COMPUTATIONAL EFFICIENCY IN SYMBOLIC OPTIMAL CONTROL

Elisei Macoveiciuc

Vollständiger Abdruck der von der Fakultät für Luft- und Raumfahrttechnik der Universität der Bundeswehr München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

angenommenen Dissertation.

Gutachter/Gutachterin:

1. Prof. Dr. habil. Gunther Reißig
2. Prof. Dr.-Ing. Thomas Moor

Die Dissertation wurde am 30.06.2023 bei der Universität der Bundeswehr München eingereicht und durch die Fakultät für Luft- und Raumfahrttechnik am 11.04.2024 angenommen. Die mündliche Prüfung fand am 16.05.2024 statt.

Contents

List of Tables

List of Figures

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my doctorate supervisor Professor Dr. habil Gunther Reißig for his immense research expertise and his desire to readily provide patient evaluation and support. Additionally, this work would not have been possible without Munich Aerospace scholarship, that financed the research project.

I would also like to thank members of the Institute for Control Engineering (LRT-15) of the University of the Federal Armed Forces Munich - Mister Herman Lex and Miss Elisabeth Loessl and head of the Institute - Professor Dr.-Ing. i.R. Ferdinand Svaricek. Their timely help and professionalism were invaluable during my work.

I am also sincerely grateful to my colleagues, that I had the luck and pleasure to work with - Alexander Weber, Hao Zhou, Mohamed Serry and Victor Cheidde Chaim, for their time and curious discussions.

Lastly, I would be remiss in not mentioning the defense committee, who generously provided knowledge and expertise.

1 Short Summary

Control synthesis is the problem of constructing a strategy to manipulate a given system that induces that system to conform to a declared behavior. Two fundamental control objectives with a wide variety of applications are reach-avoid problem, and the problem of estimation of robust control-invariant set of a given system. Evidently, a large number of solution approaches exist in literature. Majority of available methods, however, possess unattractive drawbacks that motivate further research: suboptimal solutions, inapplicability to uncertain dynamic models, restriction to narrow classes of systems. On the other hand, theoretic advances that allow synthesis of controllers for large classes of uncertain nonlinear systems are impractical due to the need of severe computational effort.

The present work is concerned with providing general computationally-efficient symbolic control methods for highly coupled non-linear systems, without any structural assumptions on system dynamics. Two fundamental specifications are considered: optimal reachability and qualitative invariance. These types of problems often appear in practical scenarios: robotic control, rocket manipulations, path planning of UAVs. The main contributions are : First, novel space-efficient methods are introduced: for optimal control and qualitative invariance respectively. Large reduction of space complexity is proven and no restrictions on continuous dynamics are placed, which makes these algorithms applicable to a wide class of control problems. In fact one source of exponential space complexity is completely removed with no significant increase of time consumption. Guaranteed memory reduction and generality of developed algorithms are the main contributions of this thesis. Up to the knowledge of the author proposed methods are unique to simultaneously prove space relief and take time linear in abstraction size. Moreover, it is demonstrated that controllers synthesized through the use of the newly introduced methods are at least as permissible as controllers synthesized by any standard abstraction-based algorithms. Thus reduction of computational complexity does not produce controllers of inferior quality compared to standard symbolic methods. Second, this work presents a jointly composed software package, ABS, that contains all functionality to construct abstractions of general nonlinear systems and includes the developed synthesis methods. Contrary to many of its competitors, ABS constructs abstractions/ synthesizes controllers fully automatically, with no manual interference needed, aside from constructing a file describing the control problem of interest. Finally, several numerical tests are conducted, which show that the computational savings in practice correspond to reduction of theoretical algorithm complexity. It is shown, that the newly developed algorithms are able to solve control problems that can not be tackled by competing symbolic methods - due to either time or memory constrains.

2 Zusammenfassung

Bei der Kontrollsynthese geht es darum, eine Strategie zur Beeinflussung eines gegebenen Systems zu konstruieren, die dieses System dazu bringt, ein bestimmtes Verhalten an den Tag zu legen. Zwei grundlegende Kontrollziele mit einer Vielzahl von Anwendungen sind das Reichweitenvermeidungsproblem und das Problem der Schätzung der robusten kontrollinvarianten Menge eines gegebenen Systems. In der Literatur gibt es eine große Anzahl von Lösungsansätzen. Die Mehrzahl der verfügbaren Methoden weist jedoch unattraktive Nachteile auf, die zu weiterer Forschung motivieren: suboptimale Lösungen, Unanwendbarkeit auf unsichere dynamische Modelle, Beschränkung auf enge Systemklassen. Andererseits sind theoretische Fortschritte, die die Synthese von Reglern für große Klassen unsicherer nichtlinearer Systeme ermöglichen, aufgrund des hohen Rechenaufwands unpraktisch.

Die vorliegende Arbeit befasst sich mit der Bereitstellung allgemeiner, rechnerisch effizienter symbolischer Kontrollmethoden für hochgradig gekoppelte nichtlineare Systeme, ohne strukturelle Annahmen über die Systemdynamik. Es werden zwei grundlegende Spezifikationen betrachtet: optimale Erreichbarkeit und qualitative Invarianz. Diese Arten von Problemen treten häufig in praktischen Szenarien auf: Robotersteuerung, Raketenmanipulationen, Bahnplanung von UAVs. Die wichtigsten Beiträge sind: Erstens werden neuartige raumeffiziente Methoden eingeführt: für optimale Kontrolle bzw. qualitative Invarianz. Große Reduzierung der Raumkomplexität ist erwiesen, und es gibt keine Einschränkungen hinsichtlich der kontinuierlichen Dynamik, so dass diese Algorithmen auf eine breite Klasse von Steuerungsproblemen anwendbar sind. Tatsächlich wird eine Quelle exponentieller Raumkomplexität vollständig beseitigt, ohne dass sich der Zeitaufwand signifikant erhöht. Die garantierte Speicherreduktion und die Allgemeinheit der entwickelten Algorithmen sind die Hauptbeiträge dieser Arbeit. Nach dem Kenntnisstand des Autors sind die vorgeschlagenen Methoden die einzigen, die gleichzeitig eine räumliche Entlastung beweisen und eine Zeit benötigen, die linear zur Abstraktionsgröße ist. Darüber hinaus wird gezeigt, dass Regler, die mit Hilfe der neu eingeführten Methoden synthetisiert werden Methoden synthetisiert werden, mindestens genauso zulässig sind wie Regler, die mit Hilfe von Standard-Algorithmen auf Abstraktionsbasis synthetisiert werden. Die Verringerung der Berechnungskomplexität führt nicht zu einer schlechteren Qualität der Regler im Vergleich zu symbolischen Standardmethoden. Zweitens wird in dieser Arbeit ein gemeinsam zusammengestelltes Softwarepaket, ABS, vorgestellt, das alle Funktionen zur Konstruktion von Abstraktionen allgemeiner nichtlinearer Systeme enthält und die entwickelten Synthesemethoden einschließt. Im Gegensatz zu vielen seiner Konkurrenten konstruiert ABS die Abstraktionen bzw. synthetisiert die Regler vollautomatisch, ohne dass manuelle Eingriffe erforderlich sind, abgesehen von der Erstellung einer Datei, die das interessierende Regelungsproblem beschreibt. Schließlich werden mehrere numerische Tests durchgeführt, die zeigen, dass die Einsparungen an Rechenzeit in der Praxis der Reduzierung der theoretischen Komplexität des Algorithmus entsprechen. Es wird gezeigt, dass die neu entwickelten Algorithmen in der Lage sind, Steuerungsprobleme zu lösen, die mit konkurrierenden symbolischen Methoden nicht gelöst werden können - entweder aufgrund von Zeit- oder Speicherbeschränkungen.

3 Abstract

Control synthesis is the problem of constructing a strategy to manipulate a given system that induces that system to conform to a declared behavior. Two fundamental control objectives with a wide variety of applications are reach-avoid problem, and the problem of estimation of robust control-invariant set of a given system. A large number of solution approaches, that are now considered classical, have been developed.

Reach - avoid problems, which consist in driving the given system into a target set, possibly within a given time interval and avoiding obstacles, posses two intrinsic issues. First performance of pre-computed feed-forward control strategies can severely degrade in presence of modeling errors or external disturbances in real-world scenarios - actual system trajectory may differ drastically compared to the simulated one. Second, complex obstacle environment make mere derivation of feed-forward trajectory highly nontrivial. First issue was understood to be problematic rather early, with significant efforts put to set analysis of system behavior. Rather than working with a single state trajectory, a sequence of reachable sets can provide valuable insights on the controlled loop behavior in presence of properly modeled disturbance signals. Most results, however, were initially derived for linear systems due to their regular structure. Linear certain systems, for example, allow simple derivation of compactness and convexity results of reachable sets under some regularity assumptions, such as invertibility of state matrix [10]. Formal analysis becomes more difficult for uncertain systems. However, for linear systems under polytopic uncertainty, it is possible to show that convex hull of the reachable sets is propagated recursively [10]. [9] provided necessary and sufficient conditions for existence of a solution of reachability of a target set or a tube by discrete dynamic system for the case where the state can be measured exactly, and sufficient conditions for the case when only disturbed output measurements are available. See [9],[18],[78] for an ellipsoidal estimate of the attainability domain for a linear controlled system and [19] for an algorithm based on polyhedral sets, [29] for hyperplane approximation. Note that all of the above works do not include obstacle environments in their analysis. Development of computational hardware has made tackling of nonlinear problems also possible. One of the most general approaches has traditionally consisted in utilizing continuous optimization to construct a controlled trajectory for physical systems modeled with general non-linear differential equations [32] under the name of Model Predictive Control (MPC). MPC has become a core control method in many areas because of its natural ability to handle state and input constraints as well as complex system dynamics. This approach, however, suffers from a number of intrinsic drawbacks. First, non-linear optimization does not, in general, produce globally optimal solution. Second, derivation of the desired trajectory through non-convex state-space under non-linear dynamics, is a highly challenging problem. Finally, as mentioned above, this approach does not provide any guarantees on the controlled system behavior and, therefore, requires a separate verification phase after the design phase (see, for example [75]). Online divergence from the pre-computed trajectory can be rectified by introducing additional control policies - Tube-based MPC. Tube-based MPC augments classical MPC solution with a controller that attempts to correct deviations and computes a tube that includes all possible closed-loop trajectories. See [66, 3, 91, 43, 63, 99] for nonlinear tube MPC with various optimization methods. These works optimize the tube size around the given trajectory subject to state, input and disturbance bounds. Since optimization problems remain non-linear for general case, tube-based MPC also produces only locally optimal solutions in terms of tube size. All of the mentioned non-linear control methods also disregard obstacles since there exist no efficient optimization algorithms on discontinuous domains.

The second control problem of interest, the invariance problem, consists in computing for a dynamical system a subset of the state space, from which state vector trajectories never escape for indefinite period of time. Guaranteed containment of closed-loop trajectories makes

invariant sets attractive for analysis of safety-critical system behavior with respect to bounded disturbances as well as general controller synthesis - see, for example, domains of attraction for reachability controllers [13]. Invariant sets for linear systems have been actively studied : with recent works focusing on robust invariance [13] and set invariance in the presence of a control action [98, 33, 59, 6, 7]. Approaches for discrete time linear systems include geometric construction using zonotopic bound [101], Linear Matrix Inequality (LMI)- based algorithms [92, 105], Minkowski partial sums-based approximations [72], linear programming [106], and control invariance using convex optimization [83]. Analogues of these methods exist for piecewise affine systems in [108, 84]. General non-linear systems are evidently more difficult to tackle, making works on this subject sparse: [8] - results characterizing invariant sets for discrete-time nonlinear systems and practical examples for the linear case, [23] - invariant sets for dynamics representable as difference of convex functions, [42, 34, 51] - polynomial systems. Notice that all mentioned works require special properties of system dynamics. For general systems, however, no algebraic solution is currently available. Thus, geometric computational approaches have attracted some attention. Most deal with linear and polynomial systems, for example - [55, 82]. Also few computation-oriented results are available for general nonlinear systems [24, 5]. However, most classical approaches discussed above, construct over approximations of invariant sets and thus, can not provide strict mathematical guarantees that trajectories starting from any chosen subset of the computed invariant set would remain inside.

Reliance of classical control methods on different properties of explicit algebraic representation of system dynamics stimulated the demand for formal algorithmic techniques that could reduce analytical complexity of previous approaches and process highly non-linear system dynamics and control objectives in a unified automatic manner. Thus, increasing attention has been drawn to algorithmically determining the global behavior of strongly nonlinear systems. The main idea consists in constructing a finite structure (called abstraction, symbolic model or graph in some works) that approximates behavior of the given continuous system, solve the problem of interest on it and translate the obtained solution back to the continuous domain. Such approach would naturally be able to handle complex obstacle environment, due to its finite nature. One of the earliest works developing this class of methods is [109] providing simplified method of approximating first-order ordinary differential equations by a finite-state system, [36] that replaced continuous state-space by finite set of cells and used cell-to-cell mappings to determine, for example, regions of the state-space where attractors may exist, domains of attraction and boundaries between them. [4] studied classes of hybrid systems that could be translated to purely discrete systems. [73] employs reachable sets to construct cell-to-cell finite maps. See also [97] for modeling of discrete-time systems via stochastic finite approximations. So far, the mentioned works have been mostly theoretical, aimed at general study of complex systems, rather than (approximate) solution of high dimensional problems. Yet, a distinct line of methods needs to be mentioned here, that conforms to the idea of approximate finite analysis of infinite domains and aims at practical problems - Rapidly Exploring Random Trees (RRT) [54] and their derived algorithms - RRT* converging to optimal path [45], RRT+ searching in low-dimensional subspaces [115], A*-RRT, A*-RRT* [14] additionally utilizing geometric graph search, RRT$^{\mathrm{X}}$ [74] algorithm for changing obstacle environments and many more. RRT algorithms try to roughly and rapidly approximate a continuous disconnected domain with a finite graph, usually using MPC to compute trajectories connecting nodes. Construction of a new nodes is iteratively aimed at unvisited areas. RRT methods can not assure efficiency, but are typically fast in practice even for high dimensional systems [17]. Standard RRT, however, possesses the same disadvantage as classical MPC, discussed above - no guarantees of successful solution of continuous problem due to online deviations from computed trajectory.

Formal study of sufficient relations between a system and its abstraction, that could guarantee

solution of the original continuous control problem, continued with [52, 69, 26, 15]. Results concerning linear systems [104, 102] showed that discrete–time controllable linear systems admit symbolic models. Most of the mentioned works were based on appropriately adapting the notion of bi-simulation relation [68, 76] to continuous and hybrid systems. A more general approach is presented in [116, 35, 28, 102], where an approximate version of bi-simulation was considered. Introduction of bi-simulation relations allowed [79] to construct symbolic models for the class of incrementally globally asymptotically stable nonlinear control systems. However, [81] has shown that approximate bi-simulation fails to distinguish between the different role of control inputs and disturbance signals. Consequently, control strategies synthesized on approximately bi-similar symbolic models cannot be transferred to the original systems robustly with respect to disturbance inputs. [81] introduced alternating approximate bi-simulation, which addressed the disturbance issue and allowed for construction of symbolic models for incrementally globally asymptotically stable systems. [119] constructed abstractions for incrementally forward complete systems using alternating approximate simulation relations without any stability assumptions.

The mentioned works, while providing valuable insights into construction of symbolic models, still could not provide the desired general algorithmic approach. First, most works still were applicable only to certain classes of systems. Second, as has been shown later, alternating simulation relations possessed issues regarding the refined controller complexity [88]. Finally, symbolic models in the mentioned works did not take into account any notion of controlled trajectory cost. The aforementioned three issues were resolved in two fundamental works concerning abstraction-based control: [88, 86]. [88] provided framework for construction of abstractions for general non-linear systems which allow synthesizing robust controllers with respect to various disturbances including plant uncertainties, input disturbances and measurement errors. This allows for complete elimination of the controller verification phase. [88] also underlined the problems of controller refinement of symbolic models based on alternating simulation relations, and introduced novel *feedback refinement* system relation resolving the issues, and allowing for refined abstract controllers to be static, while preserving correctness guarantees. [86] expanded the notion of feedback refinement relation to the valued case, including costs associated with the system trajectories and allowing for construction of symbolic models that can be used to solve *optimal control* problems. Moreover, [86] provided a method to compute both upper bounds of the achievable controlled closed-loop performance, and symbolic feedback controllers realizing those bounds. Finally, this work has shown that the computed bounds and the performance of the symbolic controllers converge to the optimal closed-loop performance as the precision parameters of symbolic models approach zero.

To summarize, abstraction-based control methodology has been developed to the point of allowing:

(i) automatic construction of symbolic models for general non-linear systems,

(ii) automatic synthesis of symbolic controllers that can be refined to solve the original control problem;

(iii) computation of mathematically precise bounds on the cost of controlled trajectories;

(iv) guarantees of controller performance and convergence to the optimal performance with increased abstraction precision parameters.

Such attractive theoretic properties of symbolic approach have promising applications - see, for example [11] for a medical glucose control technique based on the use of alternating simulation symbolic models, [67] for temperature regulation in smart buildings, [46] for guaranteed road maneuvering of an autonomous vehicle, optimized routing of vehicle groups [110], delivery

optimization under continuous non-linear constraints [111]. However, industrial application of symbolic control is still severely limited. Success of most of the aforementioned practical examples heavily relies on the fact that their dynamical systems are highly decoupled. This allows for significant computational savings, and therefore, computationally feasible solution of control problems. Yet, many challenging problems posses rather large (subsystems of) highly interconnected dynamics. Abstractions for general non-linear coupled continuous systems require discretization of both state and input spaces and suffer from extreme computational costs since the number of abstract transitions for a discretized state-input pair grow exponentially with the dimension of continuous state-space. Since modern CPUs have reached significant processing power, in practice, memory costs of abstractions have been the main reason why standard algorithms failed on problems with state-space dimension greater than 2.

The present work is concerned with increasing efficiency of abstraction based control for highly coupled general non-linear systems, without any structural assumptions on system dynamics for two specifications - invariance and optimal reachability.

The research to develop more efficient synthesis methods within abstraction-based methodology have followed 3 main approaches: i) algorithms using special properties of classes of continuous systems. The work [118] provided a method to construct abstractions without discretization of the state space, which could potentially be more scalable, yet is limited to stochastic switched systems. Similarly, discretization-free abstractions for incrementally stable switched systems are constructed in [27]. Methods for decoupled systems have shown to be able to cope with systems in higher dimensions, however, this approach is only applicable to plant dynamics that suitably decompose. [85] exploits state- decomposable structure of continuous systems dynamics to drastically reduce abstraction storage costs, while producing controllers with no loss of performance. [48] enables a compositional synthesis approach by employing directed assume–guarantee contracts between monotone systems. [80] constructs abstractions for networks of control systems. Analogously, [71] synthesizes controllers for collections of systems. [65] compositionally abstracts stochastic control systems. [64] presents efficient abstraction methods for weakly interconnected sub-components, possibly connected in feedback, and models the coupling signals as disturbances. [40] decomposes abstraction computation for partially feedback-linearizable systems. [53] describes compositional abstractions for networks of stochastic systems. [12] decentralizes abstraction techniques for multi-agent systems. [117] includes refinement of abstractions for piece-wise affine systems. [70] provides the tool COSYMA applicable to incrementally stable switched systems and uses multi-scaled abstractions. [93] efficiently synthesizes controllers guided by specification for linear systems and safe linear-time temporal logic. [96] presents efficient safety controller synthesis for monotone systems. ii) decomposition algorithms, applicable to very general classes of systems, trying to represent abstractions as composition of lower-dimensional subsystems. A general abstraction-based decomposition algorithm is presented by [67] which provides large reduction at the cost of additional controller conservatism. [49] also provides a general method for modular construction of abstractions but suffers from similar drawback of additional non-determinism. iii) on-the-fly methods: methods that attempt to compute only those parts of symbolic model that are needed to solve the given control task. [44] constructs lazy controller synthesis algorithm, which uses the incremental forward exploration of the symbolic dynamics from initial states and adaptive grids. [44] provides an on-the-fly algorithm to find an invariant set of certain structure. [20, 56, 38] refine grids over the state space during invariance or reachability synthesis. While demonstrating significant improvement on several examples, these methods perform worse memory-wise as a result of stored coarse abstractions, in cases where successful synthesis heavily depends on computations at finest layers, e.g. when system dynamics is heavily disturbed or obstacle environment is complicated. [94, 41] construct abstract transitions during synthesis until a (safety or reachability) control problem is solved. However, all the mentioned approaches

are heuristic in nature and no guarantees on computational reduction are provided. The tool MASCOT [37] builds so-called lazy algorithms, which use several discretization layers of varying coarseness in an on-the-fly fashion. The tool significantly reduces computational effort when only certain limited regions of the state-space present particular difficulties for controller synthesis, as is the case with many problems from robotics where complex obstacles are distributed non-uniformly across the geometrical region. ROCS [56, 57] implements two synthesis modes, one based on precomputed, uniform-grid abstractions, and an on-the-fly mode with specification-guided adaptive state discretization. In the latter mode, symbolic dynamics is computed when needed and not stored, which keeps the memory footprint amazingly small. However, heavy use of interval arithmetic throughout the synthesis process in both modes may considerably increase cpu time compared to implementations relying on standard floating-point arithmetic. The tool applies to an important class of LTL specifications and takes advantage of decomposable plant dynamics for efficiency.

Parallelization approaches that tackle the time consumption of abstraction based algorithms have also been studied: PFACES [47] facilitates parallel controller design and provides methods to parallelize both abstraction computation and discrete synthesis. It supports multiple types of computing units, including CPUs, GPUs, and Hardware Accelerators, as well as their heterogeneous combination. To control memory usage, the tool favors recomputing results over storing them. [112] provides a parallelized algorithm for reach-avoid problems with user control over maximum memory usage. When the memory limit is reached the algorithm requires re-computation of needed parts of abstraction. It is also worth to mention several works that provide optimizations to construction of abstractions: [114] optimizes state-space discretization parameters and [100] reduces the number of abstract transitions through the use of low-level controllers for monotone systems. These two methods apply to every abstract specification and can be combined with the algorithms proposed in this work.

To conclude, abstraction-based synthesis has developed theoretic advantages that are unmatched by classical methods [88, 86]. Promising applications in certain safety-critical environments are expected: medicine [11], smart buildings [67], autonomous vehicles [46]. However, symbolic control is not currently applicable on industrial scale due to large computational costs of abstractions. Most methods that have been developed to address this problem are either restricted to specific classes of systems, or do not provide any guarantees of computational relief.

4 Contributions and Structure of the work

The present work is concerned with providing general computationally-efficient symbolic control methods for highly coupled non-linear systems, without any structural assumptions on system dynamics. Two fundamental specifications are considered: optimal reachability and qualitative invariance. These types of problems often appear in engineering problems: see examples above as well as robot control [50], rocket control [16], Moore-Greitzer engine control [58], path planning of a quad-rotor [121]. The main contributions are :

First, novel space-efficient methods are introduced: for optimal reachability and qualitative invariance respectively. Large reduction of space complexity is proven and no restrictions on continuous dynamics are placed, which makes these algorithms applicable to a wide class of control problems. In fact one source of exponential space complexity is completely removed with no significant increase of time consumption. Guaranteed memory reduction and generality of developed algorithms are the main contributions of this thesis. Up to the knowledge of the author proposed methods are unique to simultaneously prove space relief and take time linear in abstraction size. Moreover, it is demonstrated that controllers synthesized through the use

of the newly introduced methods are at least as permissible as controllers synthesized by any standard abstraction-based algorithms. Thus reduction of computational complexity does not produce controllers of inferior quality compared to standard symbolic methods.

Second, this work presents a jointly composed software package, ABS, that contains all functionality to construct abstractions of general nonlinear systems and includes the developed synthesis methods. Contrary to many of its competitors, ABS constructs abstractions/ synthesizes controllers fully automatically, with no manual interference needed, aside from constructing a file describing the control problem of interest.

Finally, several numerical tests are conducted, which show that the computational savings in practice correspond to reduction of theoretical algorithm complexity. It is shown, that the newly developed algorithms are able to solve control problems that can not be tackled by competing symbolic methods - due to either time or memory constrains.

This work is structured as follows: Section 6 describes method to compute abstractions of nonlinear systems and provides mathematical justification for properties that such abstraction provide. Section 7 contains theoretical contributions of this work and description of novel algorithms. Section 8 presents the developed software package, its functionality, structure and user options. Section 9 concludes the work and includes description of control problems to which the novel algorithms and the competing software have been applied and discussion of the obtained results.

5 Basic definitions and notations

We use the notation from [62], and most of this section is taken from that reference. The relative complement of the set A in the set B is denoted by $B \setminus A$. $\mathbb{R}$, $\mathbb{R}_+$, $\mathbb{Z}$ and $\mathbb{Z}_+$ denote the sets of real numbers, non-negative real numbers, integers and non-negative integers, respectively, and $\mathbb{N} = \mathbb{Z}_+ \setminus \{0\}$. We adopt the convention that $\pm\infty + x = \pm\infty$ for any $x \in \mathbb{R}$. $[a, b]$, $]a, b[$, $[a, b[$, and $]a, b]$ denote closed, open and half-open, respectively, intervals with end points a and b, e.g. $[0, \infty[= \mathbb{R}_+$. $[a; b]$, $]a; b[$, $[a; b[$, and $]a; b]$ stand for discrete intervals, e.g. $[a; b] = [a, b] \cap \mathbb{Z}$, $[1; 4[= \{1, 2, 3\}$, and $[0; 0[= \emptyset$. $\max M$, $\min M$, $\sup M$ and $\inf M$ denote the maximum, the minimum, the supremum and the infimum, respectively, of the nonempty subset $M \subseteq [-\infty, \infty]$. We identify set-valued maps $f\colon A \rightrightarrows B$ with binary relations on $A \times B$, i.e., $(a, b) \in f$ iff $b \in f(a)$. If f is set-valued, then f is *strict* and *single-valued* if $f(a) \neq \emptyset$ and $f(a)$ is a singleton, respectively, for every a. Moreover, single-valued f is identified with an ordinary map $f\colon A \to B$. The restriction of f to a subset $M \subseteq A$ is denoted $f|_M$.

The inverse mapping $f^{-1}\colon B \rightrightarrows A$ is defined by $f^{-1}(b) = \{a \in A \,|\, b \in f(a)\}$, and $f(C) = \bigcup_{a \in C} f(a)$, $C \subseteq A$. For maps $f, g\colon X \to [-\infty, \infty]$, $f < g$ if $f(x) < g(x)$ for all $x \in X$. Analogously, the relations are interpreted component-wise for elements of $[-\infty, \infty]^n$. The set of minimum points of f in some subset $Q \subseteq X$ is denoted $\operatorname{argmin} \{f(x) \,|\, x \in Q\}$.

An extended real-valued function $f\colon A \to \mathbb{R} \cup \{\infty\}$ is called *upper semi − continuous* (u.s.c.) if for every $x \in A$ and $\epsilon > 0$ there is a neighborhood U of x such that for all $x' \in U$ we have $f(x') \leq f(x) + \epsilon$. The set-valued map $H\colon X \rightrightarrows Y$ between metric spaces X and Y is u.s.c. if $H^{-1}(\Omega)$ is closed for every closed subset $\Omega \subseteq Y$, where $H^{-1}(\Omega) = \{x \in X \,|\, H(x) \cap \Omega \neq \emptyset\}$. The hypograph $\operatorname{hypo}(f)$ of a function $f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ is defined by

$$\operatorname{hypo}(f) := \{(x, \gamma) \in \mathbb{R}^n \times \mathbb{R} \,|\, \gamma \leq f(x)\}\,.$$

6 Abstraction-Based Controller Synthesis

This section describes construction of abstractions of dynamical systems. All the necessary computational procedures are first presented. Their mathematical justification follows afterwards

in section 6.3. This section presents a simplified version of concepts introduced in [88, 86]

6.1 Systems and their Finite Approximations

As already mentioned, this thesis is concerned with developing methods for abstraction-based synthesis that guarantee to reduce memory consumption. The novel methods will be implemented and tested against competitors on systems described by ordinary differential inclusions of the form:

$$\dot{x} \in f(x(t), u(t), w(t)) \tag{1}$$

where $x \in X \subset \mathbb{R}^n$ denotes state vector, $u \in U \subset \mathbb{R}^m$ denotes input vector and $w \in W \subset \mathbb{R}^n$ denotes unknown disturbance signal. Systems of the form (1) are of interest since they describe a wide variety of control problems in many domains - for example see [11, 67, 46], and section 9 of this work.

To ensure efficiency of algorithms, presented in this thesis, some assumptions, albeit very general, have to be placed on set-valued map f in the right-hand side of (1).

(i) f depends additively on disturbance signal $w(t)$ and moreover $w(t) \in [\![-w, w]\!], w \in \mathbb{R}^n_+$.

(ii) For every initial value $x_0 \in \mathbb{R}^n$ and every constant input $u \in U$ there exists a solution ξ of (1) defined on $\mathbb{R}$ that satisfies $\xi(0) = x_0$. This assumption is satisfied if the map $f(\cdot, u)$ f is continuously differentiable in its first argument.

As will be shown later, construction of symbolic model of (1) involves over-approximation of reachable sets. To ensure their fast computation condition (i) is introduced, while condition (ii) ensures that it is possible to construct symbolic model of a system in every part of the state-space.

Thus, considered systems take the following form:

$$\dot{x} \in f(x(t), u(t)) + [\![-w, w]\!] \tag{2}$$

where $f\colon \mathbb{R}^n \times U \to \mathbb{R}^n$, $w \in \mathbb{R}^n_+$ satisfies assumption (ii), and summation is interpreted as Minkowski set addition [103, p. 74].

According to theory, correctness of novel methods does not depend on assumptions above, provided that functionality to compute maps of certain properties is available - see section 7. However, for systems of the form (2) computation of such maps is efficient, is described here in detail and is thus guaranteed to exist. This section immediately proceeds to providing all algorithmic details of computing finite abstractions that capture behavior of systems of the form (2). Theoretical results that demonstrate that, indeed, the given functionality can be used to synthesize correct-by-design controllers are presented at the end of the chapter. A general scheme of abstraction-based synthesis for systems (2) is presented in Fig. 1.

To begin with, finite abstractions will be computed having the following data at hand:

- $\dot{x} \in f(x(t), u(t)) + [\![-w, w]\!]$
- sampling time $\tau \in \mathbb{Q}_+$
- f is continuously differentiable in x
- $x \in X_1 = [\![x_1, x_2]\!], x_1 = (x_{11}, ..., x_{1n}), x_2 = (x_{21}, ..., x_{2n}) \in \mathbb{R}^n$; $u \in U_1 = [\![u_1, u_2]\!], u_1, u_2 \in \mathbb{R}^m$; $w \in \mathbb{R}_+$

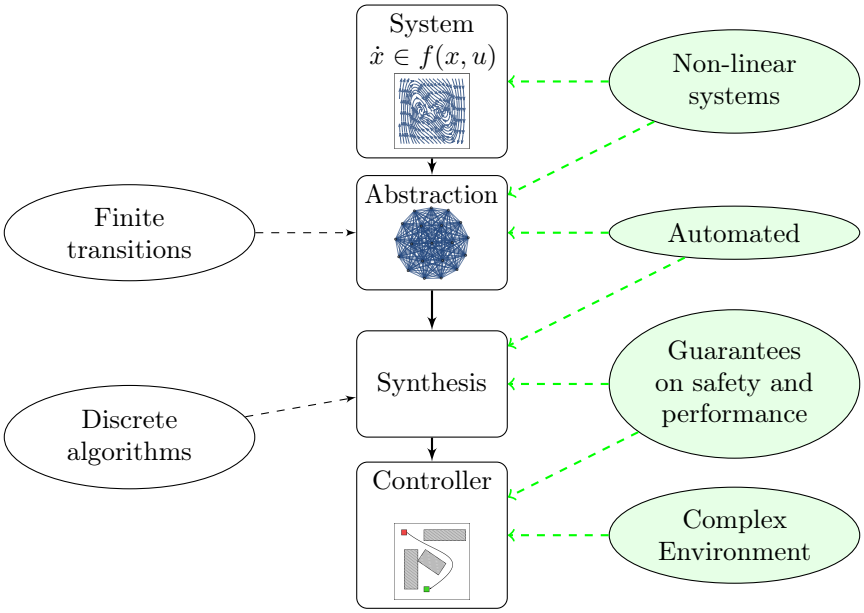


Figure 1: Standard Abstraction-based Synthesis scheme.

- discretization parameters $d^x = (d_1^x, ..., d_n^x) \in \mathbb{N}^n \setminus \{0\}$, $d^u = (d_1^u, ..., d_m^u) \in \mathbb{N}^m \setminus \{0\}$ for X_1 and U_1.

6.1.1 Computation of Abstract State-Space

Abstract state space X_2 consists of hyper-intervals, which also form a cover of the original continuous state space X_1, together with an overflow symbol $\bar{\Omega}$, which is used to model transitions leading to outside of the bounded domain X_1.

$$X_2 = \bigcup_0^{k-1} [\![a_i, b_i]\!] \cup \bar{\Omega} \tag{3}$$

where $[\![a_i, b_i]\!] = \mathbb{R}^n \cap ([a_{i1}, b_{i1}] \times \cdots \times [a_{in}, b_{in}])$, $a_i, b_i \in (\mathbb{R})^n$, $x_1 \leq a_i \leq b_i \leq x_2$, $0 \leq i \leq k-1 < \infty$. Symbol $\bar{\Omega}$ will be used by abstract dynamics to capture cases when continuous system leaves X_1 after one sampling time.

Number of needed hyper-rectangles to cover X depends on discretization parameters, specified in advance and is computed as follows:

$$k = \prod_1^n d_i^x$$

Thus, to every cell $[\![a_i, b_i]\!] \in X_2, 0 \leq i \leq k-1$ corresponds an index i and a vector of coordinates $(c_1, ..., c_n), 0 \leq c_j \leq d_i^x - 1, c_j \in \mathbb{N}\ \forall j \in [1, n]$, computed according to the following subroutines.

To form a uniform cover of the continuous state-space, diameter $r = (r_1, ..., r_n) \in \mathbb{R}^n$ of every cell $[a_i, b_i] \in X_2$ is computed component-wise as follows:

$$r = (x_2 - x_1)/d^x \tag{4}$$

Algorithm 1 Cell/Input Index → Coordinate Vector

Input: Cell/Input index idx
Require: n dimension of continuous state/input space
1: **for** $i \leftarrow 1$ to n **do**
2: $\quad div := 1$
3: $\quad$ **for** $j \leftarrow 1$ to $i - 1$ **do**
4: $\quad\quad div := div \cdot d_j$
5: $\quad modu := div \cdot d_i$
6: $\quad c_i := (idx \textbf{ mod } modu)/div$
Output: Cell/input coordinates $(c_1, ..., c_n)$

Algorithm 2 Coordinate Vector → Cell/Input Index

Input: Cell/Input coordinates $(c_1, ..., c_n)$
Require: n dimension of continuous state/input space
1: $a := 0$
2: $b := 0$
3: $d := 1$
4: **for** $i \leftarrow 1$ to n **do**
5: $\quad a := c_i$
6: $\quad a := a \cdot d$
7: $\quad b := b + a$
8: $\quad d := d \cdot c_i$
9: $idx := b$
Output: Cell/Input index idx

Then, position of a cell $[\![a_i, b_i]\!]$ is determined by procedure that computes geometrical position of its center in the continuous state-space: Algorithm 3.

Algorithm 3 Cell/Input Index $\rightarrow$ Grid Point

Input: Cell/Input index idx
Require: n dimension of continuous state/input space
Require: x_1 lowest point in $[\![x_1, x_2]\!]$ forming state/input continuous space
1: $c :=$ **Cell/Input Index $\rightarrow$ Coordinate Vector**(idx)
2: **for** $i \leftarrow 1$ to n **do**
3: $\quad p_i := x_{1i} + (c_i + 0.5) * r_i$
Output: Cell/Input float coordinates $(p_1, ..., p_n)$

To summarize, algorithms 1,2,3 provide functionality to compute abstract state-space from the original continuous control problem data.

6.1.2 Computation of Abstract Input-Space

The input alphabet U_2 of abstraction is defined as a finite subset of the continuous input set U_1 : $U_2 \subset U_1, |U_2| < \infty$. Number of abstract inputs depends on input discretization parameters, specified in advance and is computed as follows:

$$k_u = \prod_1^m d_i^u$$

Algorithms 1,2,3 can be used, analogously to the previous section, to provide functionality to compute abstract input-space from the original continuous control problem data. The difference between abstract state- and input- spaces consists in the fact that abstract states are subsets of the continuous state-space, and abstract inputs are points in the continuous input-space.

6.1.3 Computation of Abstract Transition Function

To provide functionality to compute abstract transition function F_2, it is first needed to over-approximate reachable sets of continuous systems (2). Let φ denote the general solution of the unperturbed (i.e. letting $w = 0$ in (2)) system associated with f for constant inputs on $[0, \tau]$.Let ξ denote solution of perturbed system (2) on $[0, \tau]$.

Over-approximation method used to construct abstractions has to be computationally efficient, since it will be repeated for every state in the large (although finite) abstract state-space. Therefore, the method used in this work is based on component-wise bounds of neighboring solutions (see also [120, 94]), which in turn leads to simple hyper-interval geometry of the computed set. Thus, component-wise:

$$|\xi(\tau) - \varphi(\tau, p, u)| \leq \beta(|\xi(0) - p|, u) \tag{5}$$

where $\beta \colon \mathbb{R}^n_+ \times U' \rightarrow \mathbb{R}^n_+$, $\xi(0), p \in K \subseteq \mathbb{R}^n, u \in U' \subseteq U$. Function β, called growth bounds is defined as follows:

$$\beta(r, u) = \mathrm{e}^{L(u)\tau}\, r + \int_0^\tau \mathrm{e}^{L(u)s}\, w\, \mathrm{d}s \tag{6}$$

where the following holds, $D_j f_i$ denotes the partial derivative with respect to the jth component of the first argument of f_i:

$$L_{i,j}(u) \geq \begin{cases} D_j f_i(x,u), & \text{if } i = j, \\ |D_j f_i(x,u)|, & \text{otherwise} \end{cases}$$

for $K \subseteq K' \subseteq \mathbb{R}^n$, K' convex, any $u \in U'$, any $\tau' \in [0,\tau]$ and any solution ξ on $[0,\tau']$ of (2) with input u and $\xi(0) \in K$, we have $\xi(t) \in K'$ for all $t \in [0,\tau']$. K' is called apriori enclosure [60]

Algorithm 4 Apriori Enclosure [60]

Input: State domain $X = [\![x_1, x_2]\!]$, Input domain $U = [\![u_1, u_2]\!]$, Sampling time $\tau > 0$, Disturbance bound w, Function f
Require: number of attempts $N < \infty$ pre-defined

$\bar{x} := [\![x_1, x_2]\!]$
$\bar{\bar{x}} := [\![x_1, x_2]\!]$
$x^0 := [\![x_1, x_2]\!]$
$\bar{u} := [\![u_1, u_2]\!]$
$\bar{t} := [0, \tau]$
while $\bar{x} \not\subseteq \bar{\bar{x}} \wedge n < N$ **do**
  $n := n + 1$
  $\bar{x}_f = f(\bar{x}, \bar{u})$ // Interval arithmetic
  $\bar{x}_f := \bar{x}_f + [\![-w, w]\!]$
  if $\bar{x}_f \cap \{-\infty, \infty\} \neq \emptyset$ **then**
    Return Unbound
  $\bar{\bar{x}} := \bar{x}$
  $y := \bar{x}_f \cdot \bar{t}$
  $y := y + x^0$
  $\bar{x} := y$
  if $\bar{x} \cap \{-\infty, \infty\} \neq \emptyset$ **then**
    Return Unbound
if $\bar{x} \not\subseteq \bar{\bar{x}}$ **then**
  Return Not Found

Output: Apriori Enclosure $\bar{x}$

For a cell $[\![a, b]\!] \in X_2$ and input $u \in U_2$ the image of abstract transition function is computed as follows:

$$\begin{gathered} F_2([\![a,b]\!], u) = \{[\![c,d]\!] \in X_2 | \, [\![c,d]\!] \cap \Omega_{([\![a,b]\!],u)} \neq \emptyset\} \\ \text{If } \Omega_{([\![a,b]\!],u)} \cap (\mathbb{R}^n \setminus X_2) \neq \emptyset \wedge \Omega_{([\![a,b]\!],u)} \cap X_2 \neq \emptyset \text{ then } \bar{\Omega} \in F_2([\![a,b]\!], u). \end{gathered} \tag{7}$$

where $\Omega_{([\![a,b]\!],u)} = [\![\varphi(\tau, (a+b)/2, u) - \beta((b-a)/2, u), \varphi(\tau, (a+b)/2, u) + \beta((b-a)/2, u)]\!]$.

It has to be noted that conservativeness of abstraction depends on the radius of abstract cells (assuming a uniform grid) and conservativeness of over-approximation method of reachable sets.

6.2 Abstract and Concrete Control Problems

Algorithms described in this work, focus on two control problems: reach-avoid problem and invariance problem.

A *reachability* specification is defined by:

$$\Sigma_1 = \{(u,x) \in (U \times X)^\infty \, | \, x(0) \in I \Rightarrow \exists t \in [0, T[: (x(t), u(t)) \in Z\} \tag{8}$$

An *invariance* specification on S is defined by:

$$\Sigma_2 = \left\{ (u,x) \in (U \times X)^{[0,\infty[} \,\middle|\, x(0) \in I \Rightarrow \forall t \in [0,\infty[: (x(t),u(t)) \in Z \right\} \tag{9}$$

Current implementation applies only to minimum-time reach-avoid problems: to find a controller for systems (2) that drives the system starting in initial set $I_1 \subseteq X_1$ to target set $T_1 \subseteq X_1$ while avoiding obstacles $O_1 \subseteq X_1$. Implementation of invariance problems is also restricted to qualitative problems only: to find a controller for systems (2) that restricts the system starting in initial set $I_1 \subseteq T_1 \subseteq X_1$ to target set $T_1 \subseteq X_1$ while avoiding obstacles $O_1 \subseteq X_1$ for arbitrary long period of time.

Abstract control problems for both invariance and reachability are defined through sets I_2, O_2, T_2 as follows:

$$I_2 = \{[\![c,d]\!] \in X_2 |\, [\![c,d]\!] \cap I_1 \neq \emptyset\}$$

$$O_2 = \{[\![c,d]\!] \in X_2 |\, [\![c,d]\!] \cap O_1 \neq \emptyset\}$$

$$T_2 = \{[\![c,d]\!] \in X_2 |\, [\![c,d]\!] \subseteq T_1\}$$

6.3 Mathematical Rationale

Section 6.1 provided methods to construct abstraction of a continuous system and define specific control problems on it. This section provides mathematical results stating that abstractions, constructed according to section 6.1 can be used to synthesize correct-by-design controllers for system (2). Efficiency of use of such functionality is discussed in the next chapter.

6.3.1 Feedback Refinement Relations

A general definition of a system is now introduced, at this point without any cost associated with its transitions, and will be used to define relation between the original continuous system and its abstraction, and to prove that algorithms in this work synthesize correct-by-design controllers. Formal language for systems, system relationships and controllers, is used for concise and clear description of the needed mathematical results.

A system is a tuple

$$S = (X, U, F) \tag{10}$$

where X, U, are nonempty sets, and $F \colon X \times U \rightrightarrows X$ is strict and describes evolution of the system.

A tuple $(u,x) \in U^{[0;T[} \times X^{[0;T[}$ is a *solution* of the system (10) (on $[0;T[$, starting at $x(0)$) if $T \in \mathbb{N} \cup \{\infty\}$, and

$$x(t+1) \in F(x(t), u(t)) \tag{11}$$

holds for all $t \in [0; T-1[$ and $x(0) \in X$. For the case of system (2) F would be defined implicitly through solution of an underlying continuous-time control system, i.e. through computation of reachable sets at a certain sampling time.

A controller C for the system (10) (denoted by $C \in \mathcal{F}(X,U)$) is a quintuple

$$(Z, Z_0, \widetilde{X}, \widetilde{U}, H), \tag{12}$$

where Z, Z_0, $\widetilde{X}$, $\widetilde{U}$ are non-empty state, input and output controller alphabets, $Z_0 \subseteq Z$, $X \subseteq \widetilde{X}$, $\widetilde{U} \subseteq U$, and $H\colon Z \times \widetilde{X} \rightrightarrows Z \times \widetilde{U} \times \{0,1\}$ is strict. A quadruple $(u, v, z, x) \in \widetilde{U}^{\mathbb{Z}_+} \times \{0,1\}^{\mathbb{Z}_+} \times Z^{\mathbb{Z}_+} \times \widetilde{X}^{\mathbb{Z}_+}$ is a solution of the controller (12) if $z(0) \in Z_0$ and $(z(t+1), u(t), v(t)) \in H(z(t), x(t))$ holds for all $t \in \mathbb{Z}_+$.

Finally, controlled behavior $\mathcal{B}(C \times S) \subseteq (U \times \{0,1\} \times X)^{\mathbb{Z}_+}$ of the closed loop composed of C (12) and S (10) is defined by the requirement that $(u, v, x) \in \mathcal{B}(C \times S)$ iff there exists a signal $z\colon \mathbb{Z}_+ \to Z$ such that (u, v, z, x) is a solution of C and (u, x) is a solution of S. In addition, the behavior initialized at $p \in X$ is denoted by $\mathcal{B}_p(C \times S)$ and defined by $\mathcal{B}_p(C \times S) = \{(u, v, x) \in \mathcal{B}(C \times S) \,|\, x(0) = p\}$.

To formulate results concerning correctness of abstraction-based solution of control problems, it is needed to define serial and feed-back composition of systems.

6.1 Definition. *Let $S_i = (X_i, U_i, F_i)$ be systems, $i \in \{1, 2\}$, and assume that $X_1 \subseteq U_2$. Then S_1 is* serial composable *with S_2, and the* serial composition *of S_1 and S_2, denoted $S_2 \circ S_1$, is the tuple*

$$(X_{12}, U_1, F_{12}),$$

where $X_{12} = X_1 \times X_2$, $F_{12}\colon X_{12} \times U_1 \rightrightarrows X_{12}$ satisfies

$$F_{12}(x, u_1) = \{(y_1, y_2) | y_1 \in F_1(x_1, u_1) \wedge y_2 \in F_2(x_2, y_1)\}. \tag{13}$$

6.2 Definition. *Let $S_i = (X_i, U_i, F_i)$ be systems, $i \in \{1, 2\}$, $X_2 \subseteq U_1$ and $X_1 \subseteq U_2$, and that the following condition holds:*

(Z) If $x_2 \in U_1$, $x_1 \in U_2$ and $F_2(x_2, x_1) = \emptyset$, then $F_1(x_1, x_2) = \emptyset$.

Then S_1 is feedback composable *with S_2, and the* closed loop *composed of S_1 and S_2, denoted $S_1 \times S_2$, is the tuple*

$$(X_{12}, \{0\}, F_{12}),$$

where $X_{12} = X_1 \times X_2$, and $F_{12}\colon X_{12} \times \{0\} \rightrightarrows X_{12}$ satisfies

$$F_{12}(x, 0) = \{(y_1, y_2) | y_1 \in F_1(x_1, x_2) \wedge y_2 \in F_2(x_2, x_1)\}. \tag{14}$$

At this point all needed mathematical concepts have been introduced to describe how abstractions for systems (2) have to be constructed to ensure that finite abstract controllers, can be utilized to provably solve control problems with countably infinite state and input spaces. In this scope, special system relations are presented with the required property: if abstraction relates to the original plant accordingly and a controller solves abstract control problem, then refined controller solves the original continuous problem.

6.3 Definition. *Let S_1 and S_2 be systems,*

$$S_i = (X_i, U_i, F_i) \tag{15}$$

for $i \in \{1, 2\}$. A strict relation $Q \subseteq X_1 \times X_2$ is a feedback refinement relation *from S_1 to S_2 if the following holds for all $(x_1, x_2) \in Q$:*

(i) $U_2 \subseteq U_1$;

(ii) $Q(F_1(x_1, u)) \subseteq F_2(x_2, u)$.

Existence of a specific feedback refinement relation Q from S_1 to S_2 is denoted $S_1 \preccurlyeq_Q S_2$, and $S_1 \preccurlyeq S_2$ denotes existence of some unspecified feedback refinement relation.

The main property of feedback refinement relations can now be presented. Assume S_1, S_2 and C is the plant, its abstraction and abstract controller respectively. The next theorem states that behavior of abstraction captures behavior of the original system.

Theorem Quote 1 ([88]). Let Q be a feedback refinement relation from the system S_1 to the system S_2, and assume that the system C is feedback composable with S_2. Then the following holds.

(i) C is feedback composable with $Q \circ S_1$, and $C \circ Q$ is feedback composable with S_1.

(ii) $\mathcal{B}(C \times (Q \circ S_1)) \subseteq \mathcal{B}(C \times S_2)$.

(iii) For every $(u, x_1) \in \mathcal{B}((C \circ Q) \times S_1)$ there exists a map x_2 such that $(u, x_2) \in \mathcal{B}(C \times S_2)$ and $(x_1(t), x_2(t)) \in Q$ for all t in the domain of x_1.

It is now needed to describe the process of refinement of abstract controller, solving abstract control problem, to concrete controller, solving the original continuous control problem. A control problem needs to be formally defined, as well as its abstraction.

6.4 Definition. *Let S denote the system (10). Given a set Z, any subset $\Sigma \subseteq Z^\infty$ is called a* specification on Z. *A system S is said to* satisfy *a specification Σ on $U \times Y$ if $\mathcal{B}(S) \subseteq \Sigma$. Given a specification Σ on $U \times Y$, the system C* solves the control problem *(S, Σ) if C is feedback composable with S and the closed loop $C \times S$ satisfies Σ.*

6.5 Definition. *Let S_1 and S_2 be simple systems of the form (10), let Σ_1 be a specification on $U_1 \times X_1$, and let $Q \subseteq X_1 \times X_2$ be a strict relation. A specification Σ_2 on $U_2 \times X_2$ is called an* abstract specification *associated with S_1, S_2, Q and Σ_1, if the following condition holds.*
If $(u, x_2) \in \Sigma_2$, where x_2 and u are defined on $[0; T[$ for some $T \in \mathbb{N} \cup \{\infty\}$, and if $x_1 \colon [0; T[\to X_1$ satisfies $(x_1(t), x_2(t)) \in Q$ for all $t \in [0; T[$, then $(u, x_1) \in \Sigma_1$.

For the sake of simplicity, $(S_1, \Sigma_1) \preccurlyeq_Q (S_2, \Sigma_2)$ whenever $S_1 \preccurlyeq_Q S_2$ and Σ_2 is an abstract specification associated with S_1, S_2, Q and Σ_1 The following fundamental result states that, solution of abstract problem will induces a solution of the concrete problem and describes the controller refinement procedure.

Theorem Quote 2 ([88]). If $(S_1, \Sigma_1) \preccurlyeq_Q (S_2, \Sigma_2)$ and the abstract controller C solves the control problem (S_2, Σ_2), then the refined controller $C \circ Q$ solves the control problem (S_1, Σ_1).

It has been shown that abstractions, preserving feedback refinement relations to original continuous control problem, can be used to solve it. What remains to be shown, is that computational methods presented in section 6.1 indeed produce abstractions that preserve feed back refinement relations to systems (2).

Theorem Quote 3 ([88])**.** Consider two simple systems $S_1 = (X_1, U_1, F_1)$ and $S_2 = (X_2, U_2, F_2)$ of the form (10). Given a sampling time $\tau > 0$, the transition function F_1 is defined by the requirement that $y \in F_1(x, u)$ iff there exists a solution $\xi\colon [0, \tau] \to \mathbb{R}^n$ of (2), with constant input $u \in U_1$, satisfying $\xi(0) = x$ and $\xi(\tau) = y$. Hence, S_1 represents the time-sampled behavior of (2). Let $x \in X_1 = [\![x_1, x_2]\!], x_1 = (x_{11}, ..., x_{1n}), x_2 = (x_{21}, ..., x_{2n}) \in \mathbb{R}^n$; $u \in U_1 = [\![u_1, u_2]\!], u_1, u_2 \in \mathbb{R}^m$; $w \in \mathbb{R}_+$. Set $\bar{X}_2 \subseteq X_2$. Let β be a growth bound on $\cup_{x_2 \in \bar{X}_2} x_2$, U_2 associated with τ and (2). If

(i) X_2 is a cover of X_1 by non-empty, closed hyper-intervals and every element $x_2 \in \bar{X}_2$ is compact;

(ii) $U_2 \subseteq U_1$;

(iii) for $x_2 \in \bar{X}_2$, $x_2' \in X_2$ and $u \in U_2$ we have

$$(\varphi(\tau, c, u) + [\![-r', r']\!]) \cap x_2' \neq \emptyset \Rightarrow x_2' \in F_2(x_2, u), \tag{16}$$

where $[\![a, b]\!] = x_2$, $c = \frac{b+a}{2}$, $r = \frac{b-a}{2}$ and $r' = \beta(r, u)$;

(iv) $F_2(x_2, u) = \emptyset$ whenever $x_2 \in X_2 \setminus \bar{X}_2$, $u \in U_2$.

, then $S_1 \preccurlyeq_\in S_2$.

6.3.2 Optimal Control Problems and their Symbolic Solutions

Section 6.3.1 presented first fundamental advantage of abstraction-based synthesis: abstract controllers can be refined to provably solve the original continuous control problem. However, most practical application demand taking into account trajectory costs when synthesizing controllers. This section presents the second property, unmatched by competing methods: abstract value function of an optimal control problem can approximate concrete value function with arbitrary precision. Analogously to the previous section, an optimal control problem will now be formally defined. The problem data is extended to include non-negative, extended real-valued *running* and *terminal cost functions*, g and G,

$$g\colon X \times X \times U \to \mathbb{R}_+ \cup \{\infty\}, \tag{17a}$$

$$G\colon X \to \mathbb{R}_+ \cup \{\infty\}, \tag{17b}$$

where $\mathbb{R}_+$ denotes the set of non-negative reals.

An *optimal reach-avoid* problem for system (10) $S = (X, U, F)$ is an optimal control problem that requires finding controllers $C \in \mathcal{F}(X, U)$, which, for every state $p \in X$, (approximately) minimize the total cost $J_1\colon (U \times \{0, 1\} \times X)^{\mathbb{Z}_+} \to [0, \infty]$ defined by

$$J_1(u, v, x) = G(x(T)) + \sum_{t=0}^{T-1} g(x(t), x(t+1), u(t)), \tag{18}$$

if $v \neq 0$ and $T = \min v^{-1}(1)$, and otherwise

$$J_1(u, v, x) = \infty, \tag{19}$$

for $(u, v, x) \in \mathcal{B}_p(C \times S)$, in a worst-case sense. Here, the *stopping signal* $v\colon \mathbb{Z}_+ \to \{0, 1\}$ determines the time when the evaluation of the closed loop stops.

An *optimal invariance* problem for S is an optimal control problem that requires finding controllers $C \in \mathcal{F}(X,U)$, which, for every state $p \in X$, (approximately) minimize the total cost $J_2 \colon (U \times \{0\} \times X)^{\mathbb{Z}_+} \to [0,\infty]$ defined by

$$J_2(u,v,x) = \sum_{t=0}^{\infty} g(x(t), x(t+1), u(t)), \tag{20}$$

for $(u,v,x) \in \mathcal{B}_p(C \times S)$, in a worst-case sense. Here, the evolution of the closed loop must never stop, and the stopping signal v is kept only to allow for a unified formulation of both problems.

More formally, for both the reach-avoid ($i = 1$) and the invariance ($i = 2$) problem, the *closed-loop performance* $L_i \colon X \to [0,\infty]$ associated with the controller $C \in \mathcal{F}(X,U)$ is given by

$$L_i(p) = \sup_{(u,v,x) \in \mathcal{B}_p(C \times S)} J_i(u,v,x), \tag{21}$$

and the *value functions* $V_i \colon X \to [0,\infty]$ is defined by

$$V_i(p) = \inf_{C \in \mathcal{F}(X,U)} \sup_{(u,v,x) \in \mathcal{B}_p(C \times S)} J_i(u,v,x), \tag{22}$$

where $i \in \{1,2\}$.

For the sake of convenience, we will refer to the 5-tupel

$$(X, U, F, G, g) \tag{23}$$

as the optimal control problem for the system (10) and cost functions G and g as defined above.

6.6 Example. [86] Reach-avoid problems comprise, e.g. shortest path and minimum-time problems. Moreover, the requirement that the state of S should be driven into a target set $D \subseteq X$ while avoiding an obstacle set $M \subseteq X$ is represented by the following qualitative reach-avoid problem: $G(p) = 0$ if $p \in D \setminus M$, and otherwise $G(p) = \infty$, and

$$g(p,q,u) = \begin{cases} 0, & \text{if } p \notin M \\ \infty, & \text{otherwise.} \end{cases} \tag{24}$$

On the other hand, consider the requirement that the state of S should remain in $D \subseteq X$ indefinitely while avoiding $M \subseteq X$. The requirement is represented by the following qualitative invariance problem, and $V_2^{-1}(0)$ equals the maximal controlled invariant subset of $D \setminus M$: $G = \infty$ and

$$g(p,q,u) = \begin{cases} 0, & \text{if } p \in D \setminus M \\ \infty, & \text{otherwise.} \end{cases} \tag{25}$$

□

Analogously to the previous section, it is needed to define relations which abstract control problems have to preserve to the concrete ones.

6.7 Definition. *Consider two optimal control problems Π_1 and Π_2 of the form (23). The relation $Q \colon X_1 \rightrightarrows X_2$ is a* valuated feedback refinement relation *from Π_1 to Π_2, denoted $\Pi_1 \preccurlyeq_Q \Pi_2$, if Q is strict and the following conditions hold for all $(p_1,p_2),(q_1,q_2) \in Q$ and all $u \in U_2$:*

(i) $U_2 \subseteq U_1$;

(ii) $G_1(p_1) \leq G_2(p_2)$;

(iii) $g_1(p_1, q_1, u) \leq g_2(p_2, q_2, u)$;

(iv) $Q(F_1(p_1, u)) \subseteq F_2(p_2, u)$.

It is now possible to compare abstract and concrete value functions.

Theorem Quote 4 ([86]). Let Π_1 and Π_2 be two optimal control problems with value functions V_1 and V_2, respectively. If $\Pi_1 \preccurlyeq^{\circ}_{Q} \Pi_2$, then $V_1(p_1) \leq V_2(p_2)$ for every $(p_1, p_2) \in Q$.

Results concerning convergence, require the notion of abstraction precision. To provide such definition, it is needed to limit the extent to which U_1, $G_2(\Omega)$, $g_2(\Omega, \Omega', u)$ and $F_2(\Omega, u)$ over-approximate U_2, $\sup G_1(\Omega)$, $\sup g_1(\Omega, \Omega', u)$ and $F_1(\Omega, u)$, respectively.

Let (X, d) be metric space. Then

$$d(x, N) = \inf \{d(x, y) \,|\, y \in N\},$$
$$d(M, N) = \inf \{d(x, y) \,|\, x \in M, y \in N\}$$

$\forall x \in X$, $\forall M, N \subseteq X, M \neq \emptyset, N \neq \emptyset$. $B(c, r)$ and $\bar{B}(c, r)$ denote the open, respectively, closed ball with center $c \in X$ and radius $r > 0$, and $\bar{B}(c, 0) = \{c\}$. Denote the diameter of a subset $M \subseteq X$ by $\operatorname{diam}(M)$ [39].

6.8 Definition. *Let Π_2 be an abstraction of Π_1 and suppose that Π_1 and Π_2 are of the form (23), that U_1 and X_1 are metric spaces, and that the elements of X_2 are closed subsets of X_1.*
Then Π_2 is an abstraction of conservatism ∞ *of Π_1. Moreover, Π_2 is an* abstraction of conservatism $\rho \in \mathbb{R}_+$ *of Π_1 if the following conditions hold for all $\Omega, \Omega' \in X_2$ and all $u \in U_2$:*

(i) $U_1 = \bar{B}(U_2, \rho)$;

(ii) $G_2(\Omega) \leq \rho + \sup G_1(\Omega)$;

(iii) $g_2(\Omega, \Omega', u) \leq \rho + \sup g_1(\Omega, \Omega', u)$.

If Ω satisfies the condition

$$G_1(\Omega) \cup g_1(\Omega, X_1, U_1) \neq \{\infty\}, \tag{26}$$

then additionally require the following:

(iv) $F_2(\Omega, u) \subseteq \{\Omega'' \in X_2 \,|\, d(\Omega'', F_1(\Omega, u)) \leq \rho\}$, *where d denotes the metric on X_1;*

(v) $\operatorname{diam}(\Omega) \leq \rho$.

Thus, the main result of this section can be formulated, where the following notion of convergence is used.

6.9 Definition. *Let the map $V \colon X \to \mathbb{R}_+ \cup \{\infty\}$ be u.s.c. on the metric space X, and let $L_i \colon X \to \mathbb{R}_+ \cup \{\infty\}$ satisfy $L_i \geq V$, for all $i \in \mathbb{N}$. Then the sequence $(L_i)_{i \in \mathbb{N}}$* hypo-converges *to V, denoted $V = \text{h-}\lim_{i \to \infty} L_i$, if the following condition holds. For every $p \in X$ and every $\varepsilon > 0$ there exist a neighborhood $N \subseteq X$ of p such that the inclusion*

$$(N \times \mathbb{R}) \cap \operatorname{hypo} L_i \subseteq B(\operatorname{hypo} V, \varepsilon) \tag{27}$$

holds for all sufficiently large $i \in \mathbb{N}$.

Theorem Quote 5 ([86])**.** Let (X, U, F, G, g) be optimal control problem of the form (23). Assume

(i) X is a proper metric space, U is a compact metric space, F is compact-valued, and g, G and F are u.s.c..

(ii) For every $i \in \mathbb{N}$, Π_i is an abstraction of conservatism $\rho_i \in \mathbb{R}_+ \cup \{\infty\}$ of the form (23), C_i is an optimal controller for Π_i, and L_i is the closed-loop value function of (23) associated with $C_i \circ \in$, where $\in \colon X \rightrightarrows X_i$ is the membership relation and $\lim_{i\to\infty} \rho_i = 0$.

(iii) V is the value function of (23).

Then $\underset{i\to\infty}{\text{h-lim}}\, L_i = V$.

7 Space-Efficient Symbolic Optimal Control

Section 6 demonstrated that it is possible to solve continuous control problems, using only their finite approximations, constructed to preserve certain properties, and provided computational procedures to properly construct such abstractions. Standard abstraction-based algorithms (see for example [86],[95]) operate on abstractions only and completely disregard the original continuous dynamics. Although this behavior induces generality of standard methods, it requires full abstractions to be pre-computed and stored in memory. Abstraction memory requirements for high-dimensional systems are extremely prohibitive and severely limit applications of symbolic control methods.

Uniform and non-uniform grids on continuous state- and input- spaces lead to exponential "explosion" of the number of abstract transitions, since for each abstract state-input pair number of abstract transitions grows exponentially in dimension of continuous state-space. Nevertheless, note that according to procedures described in section 6.1, construction of abstract transition function $F_2(p, u)$ for an abstract state-input pair $(p, u) \in X_2 \times U_2$ does not depend on any $(p, u) \neq (q, v) \in X_2 \times U_2$. As previously mentioned this is exploited by the class of algorithms known as "on-the-fly" methods: to attempt to construct only those parts of abstraction, that are needed to solve the given control problem.

7.1 Optimal Reach-Avoid and Optimal Invariance Synthesis

Two novel on-the-fly methods for synthesis based on abstractions are now presented. Analogous to previous sections, proposed algorithms will be described in the language of formal systems. In all algorithms (X, U, F, G, g) will be assumed to be abstract control problem that is defined implicitly: i.e F is described through a set of functions that compute $F(p, u)$ on demand for every (p, u) - see section 6.1.

Algorithm 5 describes general structure of both optimal reach-avoid and qualitative invariance controller synthesis on abstractions of continuous-state systems. Traversal of finite spaces according to the current iteration minimizer of a value function W (line 6) is a standard approach in graph theory ([21, 86]), however, due to set-valued nature of abstract transition function F, this method can not be naively applied to abstraction-based algorithms. Variables E, Q and Y are subsets of X, $c \colon X \rightrightarrows U$ and $W \colon X \to \mathbb{R}_+ \cup \{\infty\}$. Commands of the form $X \supseteq Q :\supseteq M$ on line 2 require that a set Q satisfying $M \subseteq Q \subseteq X$ is chosen, and similarly for the command on line 6. Iterations

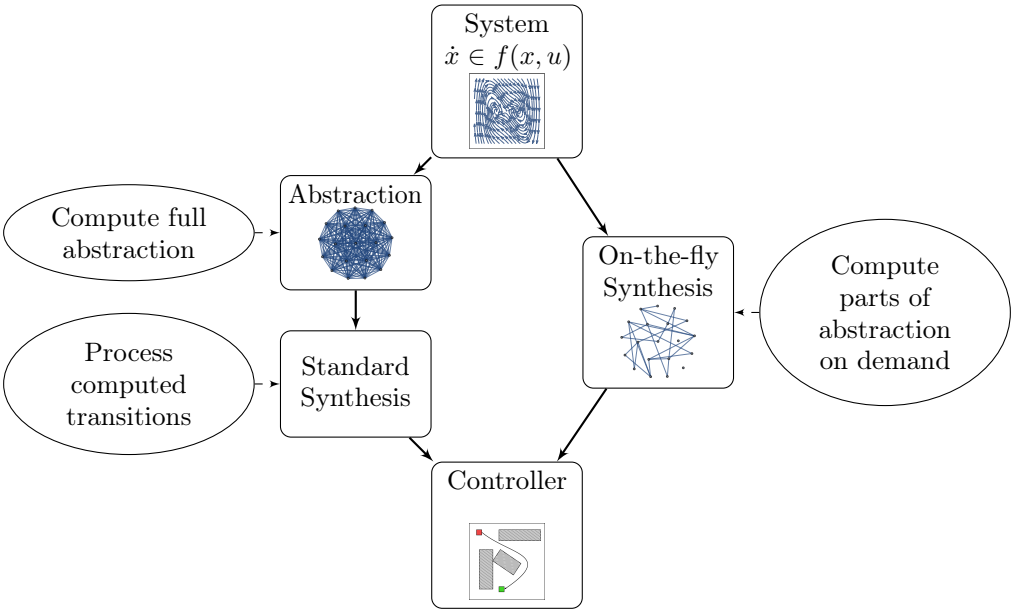


Figure 2: On-the-fly vs Standard Abstraction-based Synthesis scheme.

traverse the state-space, and construct controller c such that $u \in c(x)$ if $F(x,u)$ and E satisfy certain relation depending on control objective.

Synthesis algorithms on abstractions typically progress backwards in time since forward exploration on abstractions is infeasible due to transition map F being set-valued. Therefore, standard methods (see, e.g. [86]) require the inverse of map F to be available. While F is assumed to be given implicitly, F^{-1}, in general, can only be accessed from F stored appropriately in memory which leads to extreme space demands. In contrast, proposed methods in this thesis do not require storage of any significantly large part of F in memory and provide guarantees on large storage reduction.

Assuming absence of pre-computed F^{-1}, Algorithm 5 will require tools to predict which transitions have to be computed. Since these functions will operate on abstractions, they will be described stating their properties first, with actual implementation presented later.

7.1 Definition. *Let $\Pi = (X, U, F, G, g)$ be a control problem. A set-valued map $f^- \colon X \times U \rightrightarrows X$ is called* backward estimator *if*

$$\forall_{(p,u)\in X\times U}\ \exists_{q\in F(p,u)}\ p \in f^-(q,u) \tag{28}$$

7.2 Definition. *Let $\Pi_0 = (X_0, U_0, F_0, G_0, g_0)$, Π be of the form (23) such that $\Pi_0 \preccurlyeq_R \Pi$. A map $F^- \colon X \times U \rightrightarrows X$ is called* backward transition map *associated with Π if the following holds:*

$$\forall_{(x,u,y)\in X\times U\times X} \forall_{x\notin F^-(y,u)} R^{-1}(y) \cap F_0(R^{-1}(x), u) = \emptyset. \tag{29}$$

Map f^- will be used in synthesis of reach-avoid optimal controllers and F^- will be used for synthesis of invariance controllers. Stronger assumptions are placed on F^- than on f^-. This is motivated by the fact that for reach-avoid problems set E (line 3 Alg. 5) iteratively expands and contains only controllable states (i.e. states for which optimal control input has already been found) and functionality of line 9 Alg. 5 consists in finding states that might become controllable

Algorithm 5

Input: Control problem $\Pi = (X, U, F, G, g)$, W_0, c_0
Input: Operator P
Input: Function **ProcessTransitions**
Require: X, U finite
1: $W := P(W_0)$ // W: value function
2: $X \supseteq Q :\supseteq \{x \in X | W(x) \neq W_0(x)\}$ // Q: queue
3: $E := \emptyset$ // E: set of settled states
4: $c := c_0$ // c: controller
5: **while** $Q \neq \emptyset$ **do**
6: $\quad \emptyset \neq Y :\subseteq \operatorname{argmin} \{W(x) \mid x \in Q\}$
7: $\quad Q := Q \setminus Y$
8: $\quad E := E \cup Y$
9: $\quad (Q, c, W) :=$ **ProcessTransitions**(Π, E, Y, Q, c, W)
Output: c, W

later - $x \in X$ such that $\exists_{u \in U} F(x, u) \cap E \neq \emptyset$. Weak properties of f^- are enough to ensure that properly implemented line 9 eventually finds all controllable states (Th. 8). However, in case of invariance problems set E contains only *uncontrollable* states (i.e. states for which an input solving control problem is proven to not exist) and once it is updated (line 8 Alg. 5) functionality of line 9 Alg. 5 consists in immediately capturing all new uncontrollable states - $x \in X$ such that $\forall_{u \in U} F_0(x, u) \cap E \neq \emptyset$. Thus, conditions in Def. 7.2 cannot be weakened. Notice that every backward transition map is also a backward estimator and, therefore, can be used to solve reach-avoid problems as well. Backward estimators are chosen for reach-avoid problems, because their images for a state-input pair are typically smaller than those of backward transition maps.

Backward maps will be computed assuming abstractions for systems of the form (2):

$$\dot{x} \in f(x(t), u(t)) + [\![-w, w]\!]$$
.

where $[\![-w, w]\!]$ contains origin. Let φ denote its solution on $[0, \tau]$. An auxiliary system is introduced:

$$\dot{x} \in -f(x(t), u(t)) + [\![-w, w]\!] \tag{30}$$
.

Backward estimators, and backward transition maps will be constructed by approximating reachable sets of above system. Let $S_1 = (X_1, U_1, F_1)$ be the sampled system associated with the continuous-time system (2) and sampling time $\tau > 0$ and $S_2 = (X_2, U_2, F_2)$ its abstraction constructed according to procedure outlines in section 6.1.

Then backward estimator f^- can be computed as follows:

$$f^-([\![a, b]\!], u) = \{[\![c, d]\!] \in X_2 | \, [\![c, d]\!] \cap \Omega'_{([\![a,b]\!],u)} \neq \emptyset\} \tag{31}$$

where $\Omega'_{([\![a,b]\!],u)} = [\![\varphi(-\tau, (a+b)/2, u) - \beta'((b-a)/2, u), \varphi(-\tau, (a+b)/2, u) + \beta'((b-a)/2, u)]\!]$ and

$$\beta'(r, u) = \mathrm{e}^{L'(u)\tau} \, r \tag{32}$$

$$L'_{i,j}(u) \geq \begin{cases} D_j(-f_i(x,u)), & \text{if } i = j, \\ |D_j(-f_i(x,u))|, & \text{otherwise} \end{cases}$$

is satisfied on apriori enclosure K^- of X_1 and system (30). $D_j(-f_i)$ denotes the partial derivative with respect to the jth component of the first argument of $-f_i$

Backward transition map F^-, on the other hand, requires over-approximation of full reachable set of (30):

$$F^-([\![a,b]\!],u) = \{[\![c,d]\!] \in X_2 |\, [\![c,d]\!] \cap \Omega''_{([\![a,b]\!],u)} \neq \emptyset\} \tag{33}$$

where $\Omega''_{([\![a,b]\!],u)} = [\![\varphi(-\tau,(a+b)/2,u) - \beta''((b-a)/2,u), \varphi(-\tau,(a+b)/2,u) + \beta''((b-a)/2,u)]\!]$ and

$$\beta''(r,u) = \mathrm{e}^{L'(u)\tau}\, r + \int_0^\tau \mathrm{e}^{L'(u)s}\, w\, \mathrm{d}s \tag{34}$$

with L' being defined as above.

Let $[\![a_\varphi, b_\varphi]\!], [\![a,b]\!] \in X_2$ be such that $\varphi(\tau,(a+b)/2,u) \in [\![a_\varphi, b_\varphi]\!]$. It is evident that $[\![a_\varphi, b_\varphi]\!] \in F_2([\![a,b]\!],u)$ and $[\![a,b]\!] \in f^-([\![a_\varphi, b_\varphi]\!],u)$. Thus the map f^- satisfies requirement in Definition 7.1, and is therefore a backward estimator. Moreover, since construction of the backward transition map F^- corresponds to *over-approximating* reachable sets at negative time, F^- satisfies requirements of Definition 7.2.

Let $\Pi = (X,U,F,G,g)$, $W\colon X \to \mathbb{R}_+ \cup \{\infty\}$. Define the following operator P:

$$P(W)(p) = \min\left\{G(p), \inf_{u\in U} \sup_{q\in F(p,u)} g(p,q,u) + W(q)\right\} \tag{35}$$

Function 6 ProcessTransitions: Optimal Reach-avoid

Input: $\Pi = (X,U,F,G,g)$, E, Y, Q, c, W
Require: $\forall_{(x,u)\in X\times U} F_{mem}(x,u) \subseteq F(x,u) \setminus E$ initially
Require: f^- satisfying (28)
1: **for all** $(x,u) : (F_{mem}(x,u) \cap Y \neq \emptyset \vee x \in f^-(Y,u)) \wedge x \notin E$ **do**
2: $F_{mem}(x,u) := \bot(F(x,u) \setminus E)$
3: **if** $F_{mem}(x,u) = \emptyset$ **then**
4: $M := \max\{g(x,z,u) + W(z) \,|\, z \in F(x,u)\}$
5: **if** $W(x) > M$ **then**
6: $W(x) := M$
7: $Q := Q \cup \{x\}$
8: $c(x) := \{u\}$
Output: Q,c,W

Let R be an order on a set S. Then R is called a total order if it is *reflexive*, *transitive*, *anti-symmetric*, and either $(a,b) \in R$ or $(b,a) \in R$ for any two elements $a \neq b$ in S. Let A be a finite set endowed with a total order denoted by $\leq$. We slightly abuse notation by letting $\bot(A)$ denote the (uniquely defined) *singleton set containing the minimum element in* A, i.e., $\bot(A) = \{a' \in A \,|\, \forall_{a\in A} a' \leq a\}$ if $A \neq \emptyset$, and let $\bot(\emptyset) = \emptyset$.

Functions implementing line 9 Algorithm 5 are now presented. Algorithm 5 together with Function 6 solves any optimal reach-avoid problem Π on abstractions. Set E contains controllable

states and to declare a state p controllable all of its successors $F(p,u)$ have to either be controllable themselves or belong to the target set. This condition $F(p,u) \subseteq E$ is verified on line 3 Function 6. Memory reduction is achieved in two ways. First, backward estimator f^- is used on line 1 Function 6 instead of the inverse of abstract transition map F^- to guide synthesis backward in time. Note that $f^-(p,u)$ can be computed on demand for any $(p,u) \in X \times U$ and so f^- requires no storage. This is not the case for F^{-1}. Second, for every (p,u) test of set $F(p,u)$ against E on line 2 Function 6 is performed according to an order and the last successor $q \in F(p,u)$ outside E is stored in $F_{mem}(p,u)$. $F(p,u)$ is a finite set, therefore an order can always be defined. Since set E always expands, states z satisfying $F(p,u) \ni z \leq q \in F_{mem}(p,u) = \{q\}$ before execution of line 2 Function 6 need not be tested upon execution of line 2 Function 6. Thus, Algorithm 5 + Function 6 can be implemented such that any set $F(p,u)$ is traversed at most once. Current implementation of Algorithm 5 uses the following order on $F(p,u)$ for any (p,u). Let $z, q \in F(p,u)$, then

$$\begin{aligned} c^z &= (c_1^z, ..., c_n^z) = \textbf{Algorithm } 1\ (z) \\ c^q &= (c_1^q, ..., c_n^q) = \textbf{Algorithm } 1\ (q) \\ z \leq q &\iff \exists_{i\in[1,n]} \forall_{j<i} c_j^z = c_j^q \wedge c_i^z \leq c_i^q. \end{aligned}$$

Function 7 ProcessTransitions: Qualitative Invariance

Input: $\Pi = (X, U, F, \infty, g)$, E, Y, Q, c, W
Require: F^- satisfying (29)
1: **for all** $y \in Y$ **do**
2: **for all** $(x,u) : x \in F^-(y,u) \wedge x \notin E \cup Q \wedge (c(x) = U \vee c(x) = \{u\})$ **do**
3: **if** $c(x) = \{u\} \wedge y \in F(x,u)$ **then**
4: $c(x) := \bot\{v \in U | u < v \wedge F(x,v) \cap E = \emptyset\}$
5: **else if** $c(x) = U \wedge cardU > 1$ **then**
6: $c(x) := \bot\{v \in U | F(x,v) \cap E = \emptyset\}$
7: **if** $c(x) = \emptyset$ **then**
8: $Q := Q \cup \{x\}$
9: $W(x) = \infty$
Output: Q,c,W

We now turn to solution of invariance problems. Let $\Pi_0 = (X_0, U_0, F_0, \infty, g_0)$ be a continuous-sate invariance problem and $\Pi = (X, U, F, \infty, g)$ its abstraction, where g takes form (25), be input to Algorithm 5 together with Function 7. Set E contains uncontrollable states and in order to declare a state p uncontrollable, at least one of its successors has to be uncontrollable for every abstract control input. This condition $\forall_{u\in U} F(p,u) \cap E \neq \emptyset$ is verified on lines 4,6 Function 7. Algorithm 5 together with Function 7 does not solve Π directly. Instead the novel method solves a qualitative invariance problem $\tilde{\Pi} = (X, U, \tilde{F}, \infty, g)$ with $\tilde{F} \subseteq F$. This behavior is due to the fact that Function 7 utilizes backward transition map F^- instead of inverse map of abstract transition function F^{-1} to guide synthesis backwards in time. Then the following situation is possible after execution of Function 7 at some point in time:

$$\exists_{u\in U} p \notin F^-(E,u) \wedge F(p,u) \cap E \neq \emptyset \tag{36}$$

which implies that lines 4,6 Function 7 have not been invoked for p. Then control symbol u, is not considered unsafe for state p by Function 7 and thus Algorithm 5 together with Function

7 fails to solve Π. However, due to definition of backward transition map F^-, the original continuous system is still guaranteed to not reach unsafe set E starting from $p \subseteq X_0$ under control $u \in U \subseteq U_0$, i.e. $F_0(p,u) \cap E = \emptyset$. Therefore, Algorithm 5 together with Function 7 is still able to provide approximate solution for continuous qualitative invariance problem Π_0, utilizing not abstraction Π but $\tilde{\Pi}$, which is, nevertheless, an abstraction of Π_0 defined by the run of Algorithm 5 together with Function 7 applied to Π. Moreover, $\tilde{F}(p,u)$ disregards exactly set $F(p,u) \cap E$ in situations of type (36). The crucial property here, is that maximal controlled invariant set computed using Algorithm 5 together with Function 7 is not smaller than maximal invariant set for problem Π with no loss of quality of control - see theorem 15.

Analogously to the reach-avoid case, memory reduction is achieved in two ways. First, backward transition map F^- is used instead of the inverse of abstract transition map F^{-1} to guide synthesis backward in time. Note that any part of F^- can be computed on demand and thus requires no storage. Second, for any state p control symbols are verified on lines 4,6 Function 7 according to an order and the last safe input $u : F(p,u) \cap E = \emptyset$ is stored in $c(p)$. Abstract input space U is finite and, therefore, an order can always be defined. Since E always expands, inputs satisfying $U \ni v \leq u \in c(p)$ before lines 4,6 need not be tested on lines 4,6 Function 7. Thus Algorithm 5 together with Function 7 can be implemented such that for every state p every control input u is verified at most once. Current implementation of Algorithm 5 uses the following order on U. Let $u, v \in U$, then

$$\begin{aligned} c^u &= (c_1^u, ..., c_m^u) = \textbf{Algorithm } 1 \ (u) \\ c^v &= (c_1^v, ..., c_m^v) = \textbf{Algorithm } 1 \ (v) \\ v \leq u &\iff \exists_{i \in [1,m]} \forall_{j<i} c_j^v = c_j^u \wedge c_i^v \leq c_i^u. \end{aligned}$$

Methods describing solution of optimal reach-avoid problem and qualitative invariance problems have been introduced. This section is finalized by presenting algorithmic solution to optimal invariance problems. Let $\Pi = (X, U, F, \infty, g)$ be of the form (23) with no restrictions placed on g aside from being non-negative valued. We finally provide a method to solve *optimal* invariance problems in the sense of the value function (22)

$$V_2(p) = \inf_{C \in \mathcal{F}(X,U)} \sup_{(u,v,x) \in \mathcal{B}_p(C \times S)} J_2(u,v,x), \tag{37}$$

where the total cost (20) $J_2 \colon (U \times \{0\} \times X)^{\mathbb{Z}_+} \to [0, \infty]$ defined by

$$J_2(u,v,x) = \sum_{t=0}^{\infty} g(x(t), x(t+1), u(t)), \tag{38}$$

for $(u,v,x) \in \mathcal{B}_p(C \times S)$.

We first make an observation that $0 \leq V_2(p) \leq \infty \forall p \in X$, which follows from non-negativity of cost functions (17). Moreover, notice that $J_2(u,v,x) < \infty$ iff the following condition holds:

$$\exists_{0 \leq t_0 < \infty} \forall_{t > t_0} g(x(t), x(t+1), u(t)) = 0 \tag{39}$$

Thus controller with closed-loop performance equal to value function V_2 drives the system optimally to the set of states $X_0 \subseteq X$, where inputs exist such that the system evolution costs 0, and provably restricts system evolution to X_0, choosing only control inputs with zero transition cost, for indefinite period of time. Therefore the problem of synthesizing controllers with closed-loop performance equal to V_2 reduces to an optimal *reach-and-stay* problem :

(i) Solving qualitative invariance problem over the set of states that allow zero-cost transitions, with additional constraint of controller choosing only zero-cost control inputs, obtaining control invariant subset $I_0 \subseteq X_0$.

(ii) Solving optimal reachability problem towards I_0, $\Pi_r = (X, U, F, G_r, g)$, where

$$G_r(p) = \begin{cases} 0, & \text{if } p \in I_0 \\ \infty, & \text{otherwise.} \end{cases} \tag{40}$$

It has to be noted that computation of zero-cost invariance controllers for $\Pi = (X, U, F, \infty, g)$ is straightforward when full-abstraction is pre-computed - all non-zero cost control inputs are removed and discrete synthesis methods are applied to the modified finite system. This approach has prohibitive memory costs. On the other hand, memory-efficient invariance synthesis method, proposed in this work, can not take into account control cost, due to the fact that transition cost $g(p, q, u)$ depends on successor q of a state p and memory efficiency of the proposed method heavily relies on backward transition maps, sometimes avoiding computation of forward transitions -see (36) for such behavior. However, computation of zero-cost invariance controllers for $\Pi = (X, U, F, \infty, g)$ can be reduced to solution of a sequence of invariance problems without control cost restriction, which can then be solved space-efficiently. Define $\Pi_i, 1 \leq i \leq n < \infty$, where $\Pi_i = (X, U, F, \infty, g_i)$,

$$g_1(p, q, u) = \begin{cases} 0, & \text{if } \exists_{u \in U} g(p, F(p, u), u) = \{0\} \\ \infty, & \text{otherwise.} \end{cases} \tag{41}$$

and $\Pi_{j+1} = (X, U, F, \infty, g_{j+1}), j \geq 1$ defined by

$$D_{j+1} = W_j^{-1}(0) \setminus \left\{ x \in W_j^{-1}(0) \,\middle|\, \nexists_{u \in U} F(x, u) \subseteq W_j^{-1}(0) \wedge g(x, F(x, u), u) = \{0\} \right\} \tag{42}$$

and

$$g_{j+1}(p, q, u) = \begin{cases} 0, & \text{if } p \in D_{j+1} \\ \infty, & \text{otherwise.} \end{cases} \tag{43}$$

where W_j is the value function obtained by applying Algorithm 5 and Function7 to Π_j.

Thus, Algorithm 8 solves optimal invariance problem $\Pi = (X, U, F, \infty, g)$.

Algorithm 8 Optimal Invariance

Input: Control problem $\Pi = (X, U, F, \infty, g)$, c_0,c_0'
Input: Operator P
Require: X, U finite
1: $\Pi_1 := (X, U, F, \infty, g_1)$ // g_1 satisfies (41)
2: $(c_1, W_1) :=$ **Algorithm 5**$(\Pi_1, 0, c_0, P)$ + **Function 7** // c: controller
3: $i := 1$
4: **while True do**
5: $i := i + 1$
6: $\Pi_i := (X, U, F, \infty, g_i)$ // g_i satisfies (43)
7: $(c_i, W_i) :=$ **Algorithm 5**$(\Pi_i, 0, c_0, P)$ + **Function 7** // c: controller
8: **if** $W_{i-1} = W_i$ **then**
9: **Break While**
10: $c_i := \emptyset$
11: **for all** $x \in W_i^{-1}(0)$ **do**
12: $c_i(x) := \left\{u \in U \,\middle|\, F(x,u) \subseteq W_i^{-1}(0) \wedge g(x, F(x,u), u) = \{0\}\right\}$
13: $\Pi_r := (X, U, F, G_r, g)$ // G_r satisfies (40) for $I_0 = W_i^{-1}(0)$
14: $(c_r, W_r) :=$ **Algorithm 5**(Π_r, ∞, c_0', P) + **Function 6**
Output: c_i,c_r W_i,W_r

All necessary computational tools have been presented. Mathematical justification of novel methods proposed in this work will be presented according to the following scheme:

Step 1. Sufficient conditions, under which Algorithm 5 solves reach avoid and, respectively, qualitative invariance problem are formulated. Loop invariants (A_1) and (A_2), Theorems 6 and 10.

Step 2. Two proposed implementations of Algorithm 5 satisfy conditions formulated during step 1 - Theorems 7 and Theorems 11 .

Step 3. Correctness of proposed methods follows from Steps 1 and 2 - Theorems 8, 12.

Step 4. Efficiency of novel methods is proven - Theorems 9,13.

Step 5. Optimal invariance theory is based on the previous results - Proposition 7.4,Theorem 7.5.

7.2 Mathematical Rationale

7.2.1 Optimal Reachability

Next proposition proves that Algorithm 5 solves a finite reach-avoid problem provided that the loop invariant (A_1) holds after every execution of line 9.

(A_1) Loop invariant: Reachability. Let $\Pi = (X, U, F, G, g)$ be of the form (23).

$$E \cap Q = \emptyset \text{ and } W^{-1}(\mathbb{R}) \subseteq E \cup Q \subseteq X \tag{44a}$$
$$L_C \leq W \leq P_E(W) \tag{44b}$$

where

$$P_E(W)(p) = \min\{G(p), W_E(p)\},$$
$$W_E(p) = \inf_{u\in U: F(p,u)\subseteq E} \sup_{q\in F(p,u)} g(p,q,u) + W(q),$$

$\inf \emptyset = \infty$, and L_C is the closed-loop behavior associated with

$$C = (Z, Z, X, U, H), \tag{45}$$

where Z is any singleton set and $H\colon Z \times X \rightrightarrows Z \times U \times \{0,1\}$ is any map satisfying the following condition for all $p \in X$:

$$\emptyset \neq H(Z,p) \subseteq \begin{cases} Z \times U \times \{1\}, & \text{if } c(p) = \emptyset, \\ Z \times c(p) \times \{0\}, & \text{otherwise.} \end{cases} \tag{46}$$

Theorem Quote 6 ([62])**.** Let $\Pi = (X, U, F, G, g)$ be a reach-avoid problem with finite X and U, and let V_1 be the achievable performance associated with Π. Assume P satisfy (35) and let $\forall_{x\in X} c_0(x) = \emptyset$. Let (Π, ∞, c_0, P) be the input to Alg. 5. Then the following holds for Algorithm 5

(i) Conditions (44a)-(44b) hold upon execution of lines 3-4.

(ii) Assume that each call to 9 terminates, and that (44a)-(44b) hold upon every execution of line 9. Then $Y \neq \emptyset$ and $E \cap Y = \emptyset$ after every execution of line 6, Algorithm 5 terminates returning c and W, and $V_1 = W$ upon termination. Moreover, C defined by (45) and (46) is a static controller for (X, U, F) solving Π.

The next two theorems prove correctness of our proposed on-the-fly method to solve reach-avoid problems.

Theorem Quote 7 ([62])**.** Assume hypothesis of Proposition 6. Let line 9 Alg.5 be implemented as Function 6. Then (A_1) holds after every iteration of the while-loop.

Theorem Quote 8 ([62])**.** Algorithm 5 with Function 6 solves any reach-avoid problem $\Pi = (X, U, F, G, g)$ with finite X, U.

Efficiency of the proposed algorithm to solve optimal control problems is now discussed. Let $n = card(X)$,

$$m = \sum_{p\in X} \sum_{u\in U} card(F(p,u))$$
$$m^- = \sum_{p\in X} \sum_{u\in U} card(f^-(p,u))$$

Theorem Quote 9 ([62])**.** Assume hypothesis of Proposition 6. Let $\Pi = (X, U, F, G, g)$ be the input to Algorithm 5. Then there exists implementation of Function 6 such that Algorithm 5 with Function 6 takes $O(n \cdot cardU)$ space, i.e., $F_{mem}(x,u) \leq 1$ for all $(x,u) \in X \times U$, and takes $O(n \log n + m + m^-)$ time.

7.2.2 Qualitative Invariance

Next proposition proves that Algorithm 5 solves a finite invariance problem provided that the loop invariant (A_2) holds after every execution of line 9.

(A_2) Let $\Pi = (X, U, F, G, g)$ be of the form (23) and V_2 be the achievable performance for Π.

$$E \cap Q = \emptyset \wedge W^{-1}(]0, \infty]) \subseteq E \cup Q \subseteq X \tag{47a}$$

$$V_2 \geq W \geq P_Q(W) \tag{47b}$$

where $\sup \emptyset = \infty$,

$$P_Q(W)(p) = \min \{G(p), W_Q(p)\},$$
$$W_Q(p) = \inf_{u \in U} \sup_{q \in F(p,u) \setminus Q} g(p, q, u) + W(q).$$

Let L_C be the closed-loop performance associated with

$$C = (Z, Z, X, U, H), \tag{48}$$

where Z is any singleton set and $H \colon Z \times X \rightrightarrows Z \times U$ is any map satisfying

$$\emptyset \neq H(Z, p) \subseteq \begin{cases} Z \times U \times \{0\}, & \text{if } c(p) = \emptyset \\ Z \times c(p) \times \{0\}, & \text{otherwise} \end{cases} \tag{49}$$

for all $p \in X$ and c - output of Algorithm 5.

Theorem Quote 10 ([62])**.** Let $\Pi = (X, U, F, \infty, g)$ be an invariance problem with finite X and U, and g satisfying (25). Assume P defined as in (35) and let $\forall_{x \in X} c_0(x) = U$. Let $(\Pi, 0, c_0, P)$ be the input to Alg. 5. Then the following holds for Algorithm 5

(i) (A_2) holds for Π upon execution of lines 3-4.

(ii) Assume that each call to line 9 terminates. Assume (47a) holds upon every execution of line 9 then $Y \neq \emptyset$, $E \cap Y = \emptyset$ after every execution of line 6. Moreover, set E is strictly enlarged after every iteration of the while-loop (line 5) and Algorithm 5 terminates returning c and W. Assume exists sequence $(\Pi_i)_{i \in [1;N]}$, $\Pi_i = (X, U, F_i, \infty, g)$, F_i - strict, $\forall_{i,j \in [1;N], i<j} F_i \subseteq F_j \subseteq F$, such that (47b) holds for Π_i upon execution of line 9 on iteration i of the while-loop. Then $\tilde{V}_2 = W$ upon termination, where $\tilde{V}_2$ is the achievable performance (22)associated with $\tilde{\Pi} = \Pi_N$.

(iii) The following relation holds upon execution of lines 3-4:

$$\forall_{x \in W^{-1}(0)} F(x, c(x)) \subseteq (W^{-1}(0) \cup Q) \wedge c(x) \neq \emptyset \tag{50}$$

Assume, in addition that (50) holds for F_i upon execution of line 9 on iteration i of the while-loop. Then C defined by (48) and (49) is a static controller for $(X, U, \tilde{F})$ solving $\tilde{\Pi}$.

To formulate the main results and additional definition is needed. Let $\Pi = (X, U, F, G, g)$, $\Pi_0 \preccurlyeq_R \Pi$. We now define an auxiliary control problem associated with Π,

$$\tilde{\Pi} = (X, U, \tilde{F}, G, g), \tag{51}$$

with $\forall_{(x,u)\in X\times U}\tilde{F}(x,u) \subseteq F(x,u) \wedge \forall_{y\in F(x,u)\setminus\tilde{F}(x,u)}x \notin F^-(y,u)$ and F^- satisfying (29). The following is evident.

7.3 Lemma. *Let Π_0, Π be such that $\Pi_0 \preccurlyeq_R \Pi$ and $\tilde{\Pi}$ be as in (51). Then $\Pi_0 \preccurlyeq_R \tilde{\Pi}$*

The next two theorems prove correctness of our proposed on-the-fly method to solve qualitative invariance problems.

> **Theorem Quote 11** ([62])**.** Assume hypotheses of Proposition 10. Let $\Pi = (X, U, F, \infty, g)$ be the input to the Algorithm 5 and line 9 in Alg. 5 implemented as Function 7. Assume, in addition, Π_0 given such that $\Pi_0 \preccurlyeq_R \Pi$.
> Then each call to Func. 7 terminates and there exists sequence $(\Pi_i)_{i\in[1;N]}$ $\Pi_i = (X, U, F_i, \infty, g)$, of the form (51) F_i - strict, $\forall_{i,j\in[1;N],i<j}F_i \subseteq F_j \subseteq F$, such that ($A_2$) and (50) hold for Π_i upon execution of line 9 in Alg. 5 on iteration i of the while-loop.

> **Theorem Quote 12** ([62])**.** Assume hypotheses of Proposition 10 and Theorem 11. Let $\Pi = (X, U, F, \infty, g)$ be the input to Algorithm 5 and Π_0 be such that $\Pi_0 \preccurlyeq_R \Pi$. Then there exists $\tilde{\Pi}$ satisfying $\Pi_0 \preccurlyeq_R \tilde{\Pi}$ such that Algorithm 5 together with Function 7 solves $\tilde{\Pi}$.

Theorem 12 has proven that the novel method to solve invariance problems on abstractions is correct. The next theorem shows that it is efficient in the sense of guaranteed memory reduction with only geometrical assumptions on the problem data.

Let $n = card(X)$,

$$m = \sum_{p\in X}\sum_{u\in U} card(F(p,u))$$

$$m^- = \sum_{p\in X}\sum_{u\in U} card(F^-(p,u))$$

> **Theorem Quote 13** ([62])**.** Assume hypotheses of Proposition 10 and Theorem 11. Let $\Pi = (X, U, F, \infty, g)$ be the input to Algorithm 5. Additionally assume that
>
> $$\forall_{(x,u)\in X\times U}\exists_{[\![a,b]\!]\subseteq\mathbb{R}^n}F(x,u) = \{x \in X \mid x \cap [\![a,b]\!] \neq \emptyset\}$$
>
> Then there exists implementation of Function 7 such that Algorithm 5 together with Function 7 requires $O(n)$ memory and $O(m + m^-)$ time.

It has been shown that Algorithm 5 solves optimal abstract control problems correctly and efficiently - Theorem 6 and Theorem 9. On the other hand, Algorithm 5 does not solve the input abstract qualitative invariance problem Π (Theorem 10). However, our main objective is to solve the original continuous invariance problem Π_0 for which abstraction Π was defined. It will now be shown, that solving $\tilde{\Pi}$ instead of Π (Theorem 10) does not negatively affect quality of solution of continuous problem Π_0

Theorem Quote 14 ([86])**.** Let Π_0 and Π_1 be two optimal control problems with achievable performances V_1^0 and V_1^1. If $\Pi_0 \preccurlyeq_R \Pi_1$ then $V_1^0(p_0) \leq V_1^1(p_1)$ for every $(p_0, p_1) \in R$

Theorem Quote 15 ([62])**.** Assume hypothesis of Theorem 12. Let $\Pi = (X, U, F, \infty, g)$ be the input to the Algorithm 5, $\Pi_0, \tilde{\Pi}$ be such that $\Pi_0 \preccurlyeq_R \tilde{\Pi}$, $\tilde{\Pi}$ of the form (51) defined by Theorem 11 and $W = \tilde{V}_2$ be the value function in the output of Algorithm 5, where $\tilde{V}_2$ is the achievable performance for $\tilde{\Pi}$,V_2^0 is the achievable performance for Π_0. Let V_2 be the achievable performance for Π. Then

$$V_2^0(x_0) \leq \tilde{V}_2(x) \leq V_2(x) \text{ for every } (x_0, x) \in R$$

The above result implies that the maximal controlled invariant set obtained by solving $\tilde{\Pi}$ (defined by Function 7) is at least as large as the invariant set obtained by solving Π Π which includes an abstraction of a (continuous-state) system constructed using reachable sets at positive sampling time only. For reach-avoid case the novel presented method solves exactly Π.

7.2.3 Optimal Invariance

Let L_C is the closed-loop behavior associated with

$$C = (Z, Z, X, U, H), \tag{52}$$

where Z is any singleton set and $H \colon Z \times X \rightrightarrows Z \times U \times \{0, 1\}$ is any map satisfying the following condition for all $p \in X$:

$$\emptyset \neq H(Z, p) \subseteq \begin{cases} Z \times U \times \{1\}, & \text{if } c_i(p) = c_r(p) = \emptyset, \\ Z \times c_r(p) \times \{0\}, & \text{if } c_r(p) \neq \emptyset \\ Z \times c_i(p) \times \{0\}, & \text{if } c_i(p) \neq \emptyset = c_r(p) \end{cases} \tag{53}$$

7.4 Proposition. *Let $\Pi = (X, U, F, \infty, g)$ be an invariance with finite X and U, and let V_2 be the achievable performance associated with Π. Let $(\Pi, U, \emptyset)$ be input to Algorithm 8. Then Algorithm 8 terminates and $W = \min\{W_i, W_r\} = L_C = V_2$, i.e. C solves Π, where C is defined by (52).*

Proof. First notice that sequence card D_j defined by (42) is strictly decreasing when j increases. Since $\forall_j D_j \subseteq X$ and X is finite, loop on line 4 Algorithm 8 makes at most $N \leq \text{card } X + 1 < \infty$ number of iterations, with $D_{N-1} = D_N = \emptyset$ and $W_{N-1} = W_N = \infty$, where D_{N-1}, D_N defined by (42) and W_{N-1}, W_N output of Algorithm 5 and Function7 applied to Π_{N-1} and Π_N respectively. Lines 2 7,14 of Algorithm 8 terminate by assumption and theorems 10 and 6. Then Algorithm 8 terminates.

Notice that for any i on line 7 Algorithm 8, $V_2^{-1}(0) \subseteq W_i^{-1}(0)$ and by definition of g_i in Π_i $0 = W_i(p) \neq W_{i+1}(p)$ iff $\nexists_{u \in U} F(p, u) \subseteq W_i^{-1}(0) \wedge g(p, F(p, u), u) = \{0\}$. Let N be iteration of the while-loop be such that line 9 Algorithm 8 has been invoked. Then upon execution of line 12 Algorithm 8 $W_N^{-1}(0)$ is the first set of all sets $W_i^{-1}(0), i < N$ that satisfies $\forall_{p \in W_N^{-1}(0)}$ $F(p, c_N(p)) \subseteq W_N^{-1}(0) \wedge g(p, F(p, c_N(p)), c_N(p)) = \{0\}$. Then $V_2^{-1}(0) = W_N^{-1}(0) = L_{c_N}^{-1}(0)$. Note as well that upon execution of line 14 Algorithm 8 c_r minimizes the worst-case control cost of

reaching $W_N^{-1}(0)$ and so $V_2^{-1}(\mathbb{R}_+ \cup \{\infty\} \setminus \{0\}) = W_r^{-1}(\mathbb{R}_+ \cup \{\infty\} \setminus \{0\}) = L_{c_r}^{-1}(\mathbb{R}_+ \cup \{\infty\} \setminus \{0\})$. Then $V_2 = \min\{W_i, W_r\} = L_C$, with C defined by (52), i.e. C solves optimal invariance problem Π in the worst-case sense.

□

Complexity bounds of Algorithm 8 remain to be established. Let $n = card(X)$,

$$m = \sum_{p \in X} \sum_{u \in U} card(F(p,u))$$

$$m^- = \sum_{p \in X} \sum_{u \in U} card(F^-(p,u))$$

$$m'^- = \sum_{p \in X} \sum_{u \in U} card(f^-(p,u))$$

where f^-, F^- satisfy (28), and (29) respectively.

7.5 Theorem. *Assume hypotheses of Theorem 13 and Theorem 9 and Proposition 7.4. Let $\Pi = (X, U, F, \infty, g)$ be the input to Algorithm 8. Then there exists implementation of Algorithm 8 that requires $O(n \cdot card\, U)$ memory and $O(n \cdot (m + m^- + \log n))$ time.*

Proof. Lines 2, 7 of Algorithm 8 take $O(n)$ memory and $O(m + m^-)$ time by Theorem 13. Line 14 of Algorithm 8 takes $O(n \cdot \text{card } U)$ memory and $O(m + m'^- + n \cdot \log n)$ time by Theorem 9. Construction of problems Π_1 on line 1 and Π_i on line 6 takes $O(m)$ time and $O(n)$ space. Construction of problem Π_r on line 13 take $O(n)$ time and space. For- loop on line 11 take $O(m)$ time and $O(n \cdot \text{card } U)$ space. Loop on line 4 Algorithm 8 make at most n iterations by (42) and $m'^- \leq m^-$ by definition. Then Algorithm 8 takes $O(n \cdot \text{card } U)$ memory and $O(n \cdot (m + m^- + \log n))$ time.

□

8 ABS - Formally correct software for symbolic synthesis

Algorithms, discussed in sections 7 and 6 have been implemented in a software package under the name ABS[1] - Abstraction-Based Synthesis - Figure 8. It consists of two main parts: problem compiler part, written in C, and abstraction and synthesis functionality written in Ada. This section will focus on presenting abstraction and synthesis functionally, while for problem compiler part the reader is refereed to [113].

Ada is a high-level programming language that was developed for improved code reliability. A number of safety features can be distinguished:

(i) Run-time operation validity checks and raising of exceptions

(ii) Strong types : CHAR $\neq$ BOOL $\neq$ INT

(iii) User-defined types, operation incompatibility between distinct types (can be forced) for improved security.

(iv) Structuring code in packages: separation of specification and implementation.

[1]The following people have contributed to the software project ABS: Alexander Weber, Gunther Reissig, Hao Zhou, Elisei Macoveiciuc. See also [87, 89]. The source code reproduced in this section is the result of collaborative work and is not necessarily the present author's own work.

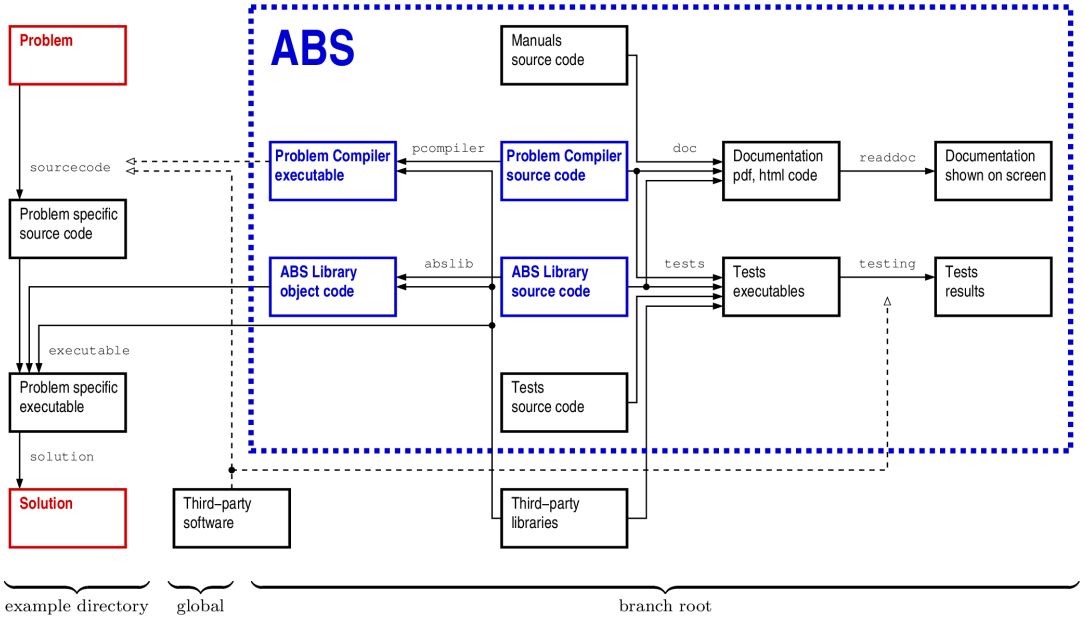


Figure 3: Structure of ABS software [89].

(v) Access types: include classical pointers and additional information for improved security.

Run-time operation validity is especially crucial for implementing software to synthesize correct-by-design controllers: programming errors leading to unexpected value appears during computations are captured during synthesis and do not lead to crashes in controlled loop run. Run-time Ada checks include:

- Access_Check - Check for de-reference of a null pointer.
- Discriminant_Check - Check for access of an unavailable component in a discriminant record (given the discriminant).
- Division_Check - divide by zero.
- Index_Check - Check for out-of-range array index.
- Length_Check - Check for array length violation.
- Overflow_Check - Check for numeric overflow.

Although, run-time checks improve implementation safety, they provide significant time overhead. Run-time checks can be turned off through Ada compiler directives called Pragmas, that are introduced inside the source code, or through compiler switches. For example, gcc switch -gnatp suppresses all run-time checks as though pragma Suppress (all_checks) had been present in the source code.

ABS should be run on x86_64-linux-gnu systems. The following third party software is required:

- `gmp` [31] - version `6.1.2` included in ABS
- `mpfr` [25] - version `3.1.5` included in ABS
- `mpfi` [90] - version `1.5.1` included in ABS
- `gcc`, https://gcc.gnu.org/
- `m4`, https://www.gnu.org/software/m4/
- `flex` [77], https://www.gnu.org/software/flex/
- `bison` [22], https://www.gnu.org/software/bison/
- `boost`, http://www.boost.org/
- `doxygen` [107], https://www.doxygen.org/
- Wolfram Mathematica, https://www.wolfram.com/mathematica/ (Optional)

8.1 Structure and Build process of ABS

ABS is structured as follows:

`ABS/trunk/`	the main branch.
`ABS/branches/`	development.
`ABS/tags/`	development.

Build of ABS is controlled through the tool make (/trunk/Makefile). The main targets are listed below. This table is taken from [89].

`make` or `make all`	Builds the whole of ABS: Produces executable and object files for the third-party libraries and software. Produces these files, each in a debug and a release variant, for Problem Compiler, ABS Library and tests. Builds complete documentation for ABS.
`make pcompiler`	Builds both the debug and the release variants of the Problem Compiler.
`make abslib`	Builds both the debug and the release variants of the ABS Library.
`make tests`	Builds both the debug and the release variants of the tests.
`make testing`	Runs both the debug and the release variants of all tests, or of a subset of tests specified by an optional argument.
`make doc`	Builds complete documentation for ABS.
`make readdoc`	Opens the manual, and the software reference documentation using the browser `firefox`.
`make manuals`	Generates manuals.
`make newproblem`	Generates and initializes a new example directory.
`make debug`	Builds the debug variants of Problem Compiler, ABS Library and tests.
`make release`	Builds the release variants of Problem Compiler, ABS Library and tests.
`make clean`	Produces executable and object files for the third-party libraries and software. Removes all executable and object files of Problem Compiler, ABS Library and tests, as well as software reference documentation files.
`make cleanall`	Performs `make clean`, then removes executable and object files for the third-party libraries and software.

Commands presented above operate on the following directories:

`trunk/doc/`	Documentation source files, as well as files produced using `make doc`.
`trunk/bin/` and `trunk/obj/`	All executables, except for problem executables.
`trunk/body/` and `trunk/spec/`	Source code .
`trunk/libs/`	Third party software, including executable and object files produced through a build process.
`trunk/examples/`	Examples of problems that can be solved..

The build process of ABS is controlled in the following way:

`ABS Library`	using the tool `GPRbuild` [1] with options specified in `trunk/spec/ABSLibrary.gpr`. Compiler flags are specified here.
`Problem Compiler`	using the tool `make` with options specified in `spec/Makefile.ProblemCompiler`
`Controller Synthesis`	using the tool `GPRbuild` and the file `problem_project.gpr` in example directory.

8.2 Software Input and Execution Flow

Entry point of a an executable compiled for a specific problem is function `Main` in `\body\readwrite\main.adb`, which contains the following main data structures (table adapted from [89]):

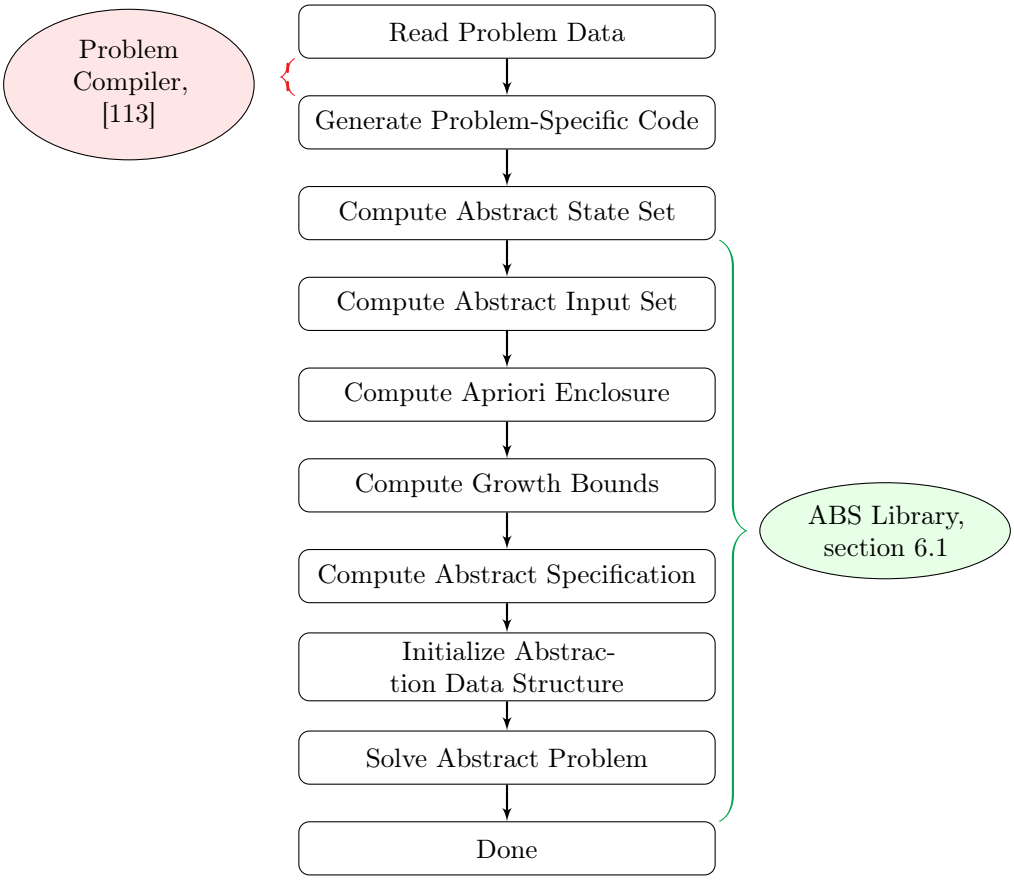


Figure 4: Progression of problem solution.

Identifier in `main.adb`	Description
`P`	Data structure containing problem description information obtained after processing the input file by the Problem Compiler.
`GB`	Data structure for holding the data related to growth bounds.
`GB_b`	Data structure for holding the data related to backward estimators.
`GB_b_w`	Data structure for holding the data related to backward transition maps.
`Cover`	Data structure for representing the cover of the continuous state space.
`Input_Values`	Data structure for representing the input set of the abstraction.
`Abstraction`	Data structure for representing the abstraction.
`Abstract_Specification`	Data structure for representing the abstract specification.
`Solution`	Data structure for representing the solution of the problem.

Problem execution flow is presented in Fig. 4.

8.2.1 User Options in Application to Control Problems

Every problem to be solved by ABS, requires construction of the problem description file in a separate directory, which is processed by the problem compiler. ABS will generate auxiliary and temporary files as well as a file containing the solution in that directory. A number of examples can be found in `trunk/examples/`. To create new example the following steps are needed. The following has been adapted from [89].

1.) Run

```
make newproblem
```

in `trunk/`.

2.) Enter problem name

3.) Enter the path to an existing directory in linux system. (no tilde ~ allowed for pointing to home directory.) In this directory, a subdirectory named as the name of new control problem will be automatically created. All the files related to this control problem (e.g. problem file, binary file) will be contained in that subdirectory. This subdirectory must not exist already

For example:

```
$ make newproblem
Please enter a name for your new control problem: test
Please enter the path to an existing directory: ./examples
Specify your control problem in './examples/test/test.abcs'.
```

New directory contains (The following table is adapted from [89]):

<*pname*>`.abcs`	Problem description, where *pname* coincides with the name of the example directory. Initially empty; problem should be specified using the language described in [87].
`Makefile`	Controls the application of ABS to solve the problem.
`problem_project.gpr`	Controls building problem-specific executable and object files - compiler flags, etc.

The main targets of `Makefile` in the example directory are (The following table is adapted from [89]):

`make` or `make all`	Produces problem-specific source code. Builds executable, object files. Runs executable.
`make sourcecode`	Produces problem-specific source code.
`make executable`	Builds problem-specific executable.
`make solution`	Synonymous with `make all`.
`make clean`	Removes all problem-specific source code, executable, object and results files, including the respective subdirectories.

List of all generated files containing problem-specific code is presented in Tab. 1.

8.2.2 Input Language

The following data has to be specified in a newly created <*pname*>`.abcs`.

Entity	Description	Type
n, m	State and input space dimension	Integer
z	Bound for measurement uncertainty	Vector
X_1	Continuous state-space - section 6.1	Hyper-interval
$\bar{U}$	Continuous input space -section 6.1	Hyper-interval
f	Unperturbed right hand side of (2)	Function
w	Bound for dynamic uncertainties (2)	Vector
τ	Sampling time	Real number
A, E, H	Initial, target, obstacle set, section . 6.2	Union of hyper-intervals
d_1, d_2	State and input space discretization parameter	Vector
p, q	Integration orders for dynamics and growth bounds	Integer

with the help of elementary functions:

$$+ \quad - \quad \times \quad \div \quad \hat{} \quad \text{atan} \quad \cos \quad \cosh \quad \exp \quad \ln \quad \sin \quad \sinh \quad \text{sqrt} \quad \tan$$

Control problem is described as follows, in the same order of data introduced.
First constants are defined:

`Real a = 2*Pi ;`	A constant $a = 2 \cdot \pi$.
`Real gamma in [0.0125,0.0126];`	An uncertain constant gamma $\in [0.0125, 0.0126]$.
`Real v[2];`	A vector $v \in \mathbb{R}^2$.
`v[0] = sqrt(2);v[1] = 2^3;`	$v = (\sqrt{2}, 2^3)$.
`Real M[2][2];`	A matrix $M \in \mathbb{R}^{2\times 2}$.

Second, actual continuous dynamics is introduced. The following is adapted from [87].

```
f: (x,u) in (Real[2],([-2,2],[0,1])) to y[2]
{
y[0] = x[1] ;
y[1] = - sin(x[0]) - cos(x[0])*u[0] - u[1]*x[1] ;
}
```

describes $f\colon \mathbb{R}^2 \times U \to \mathbb{R}^2$ defined through

$$f(x,u) = (x_2, -\sin(x_1) - \cos(x_1) \cdot u_1 - u_2 x_2),$$

where $U = [-2,2] \times [0,1]$.
Note. The dimension n of the state vector appears to the right of `y`) and the dimension m of the input vector is implicitly specified through the hyper-interval $([-2,2],[0,1])$. Input space itself is also specified implicitly through the above hyper-interval.

Sampling time:

```
SamplingTime : 0.3;
```

State space, on the other hand, is specified explicitly :

```
OperatingRange: ([0,2*Pi],[-2,2]);
```

which represents $X_1 = [0, 2\pi] \times [-2, 2] \subset \mathbb{R}^2$.

List of periodic state variables:

```
ListOfPeriodicComponents : (0);
```

Continuous control problem specification:

```
InitialSet : ([0,0],[0,0]);
TargetSet : ([Pi-.1,Pi+.1],[-.1,.1]);
ObstacleSet : ([Pi-.5,Pi-.4],[-.5,-.5]);
```

which represents initial set $A = \{0, 0\}$, target set $T = [\pi - 0.1, \pi + 0.1] \times [-0.1, 0.1]$ and obstacle set $O = [\pi - 0.5, \pi - 0.4] \times \{-0.5, 0.5\}$. Entries representing A, T have to appear at least once, while absence of O is interpreted as $O = \emptyset$.

Note. Union of target sets $T_1 \cup T_2$ is introduced as follows

```
  TargetSet : ([Pi-.3,Pi-.2],[-.3,-.2]);
  TargetSet : ([Pi-.1,Pi+.1],[-.1,.1]);
```

which corresponds to $T_1 = [\pi - 0.3, \pi - 0.2] \times [-0.3, -0.2] \cup T_2 = [\pi - 0.1, \pi + 0.1] \times [-0.1, 0.1]$. Analogously for initial and obstacle sets.

Bound on state measurement errors:

```
BoundOnMeasurementErrors :(0.0125,0.0025/180*Pi);
```

Bound on disturbance vector:

```
BoundOnDynamicUncertainties :(0.01,0.01);
```

to represent $w = [\![-0.01, 0.01]\!]$

Discretization of state space and of input space:

```
InitialStateSpaceDiscretization : (200,200);
InitialInputSpaceDiscretization : (5,5);
```

Note. Input space discretization is incremented by 1 in every dimension by ABS.

Construction of abstraction requires unperturbed solution of solution of (2), see (5), section 6.1. An approximation in terms of Taylor polynomial is used due to nonavailability of exact solution.

```
IntegrationOrder : 8 ;
```

and, similarly, to compute growth bound formula (6)

```
IntegrationOrderGrowthBound : 15;
```

which concludes the ABS input file.

Running `make` in the problem directory will start solution process of the given problem according to Fig. 4. After generation of problem-specific code has been finalized, the user has to choose solution method and the problem itself.

```
$ ------------------------------------------------
------- Abstraction-based Controller Synthesis------
--------For Reach-Avoid && Invariance Problems------
---------------------- 10-2021 ---------------------
------------------------------------------------

Choose problem and solution  type
(0) - On-the-fly solution to the reach avoid problem
(1) - On-the-fly solution to the invariance  problem
(2) - Standard   solution to the reach avoid problem
(3) - Standard   solution to the invariance  problem

```

The available methods are:

`On-the-fly ... reach avoid problem`	Synthesize minimum-time controller to reach the `Target set` avoiding `Obstacle Set`, specified in the input file, use memory efficient algorithm, see section 7. Verify if `Initial set` is controlled
`Standard ... reach avoid problem`	Synthesize minimum-time controller to reach the `Target set` avoiding `Obstacle Set`, specified in the input file, use standard algorithm, see [86],[95]. Verify if `Initial set` is controlled.
`On-the-fly ... invariance problem`	Synthesize invariance controller for simultaneously computed maximal controlled subset of the `Target set` avoiding `Obstacle Set`, specified in the input file, use memory efficient algorithm, see section 7. Verify if `Initial set` is controlled.
`Standard ... invariance problem`	Synthesize invariance controller for simultaneously computed maximal controlled subset of the `Target set` avoiding `Obstacle Set`, specified in the input file, use standard algorithm, see [95]. Verify if `Initial set` is controlled.

Note. Type of problem is specified at this point of execution flow, and not in the software input file, since data describing invariance problems is syntactically similar to reach-avoid problems.
Note. Solution method `On-the-fly ... invariance problem` requires the following, with `A.E.` standing for Apriori enclosure of the `Target Set` introduced in the input file:`Target Set` $\subset$ `A.E` $\subseteq$ `OperatingRange`. The first strict inclusion is assumed and is necessary since successful application of this method requires initial non-empty set of unsafe states, which captures all possible unsafe system behavior - i.e $Q \neq \emptyset$ before first execution of line 5 Algorithm 5. The second inclusion is verified automatically and if it does not hold, software outputs new set that has to be introduced in the input file instead of existing `OperatingRange`.

```
$ ------------------------------------------------
------- Abstraction-based Controller Synthesis------
--------For Reach-Avoid && Invariance Problems------
---------------------- 10-2021 ---------------------
------------------------------------------------

Choose problem and solution  type
(0) - On-the-fly solution to the reach avoid problem
(1) - On-the-fly solution to the invariance  problem
(2) - Standard   solution to the reach avoid problem
```

```
  (3) - Standard   solution to the invariance  problem
  1

  - Apriori Enclosure:
  Done. Time:  6.58000E-04
  - Checking if State Domain contains apriori enclosure of the Target Set
  False.
  Done. Time:  5.02000E-04
  Error: Operating Range in the input file does NOT contain apriori enclosure of
    the Target Set.
  Not Possible to apply the chosen synthesis algorithm.
  Update the input file with the following Operating Range to apply the chosen
    synthesis algorithm:
  OperatingRange:([-8.76627302406191355E-01,7.1598126095857797E
    +0],[-2.92209100802063748E+0,2.92209100802063748E+0]);

```

at this point the software stops. The user has to update the input file and run `make clean` and `make` again.

8.3 Packages and their functionality

As mentioned previously, abstraction and synthesis source code is divided into packages consisting of two files each: specification (located in `/ABS/trunk/spec/`) and body (located in `/ABS/trunk/body/`). ABS packages are now presented and their key functionality is discussed.

8.3.1 Abstraction_I14sym

Package to describe abstractions and functions to initialize and compute them. Different control algorithms require variability of abstraction representation in memory to achieve efficiency. Next type is used to specify abstraction representation in memory. Control method is chosen automatically based on the value of variable of this type in a particular abstraction.

```
type Specification_SynthesisMethod_AbstractionRepresentation is (
ReachAvoid_StandardDijkstra_SparseMatrix,
ReachAvoid_OnTheFlyDijkstra_SparseMatrix,
Invariance_StandardMethod_SparseMatrix,
Invariance_OnTheFlyMethod_SparseMatrix);
```

Listing 1: Type indicating structure of Abstraction

Then abstraction type has structure:

```
 type Abstraction_T(Synthesis:
    Specification_SynthesisMethod_AbstractionRepresentation) is record
Number_of_Controls : Input_Index_T;
First_Input_Index  : Input_Index_T;
Last_Input_Index   : Input_Index_T;
Cover              : Cell_Cover_T;
Input_data         : Input_Grid_T;
Overflow_Cell      : Cell_Index_T;
Number_of_all_Transitions            : Counter_Transitions_T:=0;

case Synthesis is
when Specification_SynthesisMethod_AbstractionRepresentation'(
    ReachAvoid_StandardDijkstra_SparseMatrix) =>
Cells                  : CellDataGrid.Grid_Record_With_Data;
Overflow_Cell_Data     : Cell_T;
when Specification_SynthesisMethod_AbstractionRepresentation'(
    ReachAvoid_OnTheFlyDijkstra_SparseMatrix) =>
```

```
Cells_OTF              : CellDataGrid_OTF.Grid_Record_With_Data;
when Specification_SynthesisMethod_AbstractionRepresentation'(
    Invariance_StandardMethod_SparseMatrix)   =>
Cells_Invariance       : CellDataGrid_Invariance.Grid_Record_With_Data;
Overflow_Cell_Data_Invariance     : Cell_Invariance_T;
when Specification_SynthesisMethod_AbstractionRepresentation'(
    Invariance_OnTheFlyMethod_SparseMatrix)   =>
Cells_Invariance_OTF   : CellDataGrid_Invariance_OTF.Grid_Record_With_Data;
when others =>

null;

end case;

end record;
```

Listing 2: Abstraction type

where structure to represent data associated with cells varies in dependence of a variable with abstraction type.

ABS abstraction computation also depends on the given control problem. Functionality to compute full abstractions independent of control objective is not implemented since abstraction require extreme memory costs and are expected to only be partially constructed at any point in time during synthesis. When abstraction is computed in full, it is expected to be used for small problems and classical algorithms and deleted afterwards, without outputting it outside RAM. Abstract target and initial cells (section 6.2) are stored separately in arrays and obstacle cells are only marked in a boolean variable associated with a cell. This is done to improve efficiency, because initialization of implemented synthesis algorithms, depends on specific control problem and requires iterations over target cells. Then abstract control problem is coded through following types:

```
type Abstract_Specification_T is record
InitialCells: Array_of_Cell_Index_T_Access;
-- A list with cells forming the abstract initial set
TargetCells : Array_of_Cell_Index_T_Access;
-- A list with cells forming the abstract target set
end record;
```

Listing 3: Initial; Target abstract states

and cells types:

```
 type Cell_T is record
Predecessors            : Predecessors_T;
counters_rem_successors : access Array_of_Counter_T;
target   : Boolean      := False;
obstacle : Boolean      := False;
initial  : Boolean      := False;
end record;

type Cell_OTF_T is record
Predecessors            : Predecessors_OTF_T_Access:=null;
Boolean_Marks           : Array_of_Boolean_Marks_Access:=null;
target   : Boolean      := False;
obstacle : Boolean      := False;
initial  : Boolean      := False;
end record;

type Cell_Invariance_T is record
Predecessors            : Predecessors_T;
```

```
target   : Boolean      := False;
obstacle : Boolean      := False;
initial  : Boolean      := False;
end record;

type Cell_Invariance_OTF_T is record
Stored_Overapproximation: Abstract_Reachable_Set_Geometry_T;
target   : Boolean      := False;
obstacle : Boolean      := False;
initial  : Boolean      := False;
end record;
```

Listing 4: Cell types

Since abstract data stored with a cell depends on the chosen synthesis method, there are different cells types, each containing a field to mark an obstacle.

Further functionality in this package includes reading and writing data to abstract cell and is omitted here.

8.3.2 Abstraction_I14sym.Computation

Methods to actually compute abstractions.

```
procedure Install(P : in Problem_T;
Abstraction: in out Abstraction_T);
```

Listing 5: Initialize data structure without storing transitions

```
procedure Compute(
Abstraction : in out Abstraction_T;
AbstractSpecification : in out Abstract_Specification_T;
P : in Problem_T);
```

Listing 6: Compute full abstraction

8.3.3 Abstraction_I14sym.predecessors

This package organizes the storing of the predecessors of each cell.

```
function Initialize_Predecessors(Number_of_Controls : in Input_Index_T) return
    Predecessors_T;
```

Listing 7: Init data structure

```
   procedure Get_Predecessors(
cell        : in Cell_Index_T;
v           : in Input_Index_T;
Abstraction : in Abstraction_T;
Predecessors: in out Dynamic_Cell_Container;
List_of_Overapproximations : in out Dynamic_Overapproximation_Container) ;
```

Listing 8: Read predecessor of a cell under abstract transition function

Notice that abstract transition function is always organized in form of storing predecessors and not successors in accordance to section 7.

8.3.4 Apriori_Enclosure

To compute apriori enclosure for a system.

```
procedure Compute(P : in out Problem_T);
```

Listing 9: Apriori Enclosure

Computes enclosure for f at both sampling time τ and $-\tau$. See section 6.1 and 7.1.

8.3.5 Bounds_Approximation_Error

```
procedure Compute(P : in out Problem_T);


```

Listing 10: Error Bounds

Pre-computes bounds on numerical errors that can occur during synthesis. See [113]. User input language contains no option to turn off usage of pre-computed error bounds in ABS. This has been done intentionally since non-formally correct synthesis was not planned. However, current way of estimating these errors can be too conservative on some interesting example. Therefore the following function was introduced to set computed errors to 0.

```
  procedure Set_Errors_to_Zero(P : in out Problem_T);
```

Listing 11: Zero Bounds

8.3.6 Cell_Cover

Types to define cell cover of continuous domain. Based on two generic packages.

```
package BasicGrid is new grids(Dimension_Index_Type      => Component_Index_T,
Float_Type                => Float_T,
Vector_Type               => State_T,
Vector_Type_Access        => State_T_Access);
-- Instance of a package for a uniform grid whose points will become the centers
    of the compact cells of the abstraction
package CellGrid is new BasicGrid.intersections(NonNeg_Float_Type =>
    NonNegative_Float_T,
Radius_Type       => State_Radius_T,
Periods_Type      => List_of_Component_Index_T);
-- Instance of a package for a uniform cell cover to be used for computing the
    transitions
```

Listing 12: Cell Cover

8.3.7 Established_Types

Global types to describe control problem.

```
  type Problem_T is record
State_Space_Dimension : Component_Index_T;
-- The state space dimension
Input_Space_Dimension : Component_Index_T;
-- The input space dimension
xmin                  : State_T_Access;
-- The hyperinterval [[xmin,xmax]] is the union of the compact cells
xmax                  : State_T_Access;
-- The hyperinterval [[xmin,xmax]] is the union of the compact cells
```

```
umin                : Input_T_Access;
-- The hyperinterval [[umin,umax]] is the continuous input set
umax                : Input_T_Access;
-- The hyperinterval [[umin,umax]] is the continuous input set
List_of_periodic_components    : List_of_Component_Index_T_Access;
-- The components in which the right hand side possesses a period
List_of_nonperiodic_components : List_of_Component_Index_T_Access;
-- The components in which the right hand side does not possess a period
Initial_State_Space_Subdivision : State_Space_Subdivision_T_Access;
-- User-defined discretization of the state space
Initial_Input_Space_Subdivision : Input_Space_Subdivision_T_Access;
-- User-defined discretization of the input space
Bounds_of_Dynamic_Uncertainties  : Bounds_of_Dynamic_Uncertainties_T_Access;
-- The vector with the bound on the dynamic uncertainties
Bounds_of_Measurement_Errors : Bounds_of_Measurement_Errors_T_Access;
-- The vector with the bound on the measurement errors
Sampling_Time : Time_T ;
-- The sampling time
Initial_State_Set  : Compact_Hyperinterval_T_Array_Access;
-- The initial set of the reach-avoid specification
Target_State_Set   : Compact_Hyperinterval_T_Array_Access;
-- The target set of the reach-avoid specification
Obstacle_State_Set : Compact_Hyperinterval_T_Array_Access;
-- The obstacle set of the reach-avoid specificaition
General_Solution_Formula  : Integration_T;
-- The integration formula for integrating the continuous-time dynamics
Growth_Bound_Formula : Growth_Bound_Integration_T;
-- The approximation formula for the growth bounds

Apriori_Enclosure                                  :
    Compact_Hyperinterval_T_Access;
Apriori_Enclosure_Backward_with_DynamicUncertanties :
    Compact_Hyperinterval_T_Access;
Apriori_Enclosure_Backward_wout_DynamicUncertanties :
    Compact_Hyperinterval_T_Access;

-- The apriori enclosure for continuous-time dynamics (it is a uniform one in the
    state and control input)
Apriori_Enclosure_Growth_Bound : Compact_Hyperinterval_T_Access;
Apriori_Enclosure_Growth_Bound_Backward_with_DynamicUncertanties :
    Compact_Hyperinterval_T_Access;
Apriori_Enclosure_Growth_Bound_Backward_wout_DynamicUncertanties :
    Compact_Hyperinterval_T_Access;
-- The apriori enclosure for the growth bounds (it is a uniform one in the state
    and control input)
Rounded_Initial_State_Radius                  : State_Radius_T_Access;
-- See Programmer's Manual;
Bounds_of_Lipschitz_Matrices                  : Vector_Float_T_Access ;
-- See Programmer's Manual;
Bounds_of_Input_Value_Rounding_Error          :
    Bounds_of_Numerical_Errors_Input_T_Access;
-- See Programmer's Manual;
Bounds_of_Approximation_Error_of_General_Solution  :
    Bounds_of_Numerical_Errors_State_T_Access;
-- Bound on the distance between the (mathematical) solution to x'=f(x,u) and its
    approximation formula
Bounds_of_Rounding_Error_of_General_Solution       :
    Bounds_of_Numerical_Errors_State_T_Access;
-- Bound on the distance between previous approximation formula and its floating-
    point implementation
Bounds_of_Approximation_Error_of_Growth_Bound :
    Bounds_of_Numerical_Errors_State_T_Access;
```

```
59 -- Bound on the distance between the (mathematical) solution to r'=Lr + w and its
       approximation formula
60 Bounds_of_Rounding_Error_of_Growth_Bound      :
       Bounds_of_Numerical_Errors_State_T_Access;
61 -- Bound on the distance between previous approximation formula and its floating-
       point implementation
62 Bounds_of_Summation_Error_Growth_Bound     :
       Bounds_of_Numerical_Errors_State_T_Access;
63 --  Bound on the rounding error when adding radii
64 Bounds_of_Summation_Error_General_Solution :
       Bounds_of_Numerical_Errors_State_T_Access;
65 --  Bound on the rounding error when adding center of cell to a radius
66 Bounds_of_Overapproximation_Rounding_Error    :
       Bounds_of_Numerical_Errors_State_T_Access;
67 -- See Programmer's Manual;
68 Bounds_of_Overapproximation_Radius            :
       Bounds_of_Numerical_Errors_State_T_Access;
69 -- See Programmer's Manual;
70 end record;
71
```

Listing 13: Type to describe control problem

Note that the software contains and stores apriori enclosures both for forward sampling time and backward sampling time - see sections 6.1,7 .

8.3.8 Growth_Bound

This package provides a data type to store growth bounds and two interfaces. The first one is to compute growth bounds and return the result values and the second one is to provide a particular value according to the cell index and input index.

```
1   procedure Compute_Growth_Bound
2  ( P : in Problem_T; GB_forward : out Growth_Bound_Values_T_Access );
3  -- Compute growth bounds after loading problem.
4  -- @param P The problem to compute growth bounds from
5  -- @exception
6  -- @return The access of the object storing growth bounds
7
8  procedure Compute_Backward_Growth_Bound_with_DynamicUncertanties
9  ( P : in Problem_T; GB_backward : out Growth_Bound_Values_T_Access );
10
11 procedure Compute_Backward_Growth_Bound_wout_DynamicUncertanties
12 ( P : in Problem_T; GB_backward : out Growth_Bound_Values_T_Access );
13
14
15 procedure Get_Growth_Bound
16 (GBV          : in out State_Radius_T; -- "GBV" stands for "growth bound value"
17 GBVs_Acc     : in  Growth_Bound_Values_T_Access;
18 Cell_Ind     : in  Cell_Index_T;
19 Input_Ind    : in  Input_Index_T);
20 -- Get a growth bound according to the cell and input.
21 -- @param GBV The growth bound value to obtain
22 -- @param GBVs_Acc The access to the object storing growth bounds
23 -- @param Cell_Ind The index of the cell
24 -- @param Input_Ind The index of the input
25
```

Listing 14: Growth Bounds

Maps f^-, F^- are computed here.

8.3.9 Input_Values

Input values package is based on generic grids package, analogously to Cell Cover package.

```
package Input_Values is
package InputGrid is new grids(Float_Type           => Float_T,
Dimension_Index_Type => Component_Index_T,
Vector_Type          => Input_T,
Vector_Type_Access   => Input_T_Access);
subtype Input_Index_T is InputGrid.Grid_Index_Type;
-- The type of the input index
type Input_Grid_T is new InputGrid.Grid_Record with null record;
-- The type of set of input values
```

Listing 15: Input Values

8.3.10 Controller_i14sym

Package to initialize and manipulate symbolic controllers.

```
type Controller_Data_Structure is (
Defined_by_Abstraction,
Explicit_Array_for_Reachabillity_Problem,
Explicit_Array_for_Safety_Problem_OnTheFly_Synthesis,
Explicit_Array_for_Safety_Problem_Standard_Synthesis
);
```

Listing 16: Controller Data Structure

Currently controllers can be stored only in arrays. If more efficient structures are founds (for example Decision Diagrams), code can be easily extended.

8.3.11 Dijkstra_Algorithm_i13absoc

Package to synthesize symbolic controllers. Contains several synthesis methods. Methods applied depends on the type of abstraction provided as input.

```
procedure Compute(Solution    : in out Abstract_OCP_Solution_Type;
Abstraction : in out abstraction_i14sym.Abstraction_T;
AbstractSpec: in abstraction_i14sym.Abstract_Specification_T );
-- @description This procedure solves the underlying optimal control problem
-- @param Solution The solution (control law and value function) of the OCP
-- @param Abstraction
-- @param AbstractSpec
  );

```

Listing 17: Synthesis

8.3.12 Grids

Generic package to define both state and input grids.

```
generic
type Float_Type is digits <>;
-- The type to represent real numbers;
type Dimension_Index_Type is range <>;
-- The type to enumerate the components of a vector
type Vector_Type is array (Dimension_Index_Type range <>) of Float_Type;
-- The type to represent a vector in the real vector space
```

```
type Vector_Type_Access is access Vector_Type;
Bits: constant := 32;

-- The maximum number of grid points is 2**Bits - 1.
-- The last index (2**Bits - 1) is reserved for exceptional situations
type Grid_Index_Type is range 0 ..  2**Bits - 1 ;
for Grid_Index_Type'Size use Bits;

 -- A finite grid on a compact hyperinterval of a more dimensional real space.
type Grid_Record is tagged record
Xmin       : Vector_Type_Access;
-- The lower end point of the hyperinterval
Xmax       : Vector_Type_Access;
-- The upper end point of the hyperinterval
Discretization: Vector_of_Coordinate_Index_Access;
-- The subdivision coefficients for the grid
Number_of_grid_points : Number_of_grid_points_Type;
-- The number of grid points in the grid
Parameter        : Vector_Type_Access;
-- Parameter = (xmax - xmin) / Discretization (component-wise)
end record;

  );
```

Listing 18: Synthesis

Note that currently only grid indices of 32 bit size are allowed. This means abstractions can have not more than $2^{32} - 1$ states and inputs.

8.3.13 Grids.intersections

This package finds cells that intersect reachable sets according to (7).

```
function Default_Boolean_Test_1  (cell: in Grid_Index_Type; successor : in
    Grid_Index_Type) return Boolean;
function Default_Boolean_Test_2  (cell: in Grid_Index_Type; successor : in
    Grid_Index_Type) return Boolean;

function traverse_overapproximation (
cell : in Grid_Index_Type;
center : in Vector_Type ;
rad    : in Radius_Type;
grid   : in Grid_on_Manifold_Record;
successors : in out Cell_Container;
Boolean_Test_1:   access function (cell: in Grid_Index_Type; successor : in
    Grid_Index_Type) return Boolean:=Default_Boolean_Test_1'Access;
Boolean_Test_2:   access function (cell: in Grid_Index_Type; successor : in
    Grid_Index_Type) return Boolean:=Default_Boolean_Test_2'Access
) return Boolean ;

-- @description This function traverses the cells that intersect the
    overapproximation hyper-interval H := [[center-rad,center+rad]]
-- @param cell The cell from which the overapproximation has been computed
-- @param center The center of the hyper-interval
-- @param rad The radius of the hyper-interval
-- @param grid
--  @param successor list of found cells
--  @param Boolean_Test_1 boolean function to be applied to each traversed cell
--  @param Boolean_Test_2 boolean function to be applied to each traversed cell


function continue_traverse_overapproximation(
```

```
25 cell : in Grid_Index_Type;
26 grid   : in Grid_on_Manifold_Record;
27 successors : in out Cell_Container;
28 lower_index: in Grid_Index_Type;
29 upper_index: in Grid_Index_Type;
30 start_index: in Grid_Index_Type;
31 Boolean_Test_1:   access function (cell: in Grid_Index_Type; successor : in
       Grid_Index_Type) return Boolean:=Default_Boolean_Test_1'Access;
32 Boolean_Test_2:   access function (cell: in Grid_Index_Type; successor : in
       Grid_Index_Type) return Boolean:=Default_Boolean_Test_2'Access
33 ) return Boolean;
34
35 -- @description This function traverses the cells that intersect the
       overapproximation hyper-interval determined by lower_index and upper_index
       starting from start_index
36 -- @param cell The cell from which the overapproximation has been computed
37 -- @param grid
38 -- @param successor list of found cells
39 -- @param lower_index index of a cell that contains lower point of
       overapproximation interval
40 -- @param upper_index index of a cell that contains upper point of
       overapproximation interval
41 -- @param start_index index of a cell from which the taversal starts
42 --  @param Boolean_Test_1 boolean function to be applied to each traversed cell
43 --  @param Boolean_Test_2 boolean function to be applied to each traversed cell
44  );
```

Listing 19: On-the-fly synthesis functionality

This functionality is also used to perform operations on lines 2 Function 6 and 4, 6 Function 7 during synthesis. Notice that synthesis algorithms are implemented in a different package - Dijkstra_Algorithm_i13absoc. Ada strong types do not allow this package to manipulate any functionality associated with computation of abstract transitions. For standard synthesis algorithms this is not a problem since they operate on abstractions already computed. In contrast, novel methods, introduced in this work, compute transitions when needed. This would require merging Grids and Dijkstra packages which would break clear software structure. This problem is resolved by supplying to Grids package pointers to clearly defined boolean tests, which are constructed by Dijkstra package during synthesis.

9 Numerical tests

This is the final section of the present work. It includes numerical tests of the developed algorithms as well as comparisons with competing software on a number of examples.

9.1 Competing software

Methods presented in this thesis, implemented in the software package ABS [113] are compared with the software SCOTS, MASCOT and ROCS, all written in C++. SCOTS [95] pre-computes full abstractions before synthesis on a fixed state grid and utilizes either Sparse Matrices or Binary Decision Diagrams (BDDs), which may reduce memory usage. MASCOT [37] utilizes multiple abstraction layers stored in BDDs. ROCS [56] utilizes interval arithmetic and refines state-space discretization if synthesis fails at coarser levels.

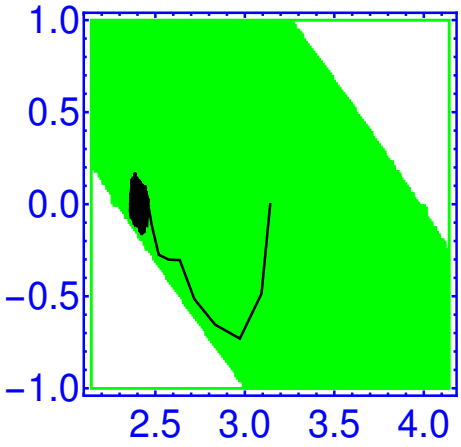


Figure 5: maximal controlled invariant set for example 9.2.1 (green), simulated randomly disturbed controlled trajectory starting from $(\pi, 0)$ for 5000 sampling steps (black)

9.2 Examples

We proceed to description of control tasks, to which the above software packages have been applied. The problem data of examples below includes target sets D and obstacle sets M. The maps G, g of an optimal control problem $\Pi = (X, U, F, G, g)$ are defined as follows:

Reach-avoid: $G(p) = 0$ if $p \in D \setminus M$, and otherwise $G(p) = \infty$, and $g(p, q, u) = 1$ if $p \notin M$, and otherwise $g(p, q, u) = \infty$. Invariance: $G(p) = \infty$, and $g(p, q, u) = 0$ if $p \in D \setminus M$, and otherwise $g(p, q, u) = \infty$.

For every problem the tool MASCOT utilizes 3 discretization layers, unless specified otherwise. MASCOT constructs abstraction layers as follows: State discretization of each next layer is 2 times finer in every dimension. Sampling time for each next layer is 2 times smaller. Last layer parameters correspond to the ones given in the problem data. ROCS non-uniform grids cannot have discretization finer than those specified in the example problem data. SCOTS and ABS employ uniform stae and input space grids.

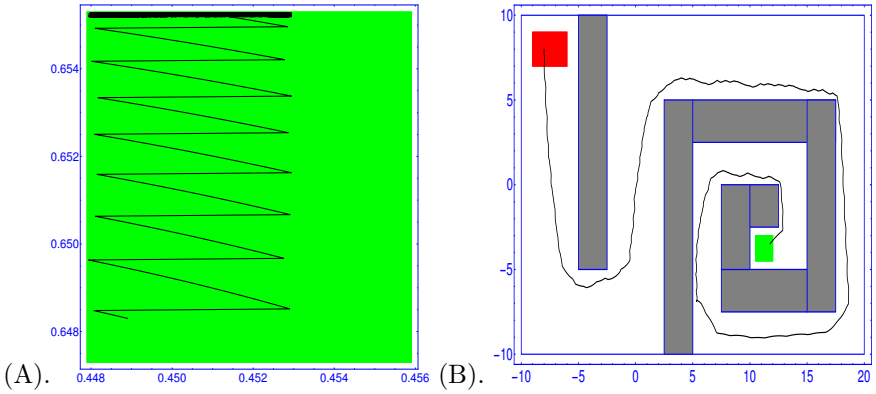
9.2.1 Pendulum Invariance around unstable equilibrium

Dynamics: We consider disturbed pendulum model ([85]), control problem taken from [61].

$$\dot{x} \in \begin{pmatrix} x_2 \\ -\sin(x_1) - \cos(x_1)u \end{pmatrix} + W \tag{54}$$

where x_1 is the pole angle and x_2 is the angular velocity. We are interested in study the disturbed dynamics behavior and finding a controller that provably keeps the pole in a set around unstable equilibrium point $(\pi, 0)$. Disturbance set W models unknown friction dynamics. We apply the developed algorithms and competing software to find maximal controlled invariant subset of $[\pi - 1, \pi + 1] \times [-1, 1]$. We let input domain to be $[-1.9, 1.9]$, and dynamical disturbances be bounded by $W = ([0, 0], [-0.5, 0.5])$. State-space discretization parameters: $(256, 256)$; Input-space discretization parameters: (19). We sample the system with 0.2 seconds.

Technical details: computations were run on a computer equipped with 8 Intel(R) Xeon(R) Platinum 8168 processors and 1 TB of RAM.



(A). (B).

Figure 6: (A) - Simulated trajectory from point $(0.4519 - 0.003, 0.6513 - 0.003)$, for example 9.2.2. (B) - Simulated robot maneuver for example 9.2.3 (states x_1, x_2) starting from point $(-8, 8, -\pi/2)$.

9.2.2 Engine Invariance

Dynamics: We consider a system describing Moore-Greitzer engine [58], control problem taken from [113].

$$\begin{pmatrix} \dot{\Phi} \\ \dot{\Psi} \end{pmatrix} = \begin{pmatrix} \frac{1}{l_c}(\psi_c - \Psi) + u_1 \\ \frac{1}{4l_c B^2}(\Phi - u_2\sqrt{\Psi}) \end{pmatrix}$$

where $\psi_c = a + H[1 + 1.5(\Phi/W - 1) - 0.5(\Phi/W - 1)^3], a = 1/3.5, H = 0.18, l_c = 8, B = 2, W = 0.25$, are engine parameters related to the configuration. The states Φ and Ψ are the average flow rate and pressure of an axial-flow jet engine compressor, respectively. The control inputs are the throttle coefficient u_2 and an additional control u_1. Input Domain : $[-0.05, 0.05] \times [0.5, 0.8]$. Sampling Time : 0.1. State-space discretization parameters: $(500, 500)$; Input-space discretization parameters: $(3, 3)$; The invariance problem consists in finding maximal controlled invariant subset of $[0.4479, 0.4559] \times [0.6473, 0.6553]$. A closed-loop trajectory is illustrated in Fig. 6.

Technical details: computations were run on a computer equipped with 8 Intel(R) Xeon(R) Platinum 8168 processors and 1 TB of RAM.

9.2.3 Robot navigation in a complex environment

Dynamics: we consider dynamics of a two-wheeled mobile robot taken from [50] with disturbance, control problem taken from [113].

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{pmatrix} \in \begin{pmatrix} u_1 \cdot \cos(x_3)/2 + u_2 \cdot \cos(x_3)/2 \\ u_1 \cdot \sin(x_3)/2 + u_2 \cdot \sin(x_3)/2 \\ 5u_1 - 5u_2 \end{pmatrix} + W. \tag{55}$$

We now consider a problem of robot navigation in an obstacle environment, illustrated in Fig. 6. Let the problem data be:

$$
\begin{aligned}
\text{Domain} =& [-10, 20] \times [-1.0, 1.0] \times [-\pi/2, \pi/2] \\
\text{Target Set} =& [10.5, 12.0] \times [-4.5, -3.0] \times [-\pi/2, \pi/2] \\
\text{Initial Set} =& [-9.0, -6.0] \times [7.0, 9.0] \times [-\pi/2, \pi/2] \\
\text{Obstacle Set} =& [-5.0, -2.5] \times [-5.0, 10] \times [-\pi/2, \pi/2] \\
& \cup [2.5, 5.0] \times [-10, 5.0] \times [-\pi/2, \pi/2] \\
& \cup [5.0, 17.5] \times [2.5, 5.0] \times [-\pi/2, \pi/2] \\
& \cup [15, 17.5] \times [-7.5, 5.0] \times [-\pi/2, \pi/2] \\
& \cup [7.5, 15] \times [-7.5, -5.0] \times [-\pi/2, \pi/2] \\
& \cup [7.5, 10] \times [-5.0, 0.0] \times [-\pi/2, \pi/2] \\
& \cup [10, 12.5] \times [-2.5, 0.0] \times [-\pi/2, \pi/2]
\end{aligned}
$$

Input domain $U = [-2, 2] \times [-2, 2]$. Sampling time: 0.1. Dynamical disturbances: $W = ([-0.5, 0.5], [-0.5, 0.5], [-0.5, 0.5])$ State-space discretization: $(300, 300, 100)$ Input-space discretization: $(5, 5)$. The control problem consists in driving the system from the Initial set to the Target Set, while avoiding obstacles. A closed-loop trajectory is illustrated in Fig. 6.

Technical details: computations were run on a computer equipped with 8 Intel(R) Xeon(R) Platinum 8168 processors and 1 TB of RAM.

9.2.4 Rocket velocity change

Dynamics: We consider a system describing rocket flight [16], control problem taken from [62].

$$
\begin{pmatrix} \dot{V} \\ \dot{\gamma} \\ \dot{h} \end{pmatrix} = \begin{pmatrix} \frac{T\cos(u)-D}{m} - g\sin(\gamma) \\ \frac{T\sin(u)+L}{mV} - \frac{g\cos(\gamma)}{V} + V\frac{\cos(\gamma)}{h+R_0} \\ V\sin(\gamma) \end{pmatrix}
$$

where V is the velocity, γ is the flight path angle, h is the height, and the angle of attack u is the input to the system, $D = 0.5\rho V^2 S_{ref}(C_X\cos(u) + C_N\sin(u))$, $L = 0.5\rho V^2 S_{ref}(-C_X\sin(u) + C_N\cos(u))$, $\rho = 1.28395$ is the atmospheric density, $C_X = 0.2907$, $C_N = 1.0643$, $S_{ref} = 3$, $g = g_0(R_0/(h+R_0))^2$, $g_0 = 9.81$, $R_0 = 6371$ is the the radius of earth, $m = 16500$, $T = 645000$. The control problem consists in driving the sampled system, with sampling time $\tau = 0.1$, from the initial point $(2750, -0.69, 44990)$ to the target set $[3000, 4000] \times [-0.1, 0.1] \times [40000, 45000]$. The admissible state domain is $X_1' = [1000, 4000] \times [-1.5, 1.5] \times [40000, 45000]$, which is discretized using $252 \times 252 \times 252$ cells, and the abstraction uses 10 equidistant inputs from $U_1 = [-0.8, 0.8]$.

Technical details: computations were run on a computer equipped with AMD EPYC 7452 CPU and 1 TB of RAM. Due to the fact that method of bounding numerical errors employed by ABS being very conservative, this example has been solved by ABS disregarding numerical error bounds and turning off Ada language run-time checks.

9.2.5 Pendulum-cart system swing up

Dynamics: We consider pendulum on a cart model:

$$
\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{pmatrix} = \begin{pmatrix} x_2 \\ -\sin(x_1) - \cos(x_1)u - 2\gamma x_2 \\ x_4 \\ u \end{pmatrix} \tag{56}
$$

where $\gamma = 0.0125$, x_1 is the pendulum angle, x_2 - angular velocity, x_3 - cart position, x_4 - cart velocity. We aim to synthesize controllers that drive the pendulum starting from point $(0, 0, 0, 0)$ upwards, to the target set $[\pi - 0.3, \pi + 0.3] \times [-0.3, 0.3] \times [-0.5, 0.5] \times [-0.5, 0.5]$. We let the admissible state and input domains be $[0, 2\pi] \times [-2.80, 2.80] \times [-1.7, 1.7] \times [-1.7, 1.7]$ and $[-2, 2]$ with discretization parameters $(200, 200, 80, 80)$ and (15) correspondingly. The sampling time is 0.3 seconds.

Technical details: computations were run on a computer equipped with 8 Intel(R) Xeon(R) Platinum 8168 processors and 1 TB of RAM. Due to the fact that method of bounding numerical errors employed by ABS being very conservative, this example has been solved disregarding bounds on errors of numerical operations.

9.2.6 Double Pendulum invariance

Dynamics: We consider a challenging example, involving perturbed dynamics of the double pendulum on a cart [114], control problem taken from [62].

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} x_3 \\ x_4 \end{pmatrix} \tag{57a}$$

$$\begin{pmatrix} \dot{x}_3 \\ \dot{x}_4 \end{pmatrix} = A^{-1} b \tag{57b}$$

where (x_1, x_2) are the pendulum angles and (x_3, x_4) angle velocities and

$$A = \begin{pmatrix} J_1 + a_1^2 m_1 + l_1^2 m_2 & a_2 l_1 m_2 \cos(x_1 - x_2) \\ a_2 l_1 m_2 \cos(x_1 - x_2) & J_2 + a_2^2 m_2 \end{pmatrix} \tag{58}$$

$$b = \begin{pmatrix} (a_1 m_1 + l_1 m_2) g \sin(x_1) - a_2 l_1 m_2 \sin(x_1 - x_2) x_4^2 - d_1 x_3 - d_2 (x_3 - x_4) + (a_1 m_1 + l_1 m_2) \cos(x_1) u \\ a_2 g m_2 \sin(x_2) + a_2 l_1 m_2 \sin(x_1 - x_2) + d_2 (x_3 - x_4) + a_2 m_2 cos(x_2) u \end{pmatrix} \tag{59}$$

The constants are given in the table Tab. 2 [30].

We seek to find a forward invariant set around $(\pi, \pi, 0, 0)$ with sampling time $\tau = 0.005$. The state domain is $X_1' = [\pi - 0.1, \pi + 0.1]^2 \times [-0.27, 0.27]^2$, which is discretized using $200 \times 200 \times 200 \times 200$ cells, and the abstraction uses 10 equidistant inputs from $U_1 = [-34.335, 34.335]$. For this example, MASCOT uses 2 discretization layers with sampling times 0.01 and 0.005, respectively. We have chosen only 2 layers, since coarser abstractions are unable to produce non-empty controllers for this problem.

Technical details: computations were run on a computer equipped with AMD EPYC 7452 CPU and 1 TB of RAM. Due to the fact that method of bounding numerical errors employed by ABS being very conservative, this example has been solved by ABS disregarding numerical error bounds and turning off Ada language run-time checks.

9.2.7 Heavily disturbed quad-copter flight

Dynamics: We consider quad-rotor model [121]:

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{pmatrix} \in \begin{pmatrix} 0.2 \cdot \cos(x_3) \\ 0.2 \cdot \sin(x_3) \\ x_4 \\ u - 0.25 \cdot x_3 \end{pmatrix} + W \tag{60}$$

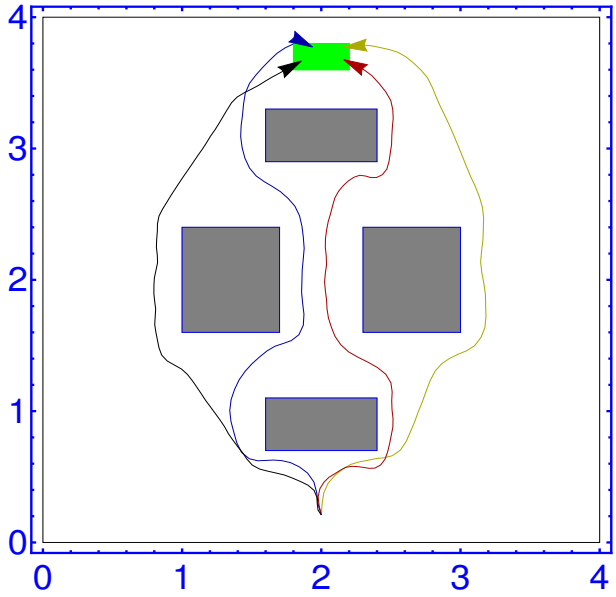


Figure 7: 4 simulated quad-copter maneuvers (colored) from example 9.2.7 with 4 random disturbance signals supplied to the system

where (x_1, x_1) are planar quad-rotor coordinates, (x_3, x_4) is the orientation angle and angle velocity respectively. Dynamical disturbance W models influences of un-modeled quad-rotor dynamics as well as unexpected weather effects. Problem Data:

$$\begin{aligned}
\text{Domain} =& [0.0, 4.0] \times [0.0, 4.0] \times [-\pi, \pi] \times [-2.0, 2.0] \\
\text{Target Set} =& [1.8, 2.2] \times [3.6, 3.8] \times [-\pi, \pi] \times [-2.5, 2.5] \\
\text{Initial Set} =& [2.0, 2.0] \times [0.3, 0.3] \times [\frac{\pi}{2}, \frac{\pi}{2}] \times [0.0, 0.0] \\
\text{Obstacle Set} =& [1.6, 2.4] \times [0.7, 1.1] \times [-\pi, \pi] \times [-2.5, 2.5] \\
& \cup [1.6, 2.4] \times [2.9, 3.3] \times [-\pi, \pi] \times [-2.5, 2.5] \\
& \cup [1.0, 1.7] \times [1.6, 2.4] \times [-\pi, \pi] \times [-2.5, 2.5] \\
& \cup [2.3, 3.0] \times [1.6, 2.4] \times [-\pi, \pi] \times [-2.5, 2.5]
\end{aligned}$$

Input Domain : $[-4, 4]$. Sampling Time : 0.3. Dynamic disturbance bound: $W = [-0.081, 0.081] \times [-0.081, 0.081] \times [-0.051, 0.051] \times [-0.051, 0.051]$; State-space discretization parameters: $(180, 180, 150, 150)$; Input-space discretization parameters: (20); The control problem consists in driving the system from the Initial set to the Target Set, while avoiding obstacles.

Technical details: computations were run on a computer equipped with 8 Intel(R) Xeon(R) Platinum 8168 processors and 1 TB of RAM. Due to the fact that method of bounding numerical errors employed by ABS being very conservative, this example has been solved disregarding bounds on errors of numerical operations.

9.3 Discussion of numerical results

Time and memory consumption are reported in tables Tab. 4 and Tab. 3. Analysis of these measurements is now presented.

Table 3: Memory consumption

Example N	SCOTS-BDD	SCOTS-SM	MASCOT	ROCS	ABS (this work)
9.2.1	141 MB	460 MB	346 MB	8 MB	10 MB
9.2.2	270 MB	147 MB	245 MB	10 MB	29 MB
9.2.3	1.1 GB	47 GB	1.8 GB	0.7 GB	0.6 GB
9.2.4	7.5 GB	15.8 GB	4.3 GB	0.55 GB	0.7 GB
9.2.5	>36.8 GB	>1000 GB	>64.8 GB	>54.1 GB	22 GB
9.2.6	>3.4 GB	>2000 GB	>180 GB	>34GB	57 GB
9.2.7	>10 GB	>1000 GB	>15 GB	>82 GB	53 GB

Table 4: Time consumption

Example N	SCOTS-BDD	SCOTS-SM	MASCOT	ROCS	ABS (this work)
9.2.1	40 sec.	2 sec.	22 sec.	6 sec.	8 sec.
9.2.2	43.6 sec.	2.76 sec.	14.9 sec.	0.009 sec.	1.17 sec.
9.2.3	473 min.	9.68 min.	1510 min.	136 min.	35 min.
9.2.4	415 min.	10 min.	93 min.	464 min.	2.1 min.
9.2.5	45 hours[(a)]	-	45 hours[(a)]	45 hours[(a)]	24 hours
9.2.6	728 hours[(a)]	-	123 hours[(a)]	728 hours[(a)]	19 hours
9.2.7	100 hours[(a)]	-	100 hours[(a)]	100 hours[(a)]	61 hours

[(a)] Computations not finished; aborted after indicated time.

For the first example 9.2.1 the newly developed methods outperformed competing algorithms memory-wise, with factors ranging from 14 to 40, except for ROCS, which required 2 MB less space. Regarding time consumption, ABS was slower than SCOTS-SM by 6 seconds, slower than ROCS by 2 seconds and outperformed other works by at least 2 times. Second example 9.2.2 shows similar results: ABS outperforms competitors, except for ROCS, memory-wise with factors 5 to 9 and time-wise with factors 2 to 40. ROCS has solved this problem very efficiently requiring 3 times less memory than ABS and practically no time.

These results are reasonable. ROCS was expected to take approximately the same amount of memory as ABS, due to not storing any abstraction parts and preforming hyper-interval analysis only. On small examples implementation details heavily influence overall time and memory consumption. ROCS and SCOTS are written in C++, which is known to be faster than Ada in general. Moreover, `access` data type in Ada language, used to declare dynamic structures, is known to contain more data than the corresponding pointer type in C, such as first and last indices of the allocated array. This in turn leads to slightly greater memory consumption of ABS.

Problems 9.2.3,9.2.4 confirm advantages of ABS with memory consumption reduction by factor of 2 (SCOTS BDD 9.2.3) to 10 (SCOTS BDD 9.2.3), 20 (SCOTS SM 9.2.4) and reaching 70 (SCOTS SM 9.2.3). Only ROCS took approximately the same amount of memory on both

examples. Time-wise, ABS lost only to SCOTS SM on example 9.2.3, which was 3 times faster. However, ABS outperformed all other competitors with factors ranging from 4 (ROCS 9.2.3) to 200 (SCOTS BDD, ROCS 9.2.4). Notice, with increased problem complexity, implementation details affect resource consumption weakly.

Examples 9.2.5, 9.2.6, 9.2.7 were solved only by ABS in the indicated time under hardware memory limits. For instance the double pendulum invariance problem 9.2.6 has been solved only by ABS, where the safe set occupies less that 1% of the admissible state domain volume. SCOTS-SM could not be run as it would have required more than the main memory available. Indeed, at least 2 TB would have been needed, as seen from the simple lower bound $2^{n+3} \cdot card(X') \cdot card(U')$, where X' and U' are abstract state and input sets, 8 bytes to store a single transition, the number of transitions is not less than 2^n per abstract state-input, and n is the dimension of the state-space. SCOTS-BDD and ROCS have been aborted after 30 days, and MASCOT crashed after 123 hours, which exceeds the time ABS spent to successfully solve the problem by a factor of 37 and 6, respectively. For example 9.2.6 SCOTS-BDD used less memory in the indicated time than ABS, yet has not finished even in 30 days. Thus it is not clear whether the final memory consumption of SCOTS and other competitors would be lower or not. Example 9.2.7 shows similar behavior.

This section is summarized by pointing out that ABS in, general, is able to significantly outperform its competitors. Although, some examples do show that ABS can take more computational resources than others, only novel algorithms were able to solve a number of control problems. Thus numerical tests confirm theoretical advances in application of abstraction-based theory.

10 Conclusions and Outlook

This thesis presents novel controller synthesis methods within abstraction-based methodology. Contrary to existing works, the developed algorithms are applicable to a large class of non-linear systems and place no assumptions on the structure of the abstract state space (such as non-uniform grids) and transition function (such as single-valued) for their efficiency. To provide space relief, the proposed routines do not exploit any specific system properties but only require availability of special predictor maps that direct synthesis procedure in analyzing abstract transitions, and order relations on abstractions to avoid redundant operations. It has been shown that the mentioned predictors can be computed for the same class of systems, for which methods to compute abstraction were presented. Moreover, since abstractions are finite by definition, suitable order relations can always be found. Novel methods, provide guarantees of large space consumption reduction, while maintaining linear time complexity for two fundamental specifications - optimal reachability and qualitative invariance. One of exponential memory requirements of abstraction-based synthesis is thus completely removed. Up to he knowledge of the author these are the first algorithms with such complexity. Nevertheless, it has to be noted that combinatorial "explosion" of number of abstract state and inputs still remains an issue. A software package ABS was developed that implements the discussed methods, and computes abstractions and controllers in purely automated fashion. No manual analysis, such as bounding of the first derivative of the right-hand side,is required. Comparison with existing software has shown that, overall, ABS is able to drastically outperform competitors, and moreover, can synthesize controllers for problems previously unsolvable in abstraction-based setting within reasonable time and space limits.

Although measurable progress in application of symbolic control has been observed, a number of questions has appeared or remains to be open. This work is concluded by discussing further obstacles to theoretical and practical development and possible ways of their mitigation.

First, linear time algorithms (e.g. $O(m)$ in abstraction size) preserving large space relief have been presented only for the worst-case optimal reachability with non-negative transition cost and the worst-case invariance with zero transition cost. Methods concerned with more general cost functions also exist - see [112] for reach-avoid problems allowing negative cost of transitions taking $O(n \cdot m)$ time and section 7.2.3 of this work for optimal worst-case invariance allowing positive-cost transitions taking $O(n \cdot (m + m^- + n \log n))$ time, where n is the size of abstract state space m is the total number of abstract transitions and $m^- \approxeq m$. Can a $O(m)$-time space-efficient algorithm be constructed for the above classes of problems? It seems that for negative transition cost optimal reachability problems this is unlikely since $O(n \cdot m)$ operations could be needed in the worst case to detect negative cycles. However, for invariance problems some results could be expected placing assumptions on structure of transition cost function, such as invertibility. Moreover, is it possible to construct $O(m)$ time $O(n)$ space algorithms for optimal invariance problems that differ from optimizing worst-case trajectory cost? An example could be optimization of average trajectory cost.

Second, large cardinality of the abstract state-space, and consequently the size of abstract controllers, is understood to be one of the main remaining obstacles to practical adoption of symbolic control. This issue has seen acceptable solutions only on special classes of systems - see [67], [11] and literature review in section 3.

A number of techniques are expected to provide significant reduction of controllers size. Algorithms, proposed in this thesis, do not place any specific assumptions on the structure of abstract state-space, transition function, and utilize exhaustive iteration of backward estimators and backward transition maps over all abstract inputs to prove controllability or non-controllability of a cell. ABS also implements only uniform state and input grids. Non-uniform state and input grids promise further computational relief, although care must be taken with additional burden of maintenance of complex state and input spaces - see comparisons in section 9.3 between ROCS and MASCOT (non-uniform state grids) on one hand and SCOTS (uniform state grids) on the other hand, where on various examples most efficient software differs.

It can also be noted that all currently available abstraction-based algorithms compute the reach-avoid value function, and its associated controller, globally, i.e. for every cell in the abstract state-space. This can be useful to study global properties of system dynamics, and resolvability of the given problem for various initial conditions. It can also be necessary for certain problems due to presence of heavy disturbances, measurements errors etc. See for example problem 9.2.7 and Fig. 7 where different disturbance signals force abstract controller to produce drastically different simulated trajectories starting from single initial point. However for many problems, volume of controlled tube starting from the initial set to the target set is very small compared to the volume of complete abstract state-space. Then computation of global value function to produce a single controlled maneuver seems rather unnecessary. Therefore, next general scheme appears to be promising in reducing further computational complexity of abstraction-based synthesis techniques.

(i) Compute a single controlled trajectory from continuous initial set to continuous target set.

(ii) Decompose the initial reach-avoid problem into a sequence of problems between sets constructed along the controlled trajectory.

(iii) Build (parts of) local abstractions in proximity to the computed trajectory. Attempt to synthesize symbolic controllers.

Such approach promises to address combinatorial explosion of number of abstract states and inputs. Since abstractions are expected to be to be built around pre-computed trajectory, number

of needed abstract states to solve local reachability problems is expected to be far smaller, than the number of states needed for computation of sufficiently precise approximation of the global value function. Moreover, heuristic choice of control inputs instead of exhaustive iteration over the inputs space can be envisioned.

Table 1: Problem Specific Files (from [89])

File name	Description
`dynamics_specification_parameters.adb`	Floating-point implementation of the user input in the Ada programming language
`dynamics_specification_parameters.ads`	Specification file to previous
`dynamics_specification_parameters.c`	Floating-point implementation of the user input in the C programming language
`dynamics_specification_parameters.h`	Header file to previous
`Function.adb`	Floating-point implementation of the right hand side of the unperturbed dynamics in the Ada programming language
`Function.ads`	Specification file to previous
`Function.c`	Floating-point implementation of the right hand side of the unperturbed dynamics in the C programming language
`Function.h`	Header file to previous
`Function_ia.c`	Interval arithmetic implementation of the right hand side of the unperturbed dynamics in the C programming language
`Function_ia.h`	Header file to previous
`Function.mma`	Floating-point implementation of the right hand side of the unperturbed dynamics in the Mathematica programming language
`Jacobian.abcs`	Floating-point implementation of the first derivative of the right hand side of the unperturbed dynamics in the problem compiler programming language
`Jacobian.c`	Interval arithmetic implementation of the first derivative of the right hand side of the unperturbed dynamics in the C programming language
`Jacobian.h`	Header file to previous
`taylorcoefficients.abcs`	Taylor coefficients for the right hand side of the unperturbed dynamics in the problem compiler programming language
`taylorcoefficients.adb`	Taylor coefficients for the right hand side of the unperturbed dynamics in the Ada programming language
`taylorcoefficients.ads`	Specification file to previous
`taylorcoefficients.c`	Taylor coefficients for the right hand side of the unperturbed dynamics in the C programming language
`taylorcoefficients.h`	Header file to previous

Table 2: Mechanical parameters of the double pendulum

Pendulum link	Inner $i = 1$	Outer $i = 2$
Length l_i (m)	0.323	0.480
Distance to center of gravity a_i (m)	0.2145	0.223
Mass m_i (kg)	0.853	0.510
Moment of inertia J_i (Nms^2)	0.0126	0.0185
Friction constant d_i (Nms)	0.005	0.005

Bibliography

[1] AdaCore. *GPRbuild and GPR Companion Tools User's Guide*. https://docs.adacore.com/gprbuild-docs/html/gprbuild_ug.html.

[2] Matthias Althoff. Reachability analysis of nonlinear systems using conservative polynomialization and non-convex sets. In *Proc. 16th Intl. Conf. Hybrid Systems: Computation and Control (HSCC), Philadelphia, PA, U.S.A., April 8-11, 2013*, pages 173–182, 2013.

[3] Matthias Althoff, Olaf Stursberg, and Martin Buss. Reachability analysis of nonlinear systems with uncertain parameters using conservative linearization. In *2008 47th IEEE Conference on Decision and Control*, pages 4042–4048, 2008.

[4] Rajeev Alur, Thomas A. Henzinger, Gerardo Lafferriere, and George J. Pappas. Discrete abstractions of hybrid systems. *Proc. IEEE*, 88(7):971–984, July 2000.

[5] Taha Ameen, Shayok Mukhopadhyay, and Nasser Qaddoumi. Computing robust forward invariant sets of multidimensional non-linear systems via geometric deformation of polytopes (https://arxiv.org/abs/2101.10407), 01 2021.

[6] Tzanis Anevlavis and Paulo Tabuada. Computing controlled invariant sets in two moves. pages 6248–6254, 12 2019.

[7] Tzanis Anevlavis and Paulo Tabuada. A simple hierarchy for computing controlled invariant sets. In *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*, HSCC '20, New York, NY, USA, 2020. Association for Computing Machinery.

[8] Zvi Artstein and Sasa V. Rakovic. Feedback and invariance under uncertainty via set-iterates. *Automatica*, 44:520–525, 02 2008.

[9] D. P. Bertsekas and I. B. Rhodes. On the minimax reachability of target sets and target tubes. *Automatica*, 7(2):233–247, mar 1971.

[10] Franco Blanchini and Stefano Miani. *Set-theoretic methods in control*. Systems & Control: Foundations & Applications. Birkhäuser Boston Inc., Boston, MA, 2008.

[11] Alessandro Borri, Giordano Pola, Pierdomenico Pepe, Maria Domenica Di Benedetto, and Pasquale Palumbo. Symbolic control design of an artificial pancreas for type-2 diabetes. *IEEE Transactions on Control Systems Technology*, pages 1–16, 2021.

[12] Dimitris Boskos and Dimos V Dimarogonas. Decentralized abstractions for feedback interconnected multi-agent systems. In *Proc. IEEE Conf. Decision and Control (CDC), Osaka, Japan, 15-18 December 2015*, pages 282–287, 2015.

[13] J.M. Bravo, D. Limon, T. Alamo, and E.F. Camacho. On the computation of invariant sets for constrained nonlinear systems: An interval arithmetic approach. *Automatica*, 41(9):1583–1589, 2005.

[14] Michael Brunner, Bernd Brüggemann, and Dirk Schulz. Hierarchical rough terrain motion planning using an optimal sampling-based method. *2013 IEEE International Conference on Robotics and Automation*, pages 5539–5544, 2013.

[15] Peter E. Caines and Yuan-Jun Wei. Hierarchical hybrid control systems: a lattice-theoretic formulation. *IEEE Trans. Automat. Control*, 43(4):501–508, 1998.

[16] Si-Yuan Chen and Qun-Li Xia. A multiconstrained ascent guidance method for solid rocket-powered launch vehicles. *Intl. J. of Aerospace Engineering*, 2016, 2016.

[17] Congcong Cheng, Xiafu Lv, Junsong Zhang, and Meng Zhang. Robot arm path planning based on improved rrt algorithm. In *2021 3rd International Symposium on Robotics and Intelligent Manufacturing Technology (ISRIMT)*, 2021.

[18] F.L. Chernous'ko. Ellipsoidal estimates of a controlled system's attainability domain. *Journal of Applied Mathematics and Mechanics*, 45(1):7–12, 1981.

[19] M. Cwikel and P.-O. Gutman. Convergence of an algorithm to find maximal state constraint sets for discrete-time linear dynamical systems with bounded controls and states. *IEEE Transactions on Automatic Control*, 31(5):457–459, 1986.

[20] Luca De Alfaro and Pritam Roy. Solving games via three-valued abstraction refinement. In *Intl. Conf. on Concurrency Theory*, pages 74–89. Springer, 2007.

[21] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1:269–271, 1959.

[22] Charles Donnelly and Richard Stallman. *GNU Bison Manual*. Free Software Foundation, 2020. https://www.gnu.org/software/bison/.

[23] Mirko Fiacchini, Teodoro Alamo, and Eduardo Camacho. On the computation of convex robust control invariant sets for nonlinear systems. *Automatica*, 46:1334–1338, 08 2010.

[24] Mirko Fiacchini, Sophie Tarbouriech, and Christophe Prieur. Polytopic control invariant sets for differential inclusion systems: A viability theory approach. *Proc. of the American Control Conference (ACC'11)*, 06 2011.

[25] Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Trans. Math. Software*, 33(2):13:1–13:15, June 2007.

[26] Dirk Förstner, Merten Jung, and Jan Lunze. A discrete-event model of asynchronous quantised systems. *Automatica*, 38(8):1277–1286, 2002.

[27] Antoine Girard and Gregor Gössler. Safety synthesis for incrementally stable switched systems using discretization-free multi-resolution abstractions. *Acta Inform.*, September 2019.

[28] Antoine Girard and George J. Pappas. Approximation metrics for discrete and continuous systems. *IEEE Transactions on Automatic Control*, 52(5):782–798, 2007.

[29] Timothy J. Graettinger and Bruce H. Krogh. Hyperplane method for reachable state estimation for linear time-invariant systems. *Journal of Optimization Theory and Applications*, 69:555–588, 1991.

[30] Knut Graichen, Michael Treuer, and Michael Zeitz. Swing-up of the double pendulum on a cart by feedforward and feedback control with experimental validation. *Automatica J. IFAC*, 43(1):63–71, 2007.

[31] Torbjörn Granlund and the GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*, 5.0.5 edition, 2012. http://gmplib.org/.

[32] Lars Grune and Jurgen Pannek. *Nonlinear Model Predictive Control: Theory and Algorithms*. 01 2011.

[33] Ankit Gupta, Hakan Koroglu, and Paolo Falcone. Computation of low-complexity control-invariant sets for systems with uncertain parameter dependence. *Automatica*, 101:330–337, 2019.

[34] Didier Henrion and Milan Korda. Convex computation of the region of attraction of polynomial control systems. *IEEE Transactions on Automatic Control*, 59(2):297–312, 2014.

[35] Thomas Henzinger, Rupak Majumdar, and Vinayak Prabhu. Quantifying similarities between timed systems. volume 3, pages 226–241, 09 2005.

[36] C. S. Hsu. *Cell-to-cell mapping*, volume 64 of *Applied Mathematical Sciences*. Springer-Verlag, New York, 1987.

[37] Kyle Hsu, Rupak Majumdar, Kaushik Mallik, and Anne-Kathrin Schmuck. Lazy abstraction-based control for safety specifications. In *Proc. 57th IEEE Conf. Decision and Control (CDC), Miami, FL, USA, 17-19 December 2018*, pages 4902–4907, 2018.

[38] Kyle Hsu, Rupak Majumdar, Kaushik Mallik, and Anne-Kathrin Schmuck. Multi-layered abstraction-based controller synthesis for continuous-time systems. In *Proc. 21st Intl. Conf. Hybrid Systems: Computation and Control (HSCC), Porto, Portugal, April 11-13, 2018*, pages 120–129, 2018.

[39] Shouchuan Hu and Nikolas S. Papageorgiou. *Handbook of multivalued analysis. Vol. I*, volume 419 of *Mathematics and its Applications*. Kluwer, 1997.

[40] Omar Hussien, Aaron Ames, and Paulo Tabuada. Abstracting partially feedback linearizable systems compositionally. *IEEE Control Systems Letters*, 1(2):227–232, October 2017.

[41] Omar Hussien and Paulo Tabuada. Lazy controller synthesis using three-valued abstractions for safety and reachability specifications. In *Proc. 57th IEEE Conf. Decision and Control (CDC), Miami, FL, USA, 17-19 December 2018*, pages 3567–3572, 2018.

[42] A. Iannelli, A. Marcos, and M. Lowenberg. Estimating the region of attraction of uncertain systems with invariant sets**this work has received funding from the european union's horizon 2020 research and innovation programme under grant agreement no 636307, project flexop. *IFAC-PapersOnLine*, 51(25):246–251, 2018. 9th IFAC Symposium on Robust Control Design ROCOND 2018.

[43] Gian Paolo Incremona, Antonella Ferrara, and Lalo Magni. Hierarchical model predictive/sliding mode control of nonlinear constrained uncertain systems. *IFAC-PapersOnLine*, 48(23):102–109, 2015. 5th IFAC Conference on Nonlinear Model Predictive Control NMPC 2015.

[44] Elena Ivanova and Antoine Girard. Lazy symbolic controller for continuous-time systems based on safe set boundary exploration. In *7th IFAC Conf. Analysis and Design of Hybrid Systems (ADHS), Brussels, Belgium, 7-9 July 2021*, volume 54 of *IFAC-PapersOnLine*, pages 109–114, 2021.

[45] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[46] Mahmoud Khaled, Eric S. Kim, Murat Arcak, and Majid Zamani. Synthesis of symbolic controllers: A parallelized and sparsity-aware approach. In Tomáš Vojnar and Lijun Zhang, editors, *Proc. Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Czech Rep., April 6-11, 2019*, volume 11428 of *Lect. Notes Comput. Sci.*, pages 265–281. Springer, 2019.

[47] Mahmoud Khaled and Majid Zamani. pFaces: An acceleration ecosystem for symbolic control. In *Proc. 22th ACM Intl. Conf. on Hybrid Systems: Computation and Control (HSCC), Montreal, QC, Canada, April 16-18, 2019*, pages 252–257, 2019.

[48] Eric S. Kim, Murat Arcak, and Sanjit A. Seshia. Symbolic control design for monotone systems with directed specifications. *Automatica J. IFAC*, 83:10–19, 2017.

[49] Eric S Kim, Murat Arcak, and Majid Zamani. Constructing control system abstractions from modular components. In *Proc. 21st Intl. Conf. Hybrid Systems: Computation and Control (HSCC), Porto, Portugal, April 11-13, 2018* [38], pages 137–146.

[50] Shunsuke Kimura, Hisakazu Nakamura, and Yuh Yamashita. Control of two-wheeled mobile robot via homogeneous semiconcave control lyapunov function. In *9th IFAC Symp. Nonlin. Control Systems, Toulouse, France, September 4-6, 2013*, volume 46 of *IFAC Proceedings Volumes*, pages 92–97, 2013.

[51] Milan Korda, Didier Henrion, and Colin N. Jones. Inner approximations of the region of attraction for polynomial dynamical systems. *IFAC Proceedings Volumes*, 46(23):534–539, 2013. 9th IFAC Symposium on Nonlinear Control Systems.

[52] Xenofon D. Koutsoukos, Panos J. Antsaklis, James A. Stiver, and Michael D. Lemmon. Supervisory control of hybrid systems. *Proc. IEEE*, 88(7):1026–1049, July 2000.

[53] Abolfazl Lavaei, Sadegh Esmaeil Zadeh Soudjani, Rupak Majumdar, and Majid Zamani. Compositional abstractions of interconnected discrete-time stochastic control systems. In *Proc. 56th IEEE Conf. Decision and Control (CDC), Melbourne, Australia, 12-15 December 2017*, pages 3551–3556, 2017.

[54] Steven M. LaValle. Rapidly-exploring random trees : a new tool for path planning. *The annual research report*, 1998.

[55] M. Lazar, A. Alessio, A. Bemporad, and W.P.M.H. Heemels. Squaring the circle: an algorithm for generating polyhedral invariant sets from ellipsoidal ones. In *2006 American Control Conference*, pages 6 pp.–, 2006.

[56] Yinan Li and Jun Liu. ROCS: A robustly complete control synthesis tool for nonlinear dynamical systems. In *Proc. 21st Intl. Conf. Hybrid Systems: Computation and Control (HSCC), Porto, Portugal, April 11-13, 2018* [38], pages 130–135.

[57] Yinan Li, Zhibing Sun, and Jun Liu. ROCS 2.0: An integrated temporal logic control synthesis tool for nonlinear dynamical systems. In *7th IFAC Conf. Analysis and Design of Hybrid Systems (ADHS), Brussels, Belgium, 7-9 July 2021* [44], pages 31–36.

[58] Yinan Li, Zhibing Sun, and Jun Liu. A specification-guided framework for temporal logic control of nonlinear systems. Technical Report 2104.01385v1, arXiv.org, April 3 2021.

[59] Chengyuan Liu, Furqan Tahir, and Imad M. Jaimoukha. Full-complexity polytopic robust control invariant sets for uncertain linear discrete-time systems. *International Journal of Robust and Nonlinear Control*, 29(11):3587–3605, 2019.

[60] Rudolf Lohner. *Einschließung der Lösung gewöhnlicher Anfangs- und Randwertaufgaben und Anwendungen*. Dissertation, Univ. Karlsruhe, Fak. f. Mathematik, 1988.

[61] Elisei Macoveiciuc and Gunther Reissig. Guaranteed memory reduction in synthesis of correct-by-design invariance controllers. In *Proc. 21st IFAC World Congress, Berlin, Germany, July 12-17, 2020*, volume 53 of *IFAC-PapersOnLine*, pages 5561–5566, 2020.

[62] Elisei Macoveiciuc and Gunther Reissig. On-the-fly symbolic synthesis with memory reduction guarantees. *IEEE Transactions on Automatic Control*, 68(4):2576–2583, 2023.

[63] Anirudha Majumdar and Russ Tedrake. Funnel libraries for real-time robust feedback motion planning. *The International Journal of Robotics Research*, 36(8):947–982, 2017.

[64] Kaushik Mallik, Anne-Kathrin Schmuck, and Rupak Majumdar. Compositional synthesis of finite state abstractions. *IEEE Trans. Automat. Control*, 64(6):2629–2636, June 2019.

[65] Kaushik Mallik, Sadegh Esmaeil Zadeh Soudjani, Anne-Kathrin Schmuck, and Rupak Majumdar. Compositional construction of finite state abstractions for stochastic control systems. In *Proc. 56th IEEE Conf. Decision and Control (CDC), Melbourne, Australia, 12-15 December 2017*, pages 550–557, 2017.

[66] David Q. Mayne and Eric C. Kerrigan. Tube-based robust nonlinear model predictive control1. *IFAC Proceedings Volumes*, 40(12):36–41, 2007. 7th IFAC Symposium on Nonlinear Control Systems.

[67] Pierre-Jean Meyer, Antoine Girard, and Emmanuel Witrant. Compositional abstraction and safety synthesis using overlapping symbolic models. *IEEE Trans. Automat. Control*, 63(6):1835–1841, 2018.

[68] R. Milner. *Communication and Concurrency*. Prentice-Hall, Inc., USA, 1989.

[69] Thomas Moor, Jörg Raisch, and Siu O'Young. Discrete supervisory control of hybrid systems based on l-complete approximations. *Discrete Event Dynamic Systems*, 12:83–107, 01 2002.

[70] Sebti Mouelhi, Antoine Girard, and Gregor Gössler. CoSyMA: A tool for controller synthesis using multi-scale abstractions. In *Proc. 16th Intl. Conf. Hybrid Systems: Computation and Control (HSCC), Philadelphia, PA, U.S.A., April 8-11, 2013* [2], pages 83–88.

[71] Petter Nilsson and Necmiye Ozay. Control synthesis for large collections of systems with mode-counting constraints. In *Proc. 19th Intl. Conf. Hybrid Systems: Computation and Control (HSCC), Vienna, Austria, April 12-14, 2016*, pages 205–214, 2016.

[72] Chong-Jin Ong and Elmer G. Gilbert. The minimal disturbance invariant set: Outer approximations via its partial sums. *Automatica*, 42(9):1563–1568, 2006.

[73] George Osipenko. *Dynamical systems, graphs, and algorithms*, volume 1889 of *Lect. Notes Math.* Springer-Verlag, Berlin, 2007.

[74] Michael W. Otte and Emilio Frazzoli. Rrtx: Real-time motion planning/replanning for environments with unpredictable obstacles. In *Workshop on the Algorithmic Foundations of Robotics*, 2014.

[75] Samir Palnitkar. *Verilog HDL: A Guide to Digital Design and Synthesis.* Prentice Hall PTR, 2nd edition, 2003.

[76] David Michael Ritchie Park. Concurrency and automata on infinite sequences. In *Theoretical Computer Science*, 1981.

[77] Vern Paxson, Will Estes, and John Millaway. *Lexical Analysis With Flex.* https://github.com/westes/flex/.

[78] Thomas Pecsvaradi and Kumpati S. Narendra. Reachable sets for linear dynamical systems. *Information and Control*, 19(4):319–344, 1971.

[79] Giordano Pola, Antoine Girard, and Paulo Tabuada. Approximately bisimilar symbolic models for nonlinear control systems. *Automatica J. IFAC*, 44(10):2508–2516, 2008.

[80] Giordano Pola, Pierdomenico Pepe, and Maria Domenica Di Benedetto. Symbolic models for networks of discrete-time nonlinear control systems. In *Proc. American Control Conference (ACC), Portland, OR, U.S.A., 4-6 June 2014*, pages 1787–1792, 2014.

[81] Giordano Pola and Paulo Tabuada. Symbolic models for nonlinear control systems: alternating approximate bisimulations. *SIAM J. Control Optim.*, 48(2):719–733, 2009.

[82] Saca V. Rakovic and Mirko Fiacchini. Invariant approximations of the maximal invariant set or "encircling the square". 07 2008.

[83] Sasa V. Rakovic and Miroslav Baric. Parameterized robust control invariant sets for linear systems: Theoretical advances and computational remarks. *IEEE Transactions on Automatic Control*, 55(7):1599–1614, 2010.

[84] S.V. Rakovic, P. Grieder, M. Kvasnica, D.Q. Mayne, and M. Morari. Computation of invariant sets for piecewise affine discrete time systems subject to bounded disturbances. In *2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601)*, volume 2, pages 1418–1423 Vol.2, 2004.

[85] Gunther Reißig. Abstraction based solution of complex attainability problems for decomposable continuous plants. In *Proc. 49th IEEE Conf. Decision and Control (CDC), Atlanta, GA, U.S.A., 15-17 December 2010*, pages 5911–5917, 2010.

[86] Gunther Reissig and Matthias Rungger. Symbolic optimal control. *IEEE Trans. Automat. Control*, 64(6):2224–2239, June 2019.

[87] Gunther Reissig, Alexander Weber, and Elisei Macoveiciuc. *ABS – A Software for Abstraction-based Controller Synthesis*. Bundeswehr University Munich, May 2022. User's Manual. v.2.

[88] Gunther Reissig, Alexander Weber, and Matthias Rungger. Feedback refinement relations for the synthesis of symbolic controllers. *IEEE Trans. Automat. Control*, 62(4):1781–1796, April 2017.

[89] Gunther Reissig, Alexander Weber, and Hao Zhou. *ABS – A Software for Abstraction-based Controller Synthesis*. Bundeswehr University Munich, May 2022. Programmer's Manual. v.2.

[90] Nathalie Revol and Fabrice Rouillier. Motivations for an arbitrary precision interval arithmetic and the MPFI library. *Reliab. Comput.*, 11(4):275–290, 2005.

[91] Matteo Rubagotti, Davide Martino Raimondo, Antonella Ferrara, and Lalo Magni. Robust model predictive control with integral sliding mode in continuous-time sampled-data nonlinear systems. *IEEE Transactions on Automatic Control*, 56(3):556–570, 2011.

[92] Daniel Rubin, Hoai-Nam Nguyen, and Per-Olof Gutman. Computation of polyhedral positive invariant sets via linear matrix inequalities. In *2018 European Control Conference (ECC)*, pages 2941–2946, 2018.

[93] Matthias Rungger, Manuel Mazo, and Paulo Tabuada. Specification-guided controller synthesis for linear systems and safe linear-time temporal logic. In *Proc. 16th Intl. Conf. Hybrid Systems: Computation and Control (HSCC), Philadelphia, PA, U.S.A., April 8-11, 2013* [2], pages 333–342.

[94] Matthias Rungger and Olaf Stursberg. On-the-fly model abstraction for controller synthesis. In *Proc. American Control Conference (ACC), Montréal, Canada, 27-29 June 2012*, pages 2645–2650, 2012.

[95] Matthias Rungger and Majid Zamani. SCOTS: A tool for the synthesis of symbolic controllers. In *Proc. 19th Intl. Conf. Hybrid Systems: Computation and Control (HSCC), Vienna, Austria, April 12-14, 2016* [71], pages 99–104.

[96] Adnane Saoud, Elena Ivanova, and Antoine Girard. Efficient synthesis for monotone transition systems and directed safety specifications. In *Proc. 58th IEEE Conf. Decision and Control (CDC), Nice, France, 11-13 December 2019*, pages 6255–6260, 2019.

[97] Jochen Schröder. *Modelling, state observation and diagnosis of quantised systems*, volume 282 of *Lect. Notes Control Inform. Sciences*. Springer-Verlag, Berlin, 2003.

[98] Maria Seron, Sorin Olaru, Florin Stoican, Jose Dona, and Ernesto Kofman. On finitely determined minimal robust positively invariant sets. pages 157–162, 11 2019.

[99] Sumeet Singh, Anirudha Majumdar, Jean-Jacques Slotine, and Marco Pavone. Robust online motion planning via contraction theory and convex optimization. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5883–5890, 2017.

[100] Vladimir Sinyakov and Antoine Girard. Abstraction of monotone systems based on feedback controllers. In *Proc. 21st IFAC World Congress, Berlin, Germany, July 12-17, 2020*, volume 53 of *IFAC-PapersOnLine*, pages 1819–1824, 2020.

[101] Florin Stoican, Sorin Olaru, José A. De Dona, and María M. Seron. Zonotopic ultimate bounds for linear systems with bounded disturbances. *IFAC Proceedings Volumes*, 44(1):9224–9229, 2011. 18th IFAC World Congress.

[102] Paulo Tabuada. Symbolic models for control systems. *Acta Informatica*, 43:477–500, 01 2007.

[103] Paulo Tabuada. *Verification and control of hybrid systems*. Springer, 2009.

[104] Paulo Tabuada and George J. Pappas. Linear time logic control of discrete-time linear systems. *IEEE Trans. Automat. Control*, 51(12):1862–1877, 2006.

[105] Furqan Tahir and Imad M. Jaimoukha. Low-complexity polytopic invariant sets for linear systems subject to norm-bounded uncertainty. *IEEE Transactions on Automatic Control*, 60(5):1416–1421, 2015.

[106] Paul Trodden. A one-step approach to computing a polytopic robust positively invariant set. *IEEE Transactions on Automatic Control*, 61, 04 2015.

[107] Dimitri van Heesch. *Doxygen Manual*. http://www.stack.nl/~dimitri/doxygen/index.html.

[108] Mario Vasak, Mato Baoti, and Nedjeljko Peric. Efficient computation of the one-step robust sets for piecewise affine systems with polytopic additive uncertainties. *2007 European Control Conference, ECC 2007*, 03 2015.

[109] P. K. C. Wang. A method for approximating dynamical processes by finite-state systems. *Internat. J. Control, I. Ser.*, 8:285–296, 1968.

[110] Alexander Weber, Florian Fiege, and Alexander Knoll. Vehicle mission guidance by symbolic optimal control. In *2022 European Control Conference (ECC)*, pages 1243–1249, 2022.

[111] Alexander Weber and Alexander Knoll. On the solution of the travelling salesman problem for nonlinear salesman dynamics using symbolic optimal control. In *2021 European Control Conference (ECC)*, pages 1995–2001, 2021.

[112] Alexander Weber, Marcus Kreuzer, and Alexander Knoll. A generalized Bellman-Ford algorithm for application in symbolic optimal control. In *18th European Control Conference (ECC), Virtual Event, Russia, May 12-15, 2020*, pages 2007–2014, 2020.

[113] Alexander Weber, Elisei Macoveiciuc, and Gunther Reissig. ABS: A formally correct software tool for space-efficient symbolic synthesis. In *Proc. 25th ACM Intl. Conf. on Hybrid Systems: Computation and Control (HSCC), Milan, Italy, May 4-6, 2022*, 2022.

[114] Alexander Weber, Matthias Rungger, and Gunther Reissig. Optimized state space grids for abstractions. *IEEE Trans. Automat. Control*, 62(11):5816–5821, November 2017.

[115] Marios Xanthidis, Joel M. Esposito, Ioannis Rekleitis, and Jason M. O'Kane. Motion planning by sampling in subspaces of progressively increasing dimension. 100(3–4):777–789, dec 2020.

[116] Mingsheng Ying and Martin Wirsing. Approximate bisimilarity. pages 309–322, 05 2000.

[117] Boyan Yordanov, Jana Tumová, Ivana Černá, Jiří Barnat, and Calin Belta. Formal analysis of piecewise affine systems through formula-guided refinement. *Automatica J. IFAC*, 49(1):261–266, 2013.

[118] Majid Zamani, Alessandro Abate, and Antoine Girard. Symbolic models for stochastic switched systems: a discretization and a discretization-free approach. *Automatica J. IFAC*, 55:183–196, 2015.

[119] Majid Zamani, Giordano Pola, Manuel Mazo, and Paulo Tabuada. Symbolic models for nonlinear control systems without stability assumptions. *IEEE Transactions on Automatic Control*, 57(7):1804–1809, 2012.

[120] Majid Zamani, Giordano Pola, Manuel Mazo, Jr., and Paulo Tabuada. Symbolic models for nonlinear control systems without stability assumptions. *IEEE Trans. Automat. Control*, 57(7):1804–1809, 2012.

[121] Tianze Zhang and Xin Huo. Path planning and control of a quadrotor uav: A symbolic approach. In *2017 11th Asian Control Conference (ASCC)*, pages 2750–2755, 2017.