dots, x_n(0) = u_n)$, we are looking for a *strong*-scale differential invariant in $\mathbb{R}_r[x_1, \dots, x_n]$ that belongs to the hyperplane $P(u_1, \dots, u_n) = 0$, *i.e.*, we are looking for Q in

$$ker(M_D) \cap \{P \mid P(u_1, \dots, u_n) = 0\}.$$

□

Corollary 6 There exists a strong-scale invariant of degree r for the differential system with initial conditions (any initial conditions, actually), if and only if the kernel of M_D is of dimension at least 2. The basis of $Ker(M_D)$ gives a *basis of a non-trivial invariant ideal* □

Demonstração.

- ($\Rightarrow$) If there is a non-trivial strong-scale invariant for any initial value, then the corresponding eigenspace has dimension at least 2. Indeed, if the space was of dimension only 1 (which is at least necessary to have strong-invariants), taking any non-zero vector Q in the eigenspace (i.e. a strong-invariant), Q should lie in any hyperplane of initial values,i.e. for every n-tuple $(u_1, \dots, u_n)$, one would have $Q(u_1, \dots, u_n) = 0$,i.e $Q = 0$, which is absurd.
- ($\Leftarrow$) Any intersection between an eigenspace of M_D with dimension at least 2 will intersect any space (semi-hyperplane, ...) given by any initial constraints.

□

Lemma 2 Let $Q_1, \dots, Q_n$ be n polynomials in $\mathbb{R}[X_1, \dots, X_n]$. Then there exists a polynomial Q such that $\partial_1 Q = Q_1, \dots, \partial_n Q = Q_n$ if and only if for any $i \neq j$, $1 \leq i, j \leq n$, one has $\partial_i Q_j = \partial_j Q_i$. □

Demonstração. We treat the case of two variables, the case of n variables being a straight generalization.

Suppose that

$$\partial_i Q_j = \partial_j Q_i$$

for each pair (i, j).

We choose a polynomial Q^1, an anti-derivative of Q_1 with respect to x_1.

Now

$$\partial_1(\partial_2 Q^1) = \partial_2(\partial_1 Q^1) = \partial_2 Q_1 = \partial_1 Q_2.$$

Hence

$$\partial_1(\partial_2 Q^1 - Q_2) = 0,$$

and so

$$\partial_2 Q^1 = Q_2 + b(x_2, \dots, x_n)$$

for some function b of $(x_2, \dots, x_n)$ which is actually a polynomial.

Choosing an anti-derivative $B(x_2,\dots,x_n)$ of $b(x_2,\dots,x_n)$ with respect to x_2, one verifies that

$$Q_{1,2} = Q^1 - B(x_2,\dots,x_n)$$

is such that

$$\partial_1 Q_{1,2} = Q_1$$

and

$$\partial_2 Q_{1,2} = Q_2.$$

Now,

$$\partial_1\partial_3 Q_{1,2} = \partial_3\partial_1 Q_{1,2} = \partial_3 Q_1 = \partial_1 Q_3,$$

and

$$\partial_2\partial_3 Q_{1,2} = \partial_2 Q_3$$

as well.

Hence,

$$\partial_3 Q_{1,2} - Q_3 = c(x_3,\dots,x_n)$$

for a polynomial c. Taking C as an anti-derivative of c with respect to x_3, one deduces that

$$Q_{1,2,3} = Q_{1,2} - C$$

is such that

$$\partial_i Q_{1,2,3} = Q_i$$

for $i = 1, 2, 3$.

Repeating the process, we construct $Q_{1,\dots,n}$ such that

$$\partial_i Q_{1,\dots,n} = Q_i,$$

for $i = 1, 2, 3$. □

Theorem 16
There exists a strong-scale invariant for a differential system if and only if there exists $(Q_1,..,Q_n)$ in $Syz(P_1,..,P_n)$ such that for any i,j with $i \neq j$ and $1 \leq i,j, \leq n$, one has $\partial_i Q_j = \partial_j Q_i$. □

Demonstração. Obtain by directly using lemma 2. □

A.2.3 Proofs of Section 4.4.2

Lemma 3
Let $Q \in \mathbb{R}[X_1, ..., X_n]$ such that

$$\mathscr{D}_Q(P_1, .., P_n, X_1, .., X_n) = \lambda Q(X_1, .., X_n).$$

Then Q is a λ-scale invariant. $\square$

Demonstração. Let $Q \in \mathbb{R}[X_1, .., X_n]$ be a polynomial such that

$$D_Q(P_1(X_1, .., X_n), .., P_n(X_1, .., X_n), X_1, .., X_n) = \lambda Q(X_1, .., X_n).$$

. As $\frac{dX_i(t)}{dt} = P_i(X_1, ..., X_n)$ for all i in $[1, n]$ and by construction of D_Q we obtain $\frac{dQ}{dt} = \lambda Q$. So, $\frac{dQ}{dt} - \lambda Q = 0$ and Q is a λ-scale invariant. $\square$

Theorem 17
A polynomial Q of $\mathbb{R}_r[X_1, .., X_n]$ is a λ-scale invariant for continuous-scale consecution of the differential system if and only if there exists an eigenvalue λ of M_D such that Q belongs to the eigenspace of M_D corresponding to λ. $\square$

Demonstração.

$$\begin{array}{cc} & (\mathscr{D}_Q(P_1, .., P_n, X_1, .., X_n) = \lambda Q(X_1, ..., X_n)) \\ \Leftrightarrow & (D(Q) = \lambda Id(Q)) \\ \Leftrightarrow & ((D - \lambda Id)(Q) = 0_{\mathbb{R}[X_1, .., X_n]}) \\ \Leftrightarrow & (Q \in Ker(D - \lambda Id)), \\ \Leftrightarrow & (Q \in Ker(M_D - \lambda I)). \end{array}$$

Using the definition of an invariant and lemma 3, we can see that Q will be a strong-scale invariant if and only if it is in the kernel of M_D. $\square$

Theorem 18
A polynomial Q in $\mathbb{R}_r[X_1, .., X_n]$ is an λ-scale invariant for the differential system with initial values $(u_1, \ldots, u_n)$ if and only if there exists an eigenvalue λ of M_D such that Q belongs to the intersection of the eigenspaces corresponding to λ and the hyperplane $Q(u_1, \ldots, u_n) = 0$. $\square$

Demonstração. We first consider theorem 14.

The initiation step defines on $\mathbb{R}_r[x_1, \ldots, x_n]$ a linear form on this space, namely,

$$I_u : P \mapsto P(u_1, ..., u_n).$$

Hence, initial values correspond to a hyperplane of $\mathbb{R}_r[X_1, .., X_n]$ given by the kernel I_u, which is

$$\{Q \in \mathbb{R}_r[X_1, .., X_n] | Q(u_1, \dots, u_n) = 0\} .$$

If we add initial conditions of the form $(x_1(0) = u_1, \dots, x_n(0) = u_n)$, we are looking for a λ-scale differential invariant in $\mathbb{R}_r[x_1, \dots, x_n]$ that belongs to the hyperplane $P(u_1, \dots, u_n) = 0$, *i.e.*, we are looking for Q in

$$ker(M_D - \lambda I) \cap \{P \mid P(u_1, \dots, u_n) = 0\} .$$

□

Corollary 7 There will be a non-null polynomial invariant for any given initial values if and only if there exists an eigenspace of M_D with dimension at least 2. □

Demonstração.

- $(\Rightarrow)$ If there is a λ-scale invariant for any initial value, then the corresponding eigenspace has dimension at least 2. Indeed, if the space was of dimension only 1 (which is at least necessary to have λ-invariants), taking any non-zero vector Q in the eigenspace (i.e. a λ-invariant), Q should lie in any hyperplane of initial values,i.e. for every n-tuple $(u_1, \dots, u_n)$, one would have $Q(u_1, \dots, u_n) = 0$,i.e $Q = 0$, which is absurd.
- $(\Leftarrow)$ Any intersection between an eigenspace of M_D with dimension at least 2 will intersect any space (semi-hyperplane, ...) given by any initial constraints.

□

A.2.4 Proofs of Section 4.5

Lemma 4
Let $Q \in \mathbb{R}[X_1, .., X_n]$ such that

$$\mathscr{D}_Q(P_1, .., P_n, X_1, .., X_n) = TQ$$

with T in $\mathbb{R}[X_1, .., X_n]$. Then Q is a T-scale invariant. □

Demonstração. Let $Q \in \mathbb{R}[X_1, .., X_n]$ be a polynomial such that

$$D_Q(P_1(X_1, .., X_n), .., P_n(X_1, .., X_n), X_1, .., X_n) = TQ(X_1, .., X_n).$$

. As $\frac{dX_i(t)}{dt} = P_i(X_1, ..., X_n)$ for all i in $[1, n]$ and by construction of D_Q we obtain $\frac{dQ}{dt} = TQ$. So, $\frac{dQ}{dt} - TQ = 0$ and Q is a T-scale invariant. □

Theorem 19
There is a polynomial-scale invariant for the differential system if and only if there exists a matrix L_T in $M(pol)$, corresponding to a polynomial T of $\mathbb{R}_{d-1}[x_1, .., X_n]$, such that $Ker(M_D - L_T)$ is not reduced to zero. And, any vector in the kernel of $M_D - L_T$ will give a T-scale differential invariant. □

Demonstração. The sketch of the proof is induced by the constructed linear algebraic reduction. Assume that there an invariant Q in $\mathbb{R}_r[x_1, \ldots, x_n]$ for differential polynomial-scale consecution corresponding to the differential system. And, there exists a polynomial T such that

$$Q(x_1, \overset{\cdot}{...}, x_n) = T(x_1, .., x_n)Q(x_1, .., x_n).$$

By definition, we have

$$D_Q(P_1, \ldots, P_n, x_1, \ldots, x_n) = TQ$$

and then

$$D(Q) = \overline{T}(Q).$$

In other words $(D - T)(Q) = 0$, i.e. $Q \in Ker(M_D - L_T)$ which means that $Ker(M_D - L_T) \neq \emptyset$. On the other hand if $Ker(M_D - L_T) \neq \emptyset$ then there exists a polynomial P such that $M_D(P) = L_T(P)$. By definition, $D(P) = \overline{T}(P)$ and $D_P(P_1, \ldots, P_n, x_1, \ldots, x_n) = TP$. In other words, $P(x_1, \overset{\cdot}{...}, x_n) = T(x_1, .., x_n)P(x_1, .., x_n)$ and P is a T invariant in $\mathbb{R}_r[x_1, \ldots, x_n]$ for differential polynomial-scale consecution corresponding to the differential system. □

Theorem 20
There is a non trivial T-scale invariant if and only if the polynomials $(E_1, .., E_s)$ admit a common root, other than the trivial one $(0, ..., 0)$. □

Demonstração. From linear algebra, we know that $M_D - L_T$ with a non trivial kernel is equivalent to it having rank strictly less than the dimension $v(r)$ of $\mathbb{R}_r[x_1, \ldots, x_n]$.

This is equivalent to the fact that each $v(r) \times v(r)$ sub-determinant of $M_D - L_T$ is equal to zero.

Those determinants are polynomials with variables $(t_1, .., t_{v(d-1)})$, which we will denote by

$$E_1(t_1, ..., t_{v(d-1)}), ..., E_s(t_1, ..., t_{v(d-1)}).$$

From the form of L_T, this is zero when

$$(t_1, ..., t_{v(d-1)}) = (0, ..., 0).$$

Hence, in this case, $M_D - L_T$ has its last column equal to zero, giving a common root for these polynomials, corresponding to the constant invariants. □

Theorem 21
Let Q be in $\mathbb{R}_r[X_1, .., X_n]$. Then Q is an inductive invariant for the differential system with initial values $(u_1, .., u_n)$ if and only if there exists a matrix $L_T \neq 0$ in $M(pol)$, corresponding to T in $\mathbb{R}_{d-1}[X_1, .., X_n]$, such that Q is in the intersection of $Ker(M_D - L_T)$ and the hyperplane $Q(u_1, \ldots, u_n) = 0$. □

Demonstração. We first consider theorem 19. The initiation step defines on $\mathbb{R}_r[x_1, \ldots, x_n]$ a linear form on this space, namely, $I_u : P \mapsto P(u_1, ..., u_n)$. Hence, initial values correspond to a hyperplane of $\mathbb{R}_r[X_1, .., X_n]$ given by the kernel I_u, which is
$\{Q \in \mathbb{R}_r[X_1, .., X_n] | Q(u_1, \ldots, u_n) = 0\}$. If we add initial conditions of the form $(x_1(0) = u_1, \ldots, x_n(0) = u_n)$, we are looking for a T-scale differential invariant in $\mathbb{R}_r[x_1, \ldots, x_n]$ that belongs to the hyperplane $P(u_1, \ldots, u_n) = 0$, *i.e.*, we are looking for Q in

$$ker(M_D - L_T) \cap \{P \mid P(u_1, \ldots, u_n) = 0\} .$$

□

Corollary 8
There are non-trivial invariants for any given initial values if and only if there exists a matrix L_T in $M(pol)$ such that $Ker(M_D - L_T)$ has dimension at least 2. □

Demonstração.

- ($\Rightarrow$) If there is a T-scale invariant for any initial value, then the corresponding eigenspace has dimension at least 2. Indeed, if the space was of dimension only 1 (which is at least necessary to have T-invariants), taking any non-zero vector Q in the eigenspace (i.e. a T-invariant), Q should lie in any hyperplane of initial values,i.e. for every n-tuple $(u_1, \ldots, u_n)$, one would have $Q(u_1, \ldots, u_n) = 0$,i.e $Q = 0$, which is absurd.
- ($\Leftarrow$) Any intersection between an eigenspace of $M_D - L_T$ with dimension at least 2 will intersect any space (semi-hyperplane, ...) given by any initial constraints.

□

A.2.5 Proofs of Section 4.6

Corollary 9
Let $\bar{M} = U \cdot S \cdot V$ be the singular value decomposition of matrix $\bar{M}$ described just above. There will be a non trivial T-invariant for any given initial condition if and only if the number of non-zero elements in matrix S is less than $v(r) - 2$, where $v(r)$ is the dimension

of $\mathbb{R}_r[x_1, \ldots, x_n]$. Moreover, the orthonormal basis for the nullspace obtained from the decomposition directly gives an ideal for non-linear invariants. $\square$

Demonstração. The right singular vectors corresponding to vanishing singular values of $\bar{M}$ span the null space of $\bar{M}$. The left singular vectors corresponding to the non-zero singular values of $\bar{M}$ span the range of $\bar{M}$. As a consequence, the rank of $\bar{M}$ equals the number of non-zero singular values which is the same as the number of non-zero elements in the matrix S. $\square$

A.2.6 Proofs of Section 4.8

Theorem 22
Let W be a hybrid system and let l be one of its locations. Let $I = \{I_1, ..., I_k\}$ a set of ideals in $\mathbb{R}[X_1, ..., X_n]$ such that $I_j = ({f^{(j)}}_1, ..., f^{(j)}_{n_j})$ where $j \in [1, k]$. Let $\otimes(I_1, ..., I_k) = \{\delta_1, ..., \delta_{n_1 n_2 \ldots n_k}\}$ be such that all elements δ_i in $\otimes(I_1, ..., I_k)$ are formed by the product of one element from each ideal in I. Assume that the I_j's are collections of invariant ideals associated to $\mathcal{S}(l)$, its differential rule, $\mathcal{C}(l)$, its local conditions, and all invariant ideals generated considering all incoming transitions at l. Then $\otimes(I_1, ..., I_k)$ is a non-trivial invariant ideal for location l. $\square$

Demonstração. Let $f^{(j)}_1, ..., f^{(j)}_{n_j}$ in $K[X_1, .., X_n]$ such that $I_j = ({f^{(j)}}_1, ..., f^{(j)}_{n_j})$, forall j in $[1, k]$.Let $\beta \in (\otimes(I_1, ..., I_k))$,then there exists $e_1, .., e_{n_1 n_2 .. n_k}$ in $K[X_1, .., X_n]$ such that $\beta = e_1\delta_1 + .. + e_{n_1 n_2 .. n_k}\delta_{n_1 n_2 .. n_k}$. Also, by construction of $\otimes(I_1, ..., I_k)$ we know that: $\forall r \in [1, .., n_1 n_2 .. n_k]$, $\delta_r \in \otimes(I_1, ..., I_k)$. $\exists(\alpha^{(r)}_1, .., \alpha^{(r)}_k) \in I_1 \times I_2 \times .. \times I_k$ such that $\delta_r = \prod_{i=o}^{k} \alpha^{(r)}_i$. Then we have $\beta = \sum_{j=1}^{n_1 n_2 .. n_k} [\lambda_j \prod_{i=1}^{k} \alpha^{(j)}_i]$. Now, for all m in $[1, k]$, if I_m correspond to a pre-computer inductive ideal of invariant associated to one of the transition τ_m at the location l, then $\forall j \in [1, n_1 n_2 .. n_k]$, $\alpha^{(j)}_m(X_1, .., X_n) = 0$. And so $\forall j \in [1, n_1 n_2 .. n_k]$, $\prod_{i=1}^{k} \alpha^{(j)}_i = 0$. Finally we obtain $\beta(X_1, .., X_n) = 0$ for all m in $[1, n_1 n_2 .. n_k]$. In other words, $(\beta(X_1, .., X_n) = 0)$ is an algebraic assertion true at any step of the iteration of the loop for any transition τ_m that could possibliy taken. Then $(\beta(X_1, .., X_n) = 0)$ is an inductive invariant and we can conclude that $(\otimes(I_1, ..., I_k))$ is an ideal of inductive invariant.

$\square$

Corollary 10
Let l be a state and let $\mathcal{C}(l) \equiv (P_i(x_1, .., x_n) < 0)$ be its semi-algebraic local conditions and Q be an inductive invariant for $\mathcal{D}(l)$, its differential rule, and all ideals of invariants generated considering all incoming transitions at l. Then $(P_i(x_1, .., x_n) - Q(x_1, .., x_n) < 0)$ is an inductive invariant. $\square$

Demonstração. This is straight forward from the fact that $(P_i(x_1, .., x_n) - Q(x_1, .., x_n) < 0)$ will be an invariant as soon as $Q(x_1, .., x_n) = 0$ is an inductive invariant at l. We conclude using theorem 22. $\square$

A.3 Proofs of Chapter 4

A.3.1 Proofs of Section 5.4

Lemma 5
Assume that matrix $A = M_{1,1}$ is triangular,i.e.

$$A = \begin{bmatrix} \lambda_1 & & & & \\ \star & \lambda_2 & & & \\ \star & \star & \ddots & & \\ \star & \star & \star & \lambda_{n-1} & \\ \star & \star & \star & \star & \lambda_n \end{bmatrix}.$$

Then $M_{p,p}$ is also triangular with diagonal terms

$$i_1\lambda_1 + \cdots + i_n\lambda_n,$$

where $i_1 + \cdots + i_n = p$. □

Demonstração. In this case,

$$P_1^j.X^1 = \lambda_j x_j + a_{j,j+1}x_{j+1} + \cdots + a_{j,n}x_n.$$

Now consider the monomial basis

$$P(X) = x_1^{i_1}\dots x_n^{i_n},$$

where $i_1 + \cdots + i_n = p$.

One has

$$\begin{array}{rcl} D_{p,p}(X) & = & i_1 x_1^{i_1-1}\dots x_n^{i_n}(\lambda_1 x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n) \\ & & +i_2 x_1^{i_1}x_2^{i_2-1}\dots x_n^{i_n}(\lambda_2 x_2 + a_{2,3}x_3 + \cdots + a_{2,n}x_n) \\ & & + \cdots + \\ & & +i_n x_1^{i_1}\dots x_n^{i_n-1}(\lambda_n x_n) \\ & = & (i_1\lambda_1 + \cdots + i_n\lambda_n)x_1^{i_1}\dots x_n^{i_n} + \Omega \end{array},$$

and Ω is a sum of higher terms monomials that come after $x_1^{i_1}\dots x_n^{i_n}$ in the ordered basis of $R_p[x_1,\dots,x_n]$.

Then, matrix $M_{p,p}$ corresponding, to $D_{p,p}$ in the canonical ordered basis of $R_p[x_1, \ldots, x_n]$, is:

$$\begin{bmatrix} p\lambda_1 & & & & & & \\ \star & (p-1)\lambda_1 + \lambda_2 & & & & & \\ \star & \star & \ddots & & & & \\ \star & \star & \star & \displaystyle\sum_{k=1}^{n} i_k \lambda_k & & & \\ \star & \star & \star & \star & \ddots & & \\ \star & \star & \star & \star & \star & \lambda_{n-1} + (p-1)\lambda_n & \\ \star & \star & \star & \star & \star & \star & p\lambda_n \end{bmatrix}$$

Thus, it is also triangular with diagonal terms

$$i_1\lambda_1 + \cdots + i_n\lambda_n,$$

where $i_1 + \cdots + i_n = p$. □

A.3.2 Proofs of Section 5.5

Theorem 23
Let F be a λ-invariant for a system S.

Let U be an open subset of $\mathbb{R}^n$, where F is defined by a normally convergent power series.

If there is an initial condition

$$x_1(0), ..., x_n(0)$$

in U such that

$$F(x_1(0), ..., x_n(0)) = 0,$$

then

$$F(x_1(t), ..., x_n(t)) = 0$$

for all t such that $x_1(t), ..., x_n(t)$ remain in U, i.e F is an invariant of S for the initial condition

$$x_1(0), ..., x_n(0).$$

□

Demonstração. As the power series defining F converges normally on U, so does any of its derivatives.

Thus,

$$\dot{F}(x_1(t),...,x_n(t)) = \sum_{i=1}^{n} \partial_i F(x_1(t),...,x_n(t))\dot{x_i}(t) = \lambda F(x_1(t),...,x_n(t))$$

because of the λ-invariant property.

So, $F(x_1(t),...,x_n(t))$ must be equal to

$$t \mapsto ke^{\lambda t}$$

for some constant k.

But as

$$F(x_1(0),...,x_n(0)) = 0,$$

then k is zero, and so is $F(x_1(t),...,x_n(t))$ for any t s.t.

$$(x_1(t),\dots,x_n(t)) \in U.$$

□

A.3.3 Proofs of Section 5.5.1

Theorem 24
Let A be the Jacobian matrix at zero of the polynomial

$$P = (P_1,...,P_n)$$

defining the system S, whose expression is:

$$(\partial_i P_j(0,...,0), i,j \in [1,n]^2).$$

Let $P_k(0,..,0) = 0$. If A is triangularizable with eigenvalues

$$\lambda_1 \leq ... \leq \lambda_n$$

then there exists a λ-invariant formal power series for S when all eigenvalues are positive, or are all negative, with

$$\lambda = \lambda_1.$$

□

Demonstração. Up to a linear change of variables, we can assume that matrix A is triangular with diagonal terms

$$\lambda_1 \leq ... \leq \lambda_n.$$

We know that matrix $M_{k,k}$ has the form described in Lemma 5.

As A is triangular, so is $M_{k,k}$, and its diagonal terms are the real numbers

$$i_1\lambda_1 + \cdots + i_n\lambda_n,$$

where

$$i_1 + \cdots + i_n = k.$$

Hence, the diagonal terms of

$$M_{k,k} - \lambda I_{k+1}$$

are

$$0 \leq \lambda_2 - \lambda ... \leq \lambda_n - \lambda$$

when $k = 1$.

Also, it has a nonzero kernel, and so we can chose a nonzero F_1, such that

$$(M_{1,1} - \lambda I_2)F_1 = 0.$$

For $k \geq 2$ and $i_1 + \cdots + i_n = k$, the diagonal terms

$$i_1\lambda_1 + \cdots + i_n\lambda_n - \lambda$$

of the triangular matrix

$$M_{k,k} - \lambda I_{k+1}$$

are greater than

$$i_1\lambda_1 + \cdots + i_n\lambda_n - \lambda = k\lambda - \lambda > \lambda > 0.$$

So, $M_{k,k} - \lambda I_{k+1}$ is invertible.

Hence we can choose:

$$F_2 = -(M_{2,2} - \lambda I_3)^{-1} M_{1,2} F_1,$$

and then

$$F_3 = -(M_{3,3} - \lambda I_4)^{-1}(M_{1,3}F_1 + M_{2,3}F_2),$$

and recursively,

$$F_k = -(M_{k,k} - \lambda I_{k+1})^{-1}(M_{k-min(k,m)+1,k}F_{k-min(k,m)+1} + \cdots + M_{k-1,k}F_{k-1}).$$

Then,

$$(F_1, F_2, \dots)$$

ia a nonzero solution of the system and the formal power series

$$\sum_i F_i X^i$$

is a λ-invariant. $\square$

Theorem 25
Let A be the Jacobian matrix at zero of the polynomial

$$P = (P_1, ..., P_n)$$

defining a system S, as in Eq. (5.2.2), and whose expression is

$$(\partial_i P_j(0, ..., 0),\ i, j \in [1, n]^2).$$

Assume, further, that $P_k(0, .., 0) = 0$.
Suppose that A is triangularizable with eigenvalues

$$\lambda_1 \leq ... \leq \lambda_n.$$

Denote λ_1 by λ and assume that the eigenspace associated with λ is of dimension at least 2.

Let F_1 and F_2 be two independent λ-invariants.

If there is an open subset U of $\mathbb{R}^n$, over which F_1 and F_2 define two normally convergent power series, then for any initial value

$$(x_{1,0}, \ldots, x_{n,0}),$$

the power series

$$F_2(x_{1,0}, \ldots, x_{n,0})F_1 - F_1(x_{1,0}, \ldots, x_{n,0})F_2$$

defines an inductive invariant on U for the solution of S with initial conditions

$$x_1(0) = x_{1,0}, \ldots, x_n(0) = x_{n,0}.$$

□

Demonstração. Both F_1 and F_2 are convergent for a solution

$$(x_1(t), \ldots, x_n(t))$$

with initial values

$$(x_{1,0}, \ldots, x_{n,0})$$

in U.

Hence, it must stay in U for small t. Moreover, since F_1 and F_2 are independent,

$$F = F_2(x_{1,0}, \ldots, x_{n,0})F_1 - F_1(x_{1,0}, \ldots, x_{n,0})F_2$$

is a nonzero λ-invariant which vanishes at

$$(x_{1,0}, .., x_{n,0}).$$

So, according to Theorem 23, F is an inductive invariant □

A.4 Proofs of Chapter 5

A.4.1 Proofs of Section 6.3.3

Theorem 26
Let $\mathcal{P}$ be a program, and GSCG and E_GSCG be respectively the guarded system call and and the extended system call graph for $\mathcal{P}$.

- If $\mathcal{P}$ contains no recursive functions then any GSCG can be simulated by Dyke Model.
- If $\mathcal{P}$ contains recursive functions then any E_GSCG can be simulated by Dyck Model. □

Demonstração. For the first assertion, we show that GSCG can be simulated with the following encoding. Let G be the GSCG of $\mathcal{P}$ and $t \in \delta_P$ such that $t = (s, g, C, A, s')$.

- If the action A is of the form $Pred(s, F) := \mathsf{True}$ then there exist a transtion in the Dyke Model departing from s and labeled by Pre_Call_F.
- If the action is of the form $Pred(s, F) := \mathsf{False}$ and the guard $Pred(s, F) == \mathsf{True}$ then there exists a transition departing from s labeled by $Post\_Call\_F$ and the top of the Dyke stack is Pre_Call_F.

For the second assertion, we show that E_GSCG can be simulated with the following encoding. Let G be the E_GSCG of $\mathcal{P}$ and $t \in \delta_E$.

- If $t = (s, \epsilon, g, C, \mathsf{Push}(\gamma), s', \gamma)$ and $\gamma = (Pred(s, F), 1)$. Then there exist a transtion in the Dyke Model departing from s and labeled by Pre_Call_F. At s' the top of the Dyke stack is Pre_Call_F and the top of E_G's stack is $(Pred(s, F), 1)$.
- If $t = (s, \gamma, (Pred(s, F) == \mathsf{True}) \wedge g, C, \mathsf{Pop}(\gamma), s', \epsilon$ and $\gamma = ((Pred(s', F), 1)$. Then there exists a transition departing from s labeled by $Post\_Call\_F$ and the top of the Dyke stack is Pre_Call_F.
- The other cases are the same as these describe for the $GSCG$.

For all sequences of system calls $w \in SysC^*$ generated by the *normal* execution of $\mathcal{P}$, if there exists an accepted run of G on w then there exists the corresponding run in the Dyke Model. □

Theorem 27
Let $\mathcal{P}$ be a program, and GSCG the guarded system call graph of $\mathcal{P}$. Any attack on $\mathcal{P}$ automatically generated by the framework described in [75] is going to be detected by GSCG. □

Demonstração. The technique described in [75] consists in finding a sequence

$$sc_1, sc_2, \ldots, sc_n$$

of system calls that are accepted by an intrusion detection model such a Dyck model. The attacker hijacks the application and executes the following sequence:

$$mc_1, sc_1, r_1, mc_2, sc_2, r_2, \ldots, mc_n, sc_n, r_n.$$

That is, it executes a malicious code mc_i before a system call sc_i in which arguments to the system call have been overwritten with values defined by the attacker, executes the system call sc_i with the attacker's arguments, and then finds a set of memory location and corresponding values so that if these memory location are overwritten in a sequence r_i of instructions, the program's flow of execution will return to a legitimate program control point to conform to the control flow integrity enforced by the model. The sequence is repeated for each system call sc_i. The GSCG guards each system call sc_i with an invariant g_i. Therefore, executing the sequence mc_i, sc_i which overrides system calls arguments with attacker's values instead of the sequence lc_i, sc_i consisting of the legitimate block of instruction lc_i preceding the system call sc_i will not follow the program execution paths and will violate the invariant gi that summarizes the data flow in lc_i. □

A.4.2 Proofs of Section 6.3.4

Theorem 28
Let $\mathcal{P}$ be a program, and GSCG the guarded system call graph of $\mathcal{P}$. If GSCG is build from deterministic abstract state graphs, then the sequences of systems calls accepted by GSCG are exactly those accepted by the $\mathcal{P}$. □

Demonstração. If the abstract state graph of $\mathcal{P}$ is deterministic then the GSCG obtained from $\mathcal{P}$'s ASG is an exact over-aproximation. In otherwords, any sequence of the GSCG is a sequence of $\mathcal{P}$. Moreover, any sequence of $\mathcal{P}$ is a sequence/run for the model. □

Referências Bibliográficas

[1] A. Tiwari. Generating box invariants. In *Proc. of the 11th Int. Conf. on Hybrid Systems: Computation and Control HSCC*, 2008.

[2] L. M. Adelman. An abstract theory of computer viruses. In *Advances in Cryptology CRYPTO'88*, 1988.

[3] Behzad Akbarpour and Lawrence C. Paulson. Applications of metitarski in the verification of control and hybrid systems. In *HSCC '09: Proceedings of the 12th International Conference on Hybrid Systems: Computation and Control*, pages 1–15, Berlin, Heidelberg, 2009. Springer-Verlag.

[4] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. h. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[5] Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In *STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 202–211, New York, NY, USA, 2004. ACM.

[6] Franz Baader and Jörg H. Siekmann. Unification theory. pages 41–125, 1994.

[7] David Baelde, Andrew Gacek, Dale Miller, Gopalan Nadathur, and Alwen Tiu. The bedwyr system for model checking over syntactic expressions. In *CADE-21: Proceedings of the 21st international conference on Automated Deduction*, pages 391–397, Berlin, Heidelberg, 2007. Springer-Verlag.

[8] Andreas Bauer, Markus Pister, and Michael Tautschnig. Tool-support for the analysis of hybrid systems and models. In *DATE '07: Proceedings of the conference on Design, automation and test in Europe*, pages 924–929, San Jose, CA, USA, 2007. EDA Consortium.

[9] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Uppaal - a tool suite for automatic verification of real-time systems. In *Hybrid Systems*, pages 232–243, 1995.

[10] S Bensalem, M Bozga, J-C Ghirvu, and L Lakhnech. A transformation approach for generating non-linear invariants. *Static Analysis Symposium*, 5:101–114, June 2000.

[11] Dirk Beyer, Thomas A. Henzinger, Ranjit Jhala, and Rupak Majumdar. The software model checker blast: Applications to software engineering. In *Journal on Software Tools for Technology Transfer (paper from FASE2005)*, 2007.

[12] Sandeep Bhatkar, Abhishek Chaturvedi, and R. Sekar. Dataflow anomaly detection. In *IEEE Symposium on Security and Privacy*, pages 48–62. IEEE Computer Society, 2006.

[13] A. Biere, A. Cimatti, E. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. 58, 2003.

[14] Nikolaj Bjorner, Anca Browne, and Zohar Manna. Automatic generation of invariants and intermediate assertions. *Theor. Comput. Sci.*, 173(1):49–87, 1997.

[15] J. Bochnak, M. Coste, and M. F. Roy. *Real Algebraic Geometry.* Springer, 1998.

[16] Cristina Borralleras, Salvador Lucas, Rafael Navarro-Marset, Enric Rodriguez-Carbonell, and Albert Rubio. Solving non-linear polynomial arithmetic via sat modulo linear arithmetic. *CADE*, pages 294–305, 2009.

[17] Driss Boularas and Abdelkader Chouikrat. Equations d'amorçage d'intégrales premières formelles. In *Linear and Multilinear Algebra*, volume 54, pages 219–233, 2006.

[18] I. A. Browne, Zohar Manna, Henny B. Sipma, Z. Manna, and H. B. Sipma. Generalized temporal verification diagrams. In *In 15th Conference on the Foundations of Software Technology and Theoretical Computer Science*, pages 726–765. Springer-Verlag, 1995.

[19] David Brumley and Ivan Jager. The bap hanbook. In *Technical Report TR*, 2009.

[20] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, 35(8):677–691, 1986.

[21] Bruno Buchberger. Symbolic computation: Computer algebra and logic. In *Frontiers of Combining Systems: Proceedings of the 1st Int. Workshop, Munich (Germany)*, pages 193–220, 1996.

[22] Miguel Castro, Manuel Costa, and Tim Harris. Securing software by enforcing data-flow integrity. In *Proceedings of the Sixth Symposium on Operating Systems Design and Implementation*, November 2006.

[23] Shuo Chen, Jun Xu, Emre C. Sezer, Prachi Gauriar, and Ravisha nkar K. Iyer. Non-control-data attacks are realistic threats. In *USENIX Security Symposium*, 2005.

[24] Yinghua Chen, Bican Xia, Lu Yang, and Naijun Zhan. Generating polynomial invariants with discoverer and qepcad. In *Formal Methods and Hybrid Real-Time Systems*, pages 67–82, 2007.

[25] Michael Christodorescu, Somesh Jha, and Christopher Kruegel. Mining specifications of malicious behavior. In *Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2007.

[26] Mihai Christodorescu, Somesh Jha, Sanjit A. Seshia, Dawn Song, and Randal E. Bryant. Semantics-aware malware detection. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy (Oakland 2005)*, pages 32–46, Oakland, CA, USA, May 2005. ACM Press.

[27] Giovanni L. Ciampaglia and Rachid Rebiha. An ant colony verification algorithm. In *ISDA '07: Proceedings of the Seventh International Conference on Intelligent Systems Design and Applications*, pages 901–906, Washington, DC, USA, 2007. IEEE Computer Society.

[28] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50(5):752–794, 2003.

[29] Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, Cambridge, MA, 2000.

[30] F.B. Cohen. A short courses on computer viruses. In *Wiley,*, 1990.

[31] C. Collberg, C. Thomborson, and D. Low. A taxonomy of obfuscating transformations. Technical Report 148, University of Auckland, Department of Computer Science, Auckland New Zealand, 1997.

[32] G E Collins. *Quantifier Elimination for the Elementary Theory of Real Closed Fields by Cylindrical Algebraic Decomposition.* LNCS, 1975.

[33] Michael A. Colon, Tomás E. Uribe, Michael A. Col'on, and Tom'as E. Uribe. Generating finite-state abstractions of reactive systems using decision procedures. In *in Computer Aided Verification*, pages 293–304. Springer, 1998.

[34] P Cousot. Proving program invariance and termination by parametric abstraction, lagrangian relaxation and semidefinite programming. In *Sixth Int. Conf. on Verification, Model Checking and Abstract Interpretation (VMCAI'05)*, pages 1–24, Paris, France, LNCS 3385, January 17–19 2005.

[35] P Cousot and R Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conf. Record of the 4th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, NY.

[36] P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 13(2–3):103–179, 1992.

[37] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Conference Record of the Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 84–97, Tucson, Arizona, 1978. ACM Press, New York, NY.

[38] D. A. Cox, J. B. Little, and D. O'Shea. *Ideals, varieties, and algorithms: an introduction to computational algebraic geometry and commutative algebra.* Springer, 1997.

[39] CVE. The common vulnerabilities and exposures (cve) web site, 2002. http://cve.mitre.org/.

[40] Mila Dalla Preda, Mihai Christodorescu, Somesh Jha, and Saumya Debray. A semantics-based approach to malware detection. In *POPL '07: Proceedings of the 34th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 377–388, New York, NY, USA, 2007. ACM Press.

[41] E W Dijkstra. *A Discipline of Programming.* Prentince-Hall, 1976.

[42] D. S. Dummit and R. M. Foote. *Anstract Algebra.* Prentice Hall Inc, 1991.

[43] Bruno Dutertre and Leonardo De Moura. A fast linear-arithmetic solver for dpll(t). pages 81–94. Springer, 2006.

[44] Bruno Dutertre and Leonardo De Moura. The yices smt solver. Technical report, 2006.

[45] Michael D. Ernst, Jeff H. Perkins, Philip J. Guo, Stephen McCamant, Carlos Pacheco, Matthew S. Tschantz, and Chen Xiao. The Daikon system for dynamic detection of likely invariants. *Science of Computer Programming*, 69(1–3):35–45, December 2007.

[46] Jean-Charles Faugere. A new efficient algorithm for computing grobner bases (f4). *Journal of Pure and Applied Algebra*, 139(1-3):61–88, June 1999.

[47] Henry Hanping Feng, Jonathon T. Giffin, Yong Huang, Somesh Jha, Wenke Lee, and Barton P. Miller. Formalizing sensitivity in static analysis for intrusion detection. In *IEEE Symposium on Security and Privacy*, page 194. IEEE Computer Society, 2004.

[48] Jean-Claude Fernandez, Hubert Garavel, Alain Kerbrat, Laurent Mounier, Radu Mateescu, and Mihaela Sighireanu. Cadp - a protocol validation and verification toolbox. In *CAV '96: Proceedings of the 8th International Conference on Computer Aided Verification*, pages 437–440, London, UK, 1996. Springer-Verlag.

[49] C. Fisher. Tremor analysis(pc). In *Virus Diget, 6(88)*, 1993.

[50] R W Floyd. Assigning meanings to programs. In *Proc. 19th Symp.Applied Mathematics*, pages 19–37, 1967.

[51] Stephanie Forrest and Thomas Longstaff. A sense of self for unix processes. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 120–128, May 1996.

[52] Martin Fränzle and Christian Herde. Hysat: An efficient proof engine for bounded model checking of hybrid systems. *Form. Methods Syst. Des.*, 30(3):179–198, 2007.

[53] Martin Fränzle, Christian Herde, Tino Teige, Stefan Ratschan, and Tobias Schubert. Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. *JSAT*, 1(3-4):209–236, 2007.

[54] Steven M German. A program verifier that generates inductive assertions. In *Technical Report TR*, pages 19–74, Center for Research in Computing Technology, Harvard U., 1974.

[55] Jonathon T. Giffin, David Dagon, Somesh Jha, Wenke Lee, and Barton P. Miller. Environment-sensitive intrusion detection. In Alfonso Valdes and Diego Zamboni, editors, *Recent Advances in Intrusion Detection (RAID)*, volume 3858 of *Lecture Notes in Computer Science*, pages 185–206. Springer, 2005.

[56] Jonathon T. Giffin, Somesh Jha, and Barton P. Miller. Automated discovery of mimicry attacks. In Diego Zamboni and Christopher Krügel, editors, *RAID*, volume 4219 of *Lecture Notes in Computer Science*, pages 41–60. Springer, 2006.

[57] Rajeev Gopalakrishna, Eugene H. Spafford, and Jan Vitek. Efficient intrusion detection using automaton inlining. In *IEEE Symposium on Security and Privacy*, pages 18–31. IEEE Computer Society, 2005.

[58] Denis Gopan and Thomas W. Reps. Low-level library analysis and summarization. In Werner Damm and Holger Hermanns, editors, *CAV*, volume 4590 of *Lecture Notes in Computer Science*, pages 68–81. Springer, 2007.

[59] Susanne Graf and Hassen Saïdi. Construction of abstract state graphs with PVS. In Orna Grumberg, editor, *Computer Aided Verification*, volume 1254 of *LNCS*, pages 72–83, Haifa, Israel, June 1997. springer.

[60] S. Gulwani and A. Tiwari. Assertion checking over combined abstraction of linear arithmetic and uninterpreted functions. In P. Sestoft, editor, *European Symp. on Programming, ESOP 2006*, volume 3924 of *LNCS*, pages 279–293, 2006.

[61] S. Gulwani and A. Tiwari. Combining abstract interpreters. In T. Ball, editor, *ACM SIGPLAN Conf. on Programming Language Design and Implementation, PLDI 2006*, pages 376–386, 2006.

[62] Sumit Gulwani and Ashish Tiwari. Assertion checking unified. In *Sixth Int. Conf. on Verification, Model Checking and Abstract Interpretation (VMCAI'05)*, 2005.

[63] Thomas Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS '96)*, pages 278–292, New Brunswick, New Jersey, 1996.

[64] Thomas A. Henzinger and Pei-Hsin Ho. Hytech: The cornell hybrid technology tool. In *Hybrid Systems*, pages 265–293, 1994.

[65] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Grégoire Sutre. Software verification with BLAST. In Thomas Ball and Sriram K. Rajamani, editors, *Model Checking Software, 10th International SPIN Workshop. Portland, OR, USA,*

May 9-10, 2003, Proceedings, volume 2648 of *Lecture Notes in Computer Science*, pages 235–239. Springer, 2003.

[66] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969.

[67] T. Jebelean, L. Kovacs, and N. Popov. Experimental Program Verification in the Theorema System. *Int. Journal on Software Tools for Technology Transfer (STTT)*, 2006. in press.

[68] Ranjit Jhala and Kenneth L. McMillan. Interpolant-based transition relation approximation. In Kousha Etessami and Sriram K. Rajamani, editors, *Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings*, volume 3576 of *Lecture Notes in Computer Science*, pages 39–51. Springer, 2005.

[69] D. Kapur. Automatically generating loop invariants using quantifier elimination. *Proc. IMACS Intl. Conf. on Applications of Computer Algebra*, 2004.

[70] Michael Karr. Affine relationships among variables of a program. *Acta Inf.*, 6:133–151, 1976.

[71] Shmuel Katz and Zohar Manna. A heuristic approach to program verification. In *In the Third International Joint Conference on Artificial Intelligence*, pages 500–512, Stanford, CA, 1973.

[72] L. Kovacs and T. Jebelean. Finding polynomial invariants for imperative loops in the theorema system. In *Proc.of Verify'06 Workshop*, pages 52–67, August 15-16 2006.

[73] Laura Kovacs. Reasoning algebraically about p-solvable loops. In *TACAS 2008: Proc. of the 14th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963, pages 249–264. LNCS, 2008.

[74] A. Kreuzed and L. Robbiano. *Computational commutative algebra.* Springer Verlag, 2005.

[75] Christopher Kruegel, Engin Kirda, Darren Mutz, William Robertson, and Giovanni Vigna. Automating mimicry attacks using static binary analysis. In *Proceedings of the 14th USENIX Security Symposium*, 2005.

[76] Serge Lang. *Algebra.* Springer, January 2002.

[77] Zohar Manna. *Mathematical Theory of Computation.* McGrw-Hill, 1974.

[78] Zohar Manna and Amir Pnueli. *Temporal verification of reactive systems: safety.* Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[79] Nadir Matringe, Arnaldo V. Moura, and Rachid Rebiha. Endomorphism for non-trivial semi-algebraic loop invariant generation. Technical Report TR-IC-08-31, Institute of Computing, University of Campinas, November 2008.

[80] Nadir Matringe, Arnaldo V. Moura, and Rachid Rebiha. Endomorphisms for non-trivial non-linear loop invariant generation. In *5th Int. Conf. Theoretical Aspects of Computing*, pages 425–439. LNCS, 2008.

[81] Nadir Matringe, Arnaldo V. Moura, and Rachid Rebiha. Morphisms for non-trivial non-linear invariant generation for algebraic hybrid systems. Technical Report TR-IC-08-32, Institute of Computing, University of Campinas, November 2008.

[82] Nadir Matringe, Arnaldo V. Moura, and Rachid Rebiha. Generating multivariate formal power series for hybrid systems. Technical report, Institute of Computing, University of Campinas, 2009.

[83] Nadir Matringe, Arnaldo V. Moura, and Rachid Rebiha. Morphisms for non-trivial non-linear invariant generation for algebraic hybrid systems. In *12th Int. Conf. Hybrid Systems: Computation and Control (HSCC2009).* LNCS, 2009.

[84] Nadir Matringe, Arnaldo V. Moura, and Rachid Rebiha. Generatin invariants for non-linear hybrid systems by linear algebraic methods. In *17th Int. Static Analysis Symposium, SAS2010.* LNCS, 2010.

[85] Nadir Matringe, Arnaldo V. Moura, and Rachid Rebiha. Generating formal power series invariant automatically for non-linear hybrid systems. *under submission to the Journal of Symbolic Computation.*, 2010.

[86] Nadir Matringe, Arnaldo V. Moura, and Rachid Rebiha. Multivariate formal power series and transcendental invariant generation for non-linear hybrid systems. In *14th Int. Conf. Hybrid Systems: Computation and Control (HSCC2011), Under submission.* LNCS, 2011.

[87] Nadir Matringe, Arnaldo V. Moura, and Rachid Rebiha. Multivariate formal power series and transcendental invariant generation for non-linear differential and hybrid systems. In *38th Int. Conf. on Current Trends in Theoryand Practice of Computer Science (SOFSEM2012), Under submission.* LNCS, 2012.

[88] Nadir Matringe, Arnaldo Vieira Moura, and Rachid Rebiha. Morphisms for analysis of hybrid systems. In *ACM/IEEE Cyber-Physical Systems CPSWeek'09, Second International Workshop on Numerical Software Verification.(NSV2009) Verification of Cyber-Physical Software Systems*, San Francisco, CA, USA, 2009.

[89] The Coq development team. *The Coq proof assistant reference manual*. LogiCal Project, 2004. Version 8.0.

[90] Gary McGraw and Greg Morrisett. Attacking malicious code: A report to the infosec research council. *IEEE Software*, 17(5):33–41, 2000.

[91] Kenneth L. McMillan. *Symbolic Model Checking*. Kluwer Qcqdemic Publishers, Norwell, MA, USA, 1993.

[92] R. Milner. A calculus of communicating systems. *LNCS*, 92, 1980.

[93] Arnaldo V. Moura and Rachid Rebiha. Formal methods for forensic computer science: Automated malware invariant generation. In *6th International Conference on Forensic Computer Science, ICoFSC'2009 and ICCYBER'2009, "Best paper award".*, 2009.

[94] Arnaldo V. Moura and Rachid Rebiha. Automated malware invariant generation. *Invited submission to the Journal of Forensic Computer Science*, 2010.

[95] Arnaldo V. Moura and Rachid Rebiha. Semantic malware resistance using inductive invariants. *International Journal of Forensic Computer Science (IJoFCS0)*, 5, 2011.

[96] Markus Müller-Olm and Helmut Seidl. Polynomial constants are decidable. In *Static Analysis Symposium*, pages 4–19. LNCS, 2002.

[97] Carey Nachenberg. Computer virus-antivirus co-evolution. *Commun. ACM*, 40(1):46–51, 1997.

[98] George C. Necula. Proof-carrying code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '97, pages 106–119, New York, NY, USA, 1997. ACM.

[99] George C. Necula, Scott McPeak, Shree Prakash Rahul, and Westley Weimer. Cil: Intermediate language and tools for analysis and transformation of c programs. In *CC '02: Proceedings of the 11th International Conference on Compiler Construction*, pages 213–228, London, UK, 2002. Springer-Verlag.

[100] S. Owre, J. M. Rushby, , and N. Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, jun 1992. Springer-Verlag.

[101] Lawrence C. Paulson. *Isabelle - A Generic Theorem Prover (with a contribution by T. Nipkow)*, volume 828 of *Lecture Notes in Computer Science*. Springer, 1994.

[102] K. Pekka and L. Kalle. Ssh1 remote root exploit. http://www.hut.fi/ kalyytik/hacker/ ssh-crc32-exploit_korpinen_lyytikainen, 2002.

[103] C. Piazza, M. Antoniotti, V. Mysore, A. Policriti, F. Winkler, and B. Mishra. *Algorithmic Algebraic Model Checking I: Challenges from Systems Biology*, volume 3576. July 2005.

[104] André Platzer and Edmund M. Clarke. Computing differential invariants of hybrid systems as fixedpoints. In *Computer-Aided Verification, CAV 2008, Princeton, USA, Proceedings*, LNCS. Springer, 2008.

[105] S. Prajna and A. Jadbabaie. Safety verification of hybrid systems using barrier certificates, 2004.

[106] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in cesar. In *Proceedings of the 5th Colloquium on International Symposium on Programming*, pages 337–351, London, UK, 1982. Springer-Verlag.

[107] Nacim Ramdani, Nacim Meslem, and Yves Candau. Reachability of uncertain nonlinear systems using a nonlinear hybridization. In *Hybrid Systems: Computation and Control, HSCC'08*, volume 4981, pages 415–428. LNCS, 2008.

[108] Rachid Rebiha and Hassen Saidi. Quasi-static binary analysis: Guarded model for intrusion detection. In *TR-USI-SRI-11-2006*, November 2006.

[109] E. Rodríguez-Carbonell and D. Kapur. Automatic generation of polynomial invariants of bounded degree using abstract interpretation. *Sci. Comput. Program.*, 64(1):54–75, 2007.

[110] E. Rodríguez-Carbonell and D. Kapur. Generating all polynomial invariants in simple loops. *J. Symb. Comput.*, 42(4):443–476, 2007.

[111] E. Rodriguez-Carbonell and A. Tiwari. Generating polynomial invariants for hybrid systems. In *Hybrid Systems: Computation and Control, HSCC 2005*, volume 3414 of *LNCS*, pages 590–605, March 2005.

[112] John Rushby. Harnessing disruptive innovation in formal verification. In IEEE Computer Society, editor, *Invited paper, 4th IEEE International Conference on Software Engineering and Formal Methods, SEFM 2006*, pages 21–28, Pune, India, 2006.

[113] S. Bensalem, Y. Lakhnech, and H. Saidi. Powerful techniques for the automatic generation of invariants. In Rajeev Alur and Thomas A. Henzinger, editors, *Proc. of the 8th Int. Conf. on Computer Aided Verification CAV*, volume 1102, pages 323–335, NJ, USA, 1996.

[114] S. Gulwani and A. Tiwari. Constraint-based approach for analysis of hybrid systems. In *Proc. of the 14th Int. Conf. on Computer Aided Verification CAV*, 2008.

[115] S. Sankaranarayanan, T. Dang, and F. Ivancic. Symbolic model checking of hybrid systems using template polyhedra. In *14th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems TACAS*, 2008.

[116] Hassen Saidi. Modular and incremental analysis of concurrent software systems. In *ASE '99: Proceedings of the 14th IEEE international conference on Automated software engineering*, page 92, Washington, DC, USA, 1999. IEEE Computer Society.

[117] Hassen Saidi and Natarajan Shankar. Abstract and model check while you prove. In *CAV '99: Proceedings of the 11th International Conference on Computer Aided Verification*, pages 443–454, London, UK, 1999. Springer-Verlag.

[118] S. Sankaranarayanan, H. Sipma, and Z. Manna. Constructing invariants for hybrid system. In *Hybrid Systems: Computation and Control HSCC*, volume 2993 of *LNCS*, pages 539–554. Springer, March 2004.

[119] Sriram Sankaranarayanan. Automatic invariant generation for hybrid systems using ideal fixed points. In *HSCC '10: Proc. of the 13th ACM Int. Conf. on Hybrid systems: computation and control*, pages 221–230. ACM, 2010.

[120] Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. Non-linear loop invariant generation using grobner bases. In *POPL '04: Proc. of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 318–329, New York, NY, USA, 2004. ACM Press.

[121] Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. Constructing invariants for hybrid systems. *Form. Methods Syst. Des.*, 32(1):25–55, 2008.

[122] A. Schrijver. *Theory of Linear and Integer Programming.* Wiley, 1986.

[123] P. Starzetz. Crc32 sshd vulnerability analysis http:// packetstormsecurity.org /0102-exploits/ ssh1.crc32.txt, 2002.

[124] T. A. Henzinger and P. -H. Ho. Algorithmic analysis of nonlinear hybrid systems. In *Proceedings of the 7th International Conference On Computer Aided Verification*, volume 939, pages 225–238, 1995.

[125] A. Tiwari and G. Khanna. Nonlinear systems: Approximating reach sets. In *Hybrid Systems: Computation and Control HSCC*, volume 2993 of *LNCS*, pages 600–614. Springer, March 2004.

[126] Ashish Tiwari, Harald Rueß, Hassen Saïdi, and Natarajan Shankar. A technique for invariant generation. In *TACAS: Proc. of the 7th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, 2001.

[127] C. Tomlin, G. J. Pappas, and S. Sastry. Conflict resolution for air traffic management: a study in multiagent hybrid systems. *Automatic Control, IEEE Transactions on*, 43(4):509–521, 1998.

[128] David Wagner and Drew Dean. Intrusion detection via static analysis. In *IEEE Symposium on Security and Privacy*, pages 156–169, 2001.

[129] David Wagner and Paolo Soto. Mimicry attacks on host-based intrusion detection systems. In Ravi Sandhu, editor, *Proceedings of the 9th ACM Conference on Computer and Communications Security*, Washington, DC, USA, November 2002. ACM Press.

[130] Ben Wegbreit. The synthesis of loop predicates. *Commun. ACM*, 17(2):102–113, 1974.

[131] Volker Weispfenning. Quantifier elimination for real algebra - the quadratic case and beyond. *Applicable Algebra in Engineering, Communication and Computing*, 8(2):85–101, 1997.

[132] Y Xie and Alex Aiken. Saturn: A sat-based tool for bug detection. pages 139–143, 2005.