Refinement for Game-Based Abstractions of Continuous-Space Linear Stochastic Systems

Christopher Polster

Master's Thesis

Computational Science and Engineering

Technische Universität München

Department of Informatics

Refinement for Game-Based Abstractions of Continuous-Space Linear Stochastic Systems

Master's Thesis

Computational Science and Engineering

Technische Universität München

Department of Informatics

Author	Christopher Polster
1st Examiner	Univ.-Prof. Dr. Jan Křetínský Chair for Foundations of Software Reliability and Theoretical Computer Science
2nd Examiner	Univ.-Prof. Dr.-Ing. Matthias Althoff Chair of Robotics, Artificial Intelligence and Real-Time Systems
Submission Date	2019-05-13

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

Ich versichere hiermit, dass ich die von mir eingereichte Abschlussarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

München, 2019-05-13

Christopher Polster

Table of Contents

Abstract

The verification of a hybrid system is considered: Compute the set of initial states of a discrete-time linear stochastic system such that there exists a control strategy which guarantees that all traces starting from this set fulfill a temporal logic specification from the GR(1) fragment of linear temporal logic almost-surely. This problem has previously been posed and solved by Svoreňová et al. (2017). Their approach utilizes a turn-based 2-player probabilistic game abstraction which is analysed with a model checking procedure and iteratively refined based on the analysis results and dynamics of the system.

Starting from this procedure, refinement methods are developed that extend the capabilities and effectiveness of those introduced by Svoreňová et al. (2017). Holistic refinement procedures are proposed that match problematic patterns in the abstraction and break them up through refinement of the state space partition guided by the system dynamics. Based on robust dynamics and the decomposition of a system into a series of co-safe reachability problems induced by the transitions of the objective automaton, a multi-step refinement framework is set up. The framework is configurable and can adapt to the requirements of specific problems.

The developed refinement techniques are applied in two case studies, a reachability problem and a system with a more complex recurrence objective. The cases show that that the robust framework is able to significantly reduce the size of the state space partition compared to previous methods, thereby improving the performance of the verification procedure. Its multi-step nature is able to ensure progress also for rich specifications.

Finally, a browser-based application for interactive visualization and exploration of the considered problem and its solution scheme is presented. It serves as an educational tool and experimentation platform and was used in the development of refinement procedures.

Introduction

When model checking was introduced by Edmund M. Clarke and E. Allen Emerson in the early 1980s, it provided an automated and practical alternative to the manual and proof-theoretic reasoning methods used to verify software until then (Emerson 2008). The novel combination of finite, state-based models and specifications given in a temporal logic enabled the verification of concurrent, "reactive" programs. Today, model checking is a mature formal verification procedure, although its industrial adoption is slower than the pace of research activity surrounding it (Bennion and Habli 2014). The commonly encountered state space explosion continues to be an issue but increasing computing power as well as the development of simplification techniques have brought even large and complex systems into reach. Because of its general and therefore flexible approach, model checking has found new areas of application since its inception. Its correct-by-design philosophy and synthesis capabilities in particular are relied upon by the computer science and control engineering communities for the verification of cyber-physical systems and construction of controllers (Ehlers et al. 2017; Balkan et al. 2018).

1 Model Checking of Hybrid Systems

Systems that combine interacting discrete and continuous domains are called hybrid systems. Van Der Schaft and Schumacher (2000) remarked at the start of the 21st century that the area of hybrid systems was "still largely unexplored", but two decades later Lin and Antsaklis (2014) introduce hybrid systems by observing that they "have recently been at the center of intense research activity". The reason that this interest has emerged is the broad, multidisciplinary application spectrum of the hybrid system framework and its many related theoretical research questions. Control problems of hybrid systems arise for example in robotics, where a robot has to complete discrete tasks while moving in a continuous environment, autonomous driving, where vehicle trajectories have to satisfy safety specifications, or the modelling and supervision of chemical and biological processes which exhibit threshold behaviour.

The probably most often used example of a simple hybrid system is a heater controlled by a thermostat. The heater works in a discrete domain and its state is either on or off. The thermostat measures temperature which is a continuous, real-valued variable. Both are coupled by discrete if-then-else rules: if the observed temperature falls below some threshold, the thermostat activates the heater. Once the desired temperature is reached, the heater is turned off.

There are two fundamental problems associated with the control of hybrid systems: The verification or analysis problem is to determine from which initial states the system evolution can satisfy a specification. The synthesis problem asks for the construction of

a control strategy such that trajectories of the system state controlled by this strategy satisfy a specification. Both problems are linked and typically solved together, i.e. the system analysis provides the information necessary to synthesize satisfying control strategies. Model checking approaches have been developed that solve these problems for some classes of hybrid systems.

The category of hybrid systems encompasses a variety of characteristics that can be combined to form a specific problem. The system dynamics governs the evolution of trajectories, i.e. how the state of a specific problem instance evolves in time. This evolution is generally modelled by differential equations in contexts with continuous time or difference equations when time is discrete. Linear and piece-wise linear (also known as piece-wise affine) dynamics are commonly found due to their simplicity and invertibility but non-linear dynamics can be encountered in the general case. Aside from deterministic evolution, the dynamics can also prescribe a probabilistic evolution in environments with uncertainty. Control of the system is exerted by manipulating the dynamics, e.g. through a control term in the evolution equation or by switching between multiple admissible system evolutions. In unknown or changing environments, the controller may not be able to observe the entire system state which introduces unknowns that have to be accounted for by solution approaches.

The verification and synthesis problems require a specification or an objective that define which behaviour is considered "good" or "bad". Such objectives can be basic properties like reachability or safety but also rich specifications, usually expressed in a temporal logic. The temporal logic is chosen according to the desired properties of the specification and the characteristics of the system under consideration. Objectives can be qualitative or quantitative with respect to probabilistic aspects or reward schemes of the system. It may also be desirable to account for changes in objectives with time, e.g. due to external requirements. In such cases, solutions must be able to dynamically adjust the strategy while the system is evolving.

Model checking approaches have to be adapted to the specific characteristics of a given problem. In general, a model checking procedure will construct a discretization of the continuous domain of the hybrid system such that the essential behaviour of the system is captured. The concrete discrete abstraction model depends on the system properties and specification. Commonly encountered are automata, Markov models and (probabilistic) games. The system abstraction is then synchronized with the specification, which is first translated to an automaton. The resulting finite system model is verified and used to synthesize discrete controllers. Postprocessing of these discrete controllers can be applied to obtain continuous feedback controllers.

System abstraction and the synchronization of system and specifications generally cause an explosion of the size of the state space. This makes the verification and synthesis procedures computationally demanding and is a major challenge for hybrid system analysis.

Restrictions can be imposed on the expressivity of the temporal logic to reduce the complexity of the specification automaton and verification procedure. Iterative refinement techniques aim to generate small partitions of the continuous domain based on the system dynamics and therefore limit the extent of the state space explosion.

Verification and synthesis problems for hybrid systems have been developed and applied for example in the following works and the references therein: Kloetzer and Belta (2008) consider verification and control strategy synthesis for continuous-time linear system and X-free linear temporal logic (LTL). Yordanov and Belta (2010) and Yordanov et al. (2012) deal with discrete-time piecewise-affine systems and LTL specifications, using iterative refinement and conservative analysis of transition system abstractions, respectively. Markov set-chain abstractions for uncontrolled discrete-time, stochastic dynamics are investigated by Abate et al. (2011). Hahn et al. (2011) construct game abstractions with iterative refinement for probabilistic hybrid automata with discrete probability distributions. Aydin Gol et al. (2014, 2015) develop an abstraction refinement technique for the verification of deterministic, discrete-time linear systems with respect to co-safe LTL specifications and derive optimal model predictive control strategies with a cost-minimization approach. Discrete-time, continuous-space switched systems are considered by Lahijanian et al. (2015) who solve the verification and synthesis problems for specifications given in probabilistic computation tree logic with iteratively refined interval-valued Markov chain and bounded parameter Markov decision process abstractions.

2 Scope of This Work

Svoreňová et al. (2017) introduce a procedure to solve the verification and synthesis problems for the class of discrete-time, continuous-space and -control, linear stochastic systems and specifications from the General Reactivity(1) fragment of linear temporal logic which have to be satisfied with probability 1, i.e. almost-surely. The system is verified based on a 2-player probabilistic game abstraction of both the state- and control space. This abstraction is iteratively refined based on information obtained from the system dynamics. Satisfying controllers can be synthesized from the game once results from the verification are available.

Their solution is related to work for other classes of hybrid systems. Yordanov et al. (2012) use a game-based abstraction for systems with deterministic piece-wise linear dynamics and LTL specifications but do not implement abstraction refinement. Aydin Gol et al. (2014) present a dynamics-driven refinement procedure for deterministic linear systems. The abstraction of probability distributions is studied by Abate et al. (2011) for uncontrolled stochastic hybrid systems. Probabilistic switched (i.e. discretely controlled) systems and PCTL specifications are considered by Lahijanian et al. (2015) who develop an iterative refinement technique for a bounded-parameter Markov decision process abstraction.

In a case study, Svoreňová et al. (2017) apply their procedure to a reachability problem. Their refinement heuristics is able to generate a (partial) solution for the problem but the method proves to be computationally demanding. In this work, further refinement techniques are developed. The goal is to set up a refinement framework that achieves progress even for complex specifications and produces small state space partitions without causing excessive state space explosion. Reducing the size of the state space partition reduces the size of the game abstraction, which scales exponentially with the number of partition elements, and therefore reduces the computational demands of the procedure. The development of refinement techniques is supported by the implementation of an interactive visualization of the abstraction-analysis-refinement procedure. This visualization is used as a platform for the design of refinement heuristics and can serve as an educational tool after this work has concluded.

The chapters are structured as follows: First, the fundamental concepts required to express the topics of this work are introduced. The problem setup and solution procedure of Svoreňová et al. (2017) are then reviewed in detail. The developed refinement methods are introduced subsequently as well as the visualization tool. Finally, two case studies are carried out in which the performance of the refinement methods is assessed and compared.

Preliminaries

This chapter introduces the fundamental concepts and notation needed to express the content of this work. The introductions are kept relatively brief. The interested reader is invited to consult the references in the text for a more extensive treatment of the presented topics. In particular, the book of Baier and Katoen (2008) is an excellent resource for everything related to model checking and Baotić (2009) provides a good introduction to polytopic geometry for control problems.

3 Model Checking

Model checking is a formal technique for the automatic verification of a system model with respect to a given specification (Baier and Katoen 2008). Its general field of application is the verification of soft- and hardware systems with respect to specifications, typically formulated in a temporal logic. Specifications may concern qualitative (e.g. "can something bad happen?") or quantitative (e.g. "can something bad happen with more than 20% probability in the first hour of operation?") questions about the system.

The process of model checking starts with the construction of a system model. A system under consideration has to be transformed into a finite and unambiguous representation which reproduces its essential behaviour and provides the information required to verify the specification. A model checking procedure will then explore this system representation exhaustively by brute-force, considering every possible state of the system model, looking for counterexamples that violate the specification. This approach ensures that no malicious behaviour of the system is missed and the produced counterexamples can be used to guide the search for system or system model errors (e.g. Clarke et al. 2000). However, the exhaustive approach also often means that substantial resources are required to carry out the verification. Systems with rich behaviour require complex models for accurate representation, leading to a large number of possible states that need to be checked by the procedure. Dealing with this so-called state space explosion is a major challenge when designing a model checking procedure. Along with the continuous increase in available computing power which allows for the verification of increasingly complex systems, techniques such as partial order reduction and abstraction refinement have been and are developed to combat the state space explosion problem (Baier and Katoen 2008).

4 Transition Systems and Abstractions

Transition systems are ubiquitously used to represent system models in model checking applications (Baier and Katoen 2008). A basic transition system consists of a finite set of states which are combined into a directed graph by a transition relation that defines the

edges between states. An initial state or a set of initial states is usually defined to provide an entry point for the system. The transition relation governs the evolution of trajectories in the system from state to state. It can be deterministic, non-deterministic, probabilistic or combine different kind of transitions in the same model. States and transitions can be associated with additional properties with a labelling function. This extensibility of transition systems means they can be adapted to represent a variety of systems, e.g. finite automata or turn-based games. Transition systems are also composable and support operations such as the synchronous product, which enforces the synchronous evolution of two transition systems and is particularly important in verification for bringing together a system model and the specification it is verified against.

Transition systems are well suited as the foundation of system models for model checking purposes. However, many systems have state spaces so large they cannot reasonably be handled by a verification procedure. Some systems even have infinite state spaces. It is possible to deal with such problems by introducing an abstraction, a grouping of states of the original system into a finite set of equivalence classes based on some common properties. The grouping should ideally be minimal in the number of equivalence classes while still capturing the essential behaviour of the system which is required for the verification. Unfortunately, the knowledge that is necessary to construct such an abstraction is often not available a-priori. Counterexample-guided abstraction refinement (CEGAR, Clarke et al. 2000) provides a way out of this predicament. With CEGAR techniques, a given initial abstraction is iteratively refined by removing problematic behaviour based on feedback from the verification procedure until a suitable system model has been found.

5 Hybrid Systems

Hybrid systems combine discrete events with continuous-variable dynamics, allowing heterogeneous behaviour to emerge. It is essential for every hybrid system that its discrete- and continuous domains not just coexist but interact with each other. The discrete domain is most often governed by a set of events resulting in threshold- or switching behaviour of the system. Continuous variables are usually associated with physical processes such as mechanical phase space, temperature or electrical current. The concept of time for a hybrid system can be both discrete or continuous, leading to a description of the dynamics with difference- or differential equations, respectively. Because hybrid systems can be assembled from such a variety of components, it is difficult to find a general definition. In general, solution approaches to problems involving hybrid systems revolve around the synchronous evolution of the discrete and continuous domain, formalized by so-called hybrid automata.

5.1 Linear Stochastic Systems

A linear stochastic system (LSS) $\mathcal{T}$ is a discrete-time, continuous-space stochastic process

governed by the evolution equation

$$\boldsymbol{x}_{t+1} = \boldsymbol{A}\boldsymbol{x}_t + \boldsymbol{B}\boldsymbol{u}_t + \boldsymbol{w}_t \ , \tag{1}$$

where $\boldsymbol{x}_t \in X \subset \mathbb{R}^n$ is the state of a trace $\boldsymbol{x} = \boldsymbol{x}_0\boldsymbol{x}_1...$ at time $t \in \mathbb{N}_0$, $\boldsymbol{u}_t \in U \subset \mathbb{R}^m$ is a control input and $\boldsymbol{w}_t \in W \subset \mathbb{R}^n$ is a random perturbation. The state space X, random space W and control space U are bounded subsets of Eucledian, real-valued vector spaces of dimensions n, n and m, respectively. The probability distribution from which the random vector is sampled at each timestep is assumed to have non-zero density everywhere in W for all times t.

A trace through the linear stochastic system evolves according to matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ which transforms the current state $\boldsymbol{x}_t$, a stochastic perturbation $\boldsymbol{w}_t$ and external control in the form of a control vector $\boldsymbol{u}_t$, projected into the state space by matrix $\boldsymbol{B} \in \mathbb{R}^{n\times m}$. The control inputs are chosen for every time step by a strategy $S_{\mathcal{T}} : X^+ \to U$, where X^+ denotes the set of all non-empty finite sequences of elements of X. A strategy may therefore take the evolution of a trace up to its current state into account when selecting the control vector. If a strategy depends only on the current state it is called memoryless, otherwise it is a finite-memory strategy. To reason about properties of a trace, it is subjected to discrete events, e.g. entering or exiting some region of the state space.

5.2 Related Systems

In the case where $W = \{\boldsymbol{c}\}$, $\boldsymbol{c} \in \mathbb{R}^n$, the LSS is stripped of its stochastic dynamics and degenerates to a deterministic linear system. Linear systems in general are a special case of piece-wise affine systems, also known as piece-wise linear systems, whose state space is partitioned into multiple disjunct regions, called modes, each with their own evolution equation

$$\boldsymbol{x}_{t+1} = \boldsymbol{A}_l\boldsymbol{x}_t + \boldsymbol{B}_l\boldsymbol{u}_t + \boldsymbol{w}_{l,t} \ ,$$

where l is an index enumerating the modes. Traces in piece-wise affine systems evolve according the evolution equation associated with the mode that the current state of the trace is a member of. The non-stochastic variant of piece-wise affine systems where $W_l = \{\boldsymbol{c}_l\}$ for all modes l was used e.g. by Yordanov and Belta (2009).

More generally, evolution equations of the form

$$\boldsymbol{x}_{t+1} = F(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{w}_t)$$

can accommodate any kind of non-linear dynamics. In (stochastic) switched systems, strategies do not control traces by choosing the value of a control vector for each step, but by choosing the entire form of the dynamics

$$\boldsymbol{x}_{t+1} = U_t(\boldsymbol{x}_t, \boldsymbol{w}_t) \ .$$

The choice of $U_t : X \times W \to X$ is usually restricted to a finite set of available dynamics between which the controller can switch at each timestep. This kind of system was used for example by Lahijanian et al. (2015).

The systems presented so far have used discrete time and difference equations to describe the evolution of traces. But the dynamics can also be entirely continuous, with traces evolving in time according to differential equations. Continuous-time, countinuous-space systems can still have hybrid characteristics when equipped with discrete events or if controlled in a discrete manner (analogous to switched systems).

6 Convex Geometry

In order to describe events in and generate finite abstractions of continuous space, a discrete representation of that space is necessary. Discretizations in the framework of convex polytopic geometry are popular for this purpose as they have many advantageous properties. Convex geometry can be applied to problems of any dimension, has well understood computational properties due to its roots in linear optimization and mature libraries such as MPT (Herceg et al. 2013), TuLiP (Filippidis et al. 2016) and cdd (Fukuda 2019) are freely available. For control problems involving (piece-wise) linear dynamics, convex geometry is a commonly used and established tool (Baotić 2009).

6.1 Convex Polytope Representations

A (closed) halfspace $H \subset \mathbb{R}^n$ is the set of points

$$H = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{k} \cdot \boldsymbol{x} \leq c\}$$

that fulfill a linear inequality governed by a normal vector $\boldsymbol{k} \in \mathbb{R}^n$, $\boldsymbol{k} \neq \boldsymbol{0}$, and an offset $c \in \mathbb{R}$. The normal vector $\boldsymbol{k}$ is pointing away from the halfspace set and for convenience and without restriction of generality it is always assumed that its length is normalized such that $\|\boldsymbol{k}\| = 1$. Due to practical limitations of floating-point number representation on a computer, no distinction between closed and open halfspaces is made in this work and the closed form is generally used in the text.

The set of points bounded by an intersection of halfspaces $\{H_j\}_{j \in J}$ is a convex polytope P and can be written as

$$P = \bigcup_{j \in J} H_j = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{K}\boldsymbol{x} \leq \boldsymbol{c}\} \ ,$$

where $\boldsymbol{K}$ is the stack of transposed normal vectors $\boldsymbol{k}_j$ of the halfspaces, $\boldsymbol{c}$ is the corresponding stack of offset values c_j and the inequation holds component-wise. If the set of halfspaces is minimal, i.e. no halfspace can be removed without changing the bounded

region, the representation is called the H-representation of the convex polytope. Redundant halfspaces can be identified by solving a series of linear programs (Baotić 2009).

A convex polytope P can alternatively be defined as the convex hull

$$P = \mathrm{Hull}(X) = \left\{ \sum_{i \in I} \lambda_i x_i \;\middle|\; \forall i : \lambda_i \in [\,0,\,1\,], \sum_{i \in I} \lambda_i = 1 \right\}$$

of a set of points $X = \{x_i\}_{i \in I} \subset \mathbb{R}^n$. The vertex set of P is the smallest set of points $\mathrm{Vert}(P) \subset \mathbb{R}^n$ such that $P = \mathrm{Hull}(\mathrm{Vert}(P))$. It uniquely defines the so-called V-representation of the convex polytope.

In this work, "polytope" will always refer to a convex polytope and all polytopes mentioned are assumed to be convex unless otherwise stated. A polytope $P \subseteq \mathbb{R}^n$ is called full-dimensional if an n-dimensional ball of radius $\epsilon > 0$ exists that is a subset of P. All polytopes that are not full-dimensional (e.g. a line segment in $\mathbb{R}^2$) are treated as if they were empty sets.

6.2 Operations on Convex Polytopes

Convex polytopes have advantageous properties making them a popular choice for practical problems in computational geometry. Operations on convex polytopes are often closed, i.e. their result is again a convex polytope. They are also often very easy to express if the right representation is chosen. For example, the intersection of two convex polytopes is always a convex polytope and can easily be computed from the polytopes' H-representations by merging the sets of bounding halfspaces and reducing to minimal form. The transformations between the representations are called the vertex enumeration problem (H- to V-representation) and facet enumeration problem (V- to H-representation).

The following linear transformations of a convex polytope $X \subset \mathbb{R}^n$ are defined: application of a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ from the left and translation by a vector $\boldsymbol{v} \in \mathbb{R}^m$

$$\boldsymbol{A}X + \boldsymbol{v} \coloneqq \{\boldsymbol{y} \in \mathbb{R}^m \mid \exists \boldsymbol{x} \in X : \boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{v}\}$$

and application of a matrix $B \in \mathbb{R}^{n \times n}$ from the right

$$X\boldsymbol{B} \coloneqq \{\boldsymbol{y} \in \mathbb{R}^m \mid \boldsymbol{K}\boldsymbol{B}\boldsymbol{y} \leq \boldsymbol{c}\}\ .$$

The shorthand $-X$ is used to express the operation $(-\mathbb{1})X$, where $\mathbb{1}$ is an identity matrix of appropriate size. As evident from the definitions, matrix application from the left is easily computable with the V-representation, while the H-representation is better suited to compute the result of a matrix application from the right. Note that both operations involve a change in dimension if $n \neq m$.

Two binary operations are defined for convex polytopes $X, Y \subset \mathbb{R}^n$. The Minkowski sum

$$X \oplus Y \coloneqq \{\boldsymbol{z} \in \mathbb{R}^n \mid \exists \boldsymbol{x} \in X, \exists \boldsymbol{y} \in Y : \boldsymbol{z} = \boldsymbol{x} + \boldsymbol{y}\}$$

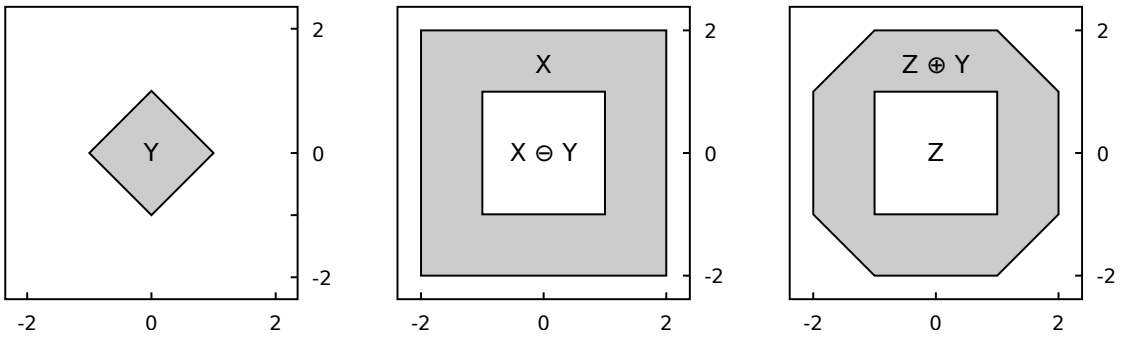


Figure 1: Illustration of the Pontryagin difference (center) and Minkowski sum (right). Note that $(X \ominus Y) \oplus Y = Z \oplus Y \neq X$. Adapted from Figures 7 and 8 of Baotić (2009).

can be computed by translating every vertex of X with every vertex of Y and then taking the convex hull of the resulting set of points. The Pontryagin difference

$$X \ominus Y := \{\boldsymbol{z} \in \mathbb{R}^n \mid \forall \boldsymbol{y} \in Y : \boldsymbol{z} + \boldsymbol{y} \in X\}$$

can be computed by translating every halfspace of X by every vertex of Y and then taking the intersection of these halfspaces. While the Minkowski sum is commutative, the Pontryagin difference is not. Minkowski sum and Pontryagin difference are not inverse operations. In general it only holds that

$$(X \ominus Y) \oplus Y \subseteq X$$

(Baotić 2009). This is illustrated in Figure 1, where a concrete counterexample demonstrates that Minkowski sum cannot generally invert a Pontryagin difference operation.

6.3 Non-convex Polytopic Regions

Not every problem encountered in practice is conveniently convex. However, any non-convex polytopic region can be partitioned into a union of convex polytopes, e.g. by triangulation of the region's surface and subsequent decomposition into simplices. Results from convex polytopic geometry are therefore generally applicable to non-convex polytopic geometry if all non-convex regions are partitioned first. While many operations on convex polytopes can be extended to unions of convex polytopes in a straightforward manner, some require special care.

Intersection can be distributed to the individual convex polytopes by taking the intersection of every polytope of one region with every polytope of the other region and keeping the non-empty intersections as the resulting union. The linear operations matrix application and translation can be applied polytope-wise. Minkowski sum can be distributed to the polytopes, but the result will generally not be a disjunct set of convex polytopes and may require postprocessing if a disjunct set of output polytopes is desired. The Pontryagin difference cannot be distributed to the convex polytopes but it is possible to express it using the Minkowski sum and set difference operations as

$$X \ominus Y = \mathbb{R}^n \setminus ((\mathbb{R}^n \setminus X) \oplus (-Y))$$

(e.g. Rakovic et al. 2004). The set difference of two convex polytopes or two polytopic regions is a non-convex region in general. It can be computed using the regiondiff algorithm of Baotić (2009), which returns the result of the difference operation as a set of disjunct convex polytopes.

7 Markov Models and Probabilistic Games

If a system of interest exhibits stochastic behaviour, there is a need for probabilistic abstractions that can reflect this stochasticity appropriately. Markov models and the closely related probabilistic games are such models for systems with discrete time evolution. A few of these models are introduced in this section.

7.1 Markov Models

Markov models are transition systems enriched with probabilistic behaviour and have found numerous applications, including in probabilistic model checking (e.g. Baier and Katoen 2008; Svoreňová et al. 2013; Chatterjee et al. 2014; Lahijanian et al. 2015).

The simplest Markov model is the Markov chain. A Markov chain is a tuple (G, δ) where G is a set of states and $\delta : G \to \mathcal{D}(G)$ a transition relation with $\mathcal{D}(G)$ denoting the set of all probability distributions over G. With every step of a trace through the Markov chain, a successor is chosen by sampling the probability distribution of successors of the current state defined by δ.

Markov decision processes (MDPs) extend Markov chains by introducing actions. An MDP is a 3-tuple (G, Act, δ), where G is a set of states, Act is a set of actions and $\delta : G \times Act \to \mathcal{D}(G)$ is a probabilistic transition relation. With every step of a trace through the MDP, an action is selected by some decision-making process and the successor state determined by sampling the probability distribution associated with the current state and chosen action through δ. Not all actions are enabled in every state, but at least one action from Act must be. A Markov chain is the degenerate case of an MDP with exactly one action enabled in every state.

The introduction of actions in the MDP opens up a game-theoretic perspective on Markov models. From the perspective of games, the transition system of an MDP is a game graph on which a turn-based game is played between a player and a probabilistic environment. The player chooses their move each turn by picking an enabled action in the current state and the successor state is sampled by the environment from the corresponding probability distribution. MDPs are therefore also called 1-player probabilistic games or 1½-player games, with one "proper" player and the environment in a player-like role. This game-based view of Markov models extends naturally to more complicated behaviour through the introduction of additional players.

7.2 2-Player Probabilistic Games

A two-player probabilistic game, or 2½-player game, is a turn-based probabilistic game played on a game graph

$$\mathcal{G} = (G_1, G_2, Act, \delta) \ ,$$

where G_1, G_2 are disjoint sets of player 1 and 2 states, respectively. Act and δ are defined as they were for the MDP with $G = G_1 \cup G_2$. A play is a sequence of states $g = g_0 g_1 ... \in G^\omega$ such that $g_i \in G_1$ for all even and $g_i \in G_2$ for all odd indices i, i.e. player's turns alternate beginning with player 1. G^ω denotes the set of all infinite sequences of members of G. When it is their turn, players choose an action that is enabled for the current game state from the set of actions Act. The next state of the play is then chosen stochastically based on the probability distribution defined by the probabilistic transition function $\delta : G \times Act \to \mathcal{D}(G)$. Therefore, for every g_i there must be an action $a \in Act$ such that $\delta(g_i, a)(g_{i+1}) > 0$.

A player k strategy is a function $S^k_{\mathcal{G}} : G^+ \to Act$ that determines the action taken after a finite prefix of a play ending in a state of player k. Analogous to strategies for linear stochastic systems, a strategy that requires a finite prefix of a play to determine an action is called finite-memory while a strategy using only the current game state for the action selection is called memoryless.

7.3 Winning and Solving 2-Player Probabilistic Games

A notion of winning is introduced by extending the game graph $\mathcal{G}$ with an acceptance condition $\mathcal{C}$ (also called winning condition) to form the game

$$\mathcal{G}' = (G_1, G_2, Act, \delta, \mathcal{C}) \ .$$

The acceptance condition separates the set of all possible plays into those which are won by player 1 and those plays are won by player 2. Every play has exacly one winning player, no plays are won by neither or both players. Acceptance conditions of games are analogous to those of ω-automata and are discussed further in section 8.2.

The solution of a game are the sets of initial states for which player 1 (2) has a winning strategy, which is a strategy that player 1 (2) can use to ensure winning under some circumstances. The circumstances depend on the type of game analysis that is carried out. In this work, (qualitative) almost-sure analysis for an adversarial and cooperative player 2 is the main concern, i.e. game states are sought for which a player 1 strategy exists that leads to player 1 winning the game with probability 1. In the adversarial setting, the strategy must lead to a player 1 win for every possible strategy of the opponent, while in the cooperative setting an almost-sure winning strategy only has to exist for some strategy of the other player, meaning that player 1 can win if the opponent is "nice" and cooperates.

The set of initial states for which player 1 has an almost-sure winning strategy is denoted in the adversarial setting by $\mathrm{Almost}(\mathcal{G}')$ and in the cooperative setting by $\mathrm{Almost}^{\mathrm{coop}}(\mathcal{G}')$.

8 Languages and Automata

A language is a collection of words, formed from an alphabet of symbols according to a set of rules. Finite automata are transition systems with which such rules can be expressed.

8.1 ω-Automata

A deterministic ω-automaton is a 5-tuple

$$\mathcal{A} = (Q, \Sigma, \delta, q_{\mathrm{init}}, \mathcal{C}) \ ,$$

where $Q \neq \emptyset$ is a finite set of states, Σ is a finite alphabet of symbols, $\delta : Q \times \Sigma \to Q$ is a deterministic transition relation and $q_{\mathrm{init}} \in Q$ is the initial state of the automaton. If the transition relation is defined for every combination of state in Q and symbol in Σ, the automaton is called complete, otherwise it is called incomplete.

A run is an infinite sequence of states $q_0 q_1 ... \in Q^\omega$ such that $q_0 = q_{\mathrm{init}}$ and $q_{i+1} = \delta(q_i, w_i)$, $w_i \in \Sigma$ for all $i \in \mathbb{N}_0$. Because the transitions of $\mathcal{A}$ are deterministic, every run is uniquely associated with a word $w = w_0 w_1 ... \in \Sigma^\omega$ through the transition labels from Σ assigned by δ. The language $\mathcal{L}(\mathcal{A})$ of an automaton $\mathcal{A}$ is defined as the set of all words whose corresponding runs are accepted by the acceptance condition $\mathcal{C}$. Note that a "run" refers to a sequence of automaton states, a "word" to sequence of symbols from an alphabet, a "trace" to a sequence of states of an LSS and a "play" is sequence of states of a game.

8.2 Acceptance Conditions

An acceptance condition $\mathcal{C}$ specifies the set of all accepted runs. If a run is not accepted, it is rejected. The set of all accepted runs is generally an infinite set. The acceptance condition is therefore usually expressed by other means than an enumeration of runs from the accepted set. A Büchi acceptance condition is defined by a set of states of which at least one has to be visited infinitely often in a run or the run is rejected. A Büchi-implication condition, also known as a one-pair Streett condition, is specified by a tuple of sets $\mathcal{C} = (E, F) \subseteq Q \times Q$ and accepts all runs r satisfying

$$(\mathrm{Inf}(r) \cap E \neq \emptyset) \implies (\mathrm{Inf}(r) \cap F \neq \emptyset) \ ,$$

where the set of states which occur infinitely often in the run r is denoted by $\mathrm{Inf}(r)$. I.e. if any state from E occurs infinitely often in a run, a state from F has to occur infinitely often as well otherwise the run is rejected. The condition is trivially fulfilled if E is empty. If F is empty, all runs in which a state from E occurs infinitely often are rejected.

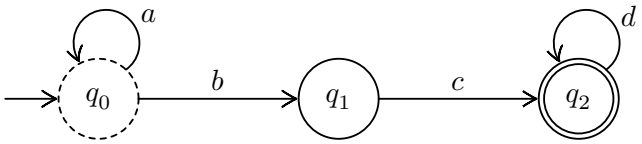


Figure 2: A deterministic, incomplete ω-automaton with a one-pair Streett acceptance condition $(E, F) = (\{q_0\}, \{q_2\})$. States from E are highlighted with a dashed border and states from F with a double border. The language of the automaton consists of all words of the form a^*bcd^ω.

Figure 2 shows the automaton

$$(\{q_0, q_1, q_2\}, \{a, b, c, d\}, \delta, q_0, (\{q_0\}, \{q_2\}))$$

with a one-pair Streett acceptance condition. The transition relation defines $\delta(q_0, a) = q_0$, $\delta(q_0, b) = q_1$, $\delta(q_1, c) = q_2$ and $\delta(q_2, d) = q_2$. The automaton is therefore incomplete. This automaton accepts all runs that stay in q_0 for a finite number of steps before transitioning to q_1 and then immediately to q_2, where they stay forever. The corresponding language of the automaton contains all words starting with a finite number of as, followed by a single b, a single c and then ds forever. Words of this form are written as a^*bcd^ω, where a^* denotes any finite repetition of a and d^ω the infinite repetition of d.

9 Temporal Logic

A temporal logic is an extension of propositional logic by a set of temporal connectives (Baier and Katoen 2008). With these connectives, statements about the current and future state of a system can be expressed. While formulae of propositional logic are evaluated with a single valuation of its atomic propositions, temporal logic formulae are evaluated with a sequence or a tree of valuations that describe the evolution of the system state in time.

9.1 Linear Temporal Logic

Linear temporal logic (LTL) is an extension of propositional logic applied to discrete-time, infinite sequences of valuations of propositional atoms. LTL formulae are able to express the relative temporal order of events occurring in such sequences. As the name suggests, LTL is concerned with a linear, path-based understanding of time where every moment in time has a unique successor.

LTL formulae over a set of atomic propositions AP are formed from the grammar

$$\varphi ::= \mathsf{true} \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \mathsf{X}\varphi \mid \varphi_1 \mathsf{U}\, \varphi_2 \ ,$$

where $a \in AP$ (Baier and Katoen 2008). $\wedge$ and $\neg$ are the known propositional operators "and" and "not". X and U are temporal operators named "next" and "until". true is

the tautology symbol. Operator precedence is adopted from propositional logic where possible, X binds as strongly as $\neg$ and U takes precedence over all propositional binary operators.

LTL formulae are interpreted over words $w = w_0 w_1 ... \in (2^{AP})^\omega$ constructed from an alphabet of valuations of the atomic propositions. The semantics of LTL is expressed by the satisfaction relation $\vDash$, given by

$$
\begin{aligned}
w &\vDash \mathsf{true} && \text{unconditionally (tautology)},\\
w &\vDash a && \iff a \in w_0 \ ,\\
w &\vDash \neg\varphi && \iff w \nvDash \varphi \ ,\\
w &\vDash \varphi_1 \wedge \varphi_2 && \iff w \vDash \varphi_1 \text{ and } w \vDash \varphi_2 \ ,\\
w &\vDash \mathsf{X}\varphi && \iff w_1 w_2 ... \vDash \varphi \ ,\\
w &\vDash \varphi_1 \mathsf{U}\, \varphi_2 && \iff \exists j \geq 0 : w_j w_{j+1} ... \vDash \varphi_2 \text{ and } \forall 0 \leq i < j : w_i w_{i+1} ... \vDash \varphi_1 \ .
\end{aligned}
$$

Other connectives can be derived for convenience, for example

$$
\begin{aligned}
\varphi_1 \vee \varphi_2 &:= \neg(\neg\varphi_1 \wedge \neg\varphi_2) && \text{“or”},\\
\mathsf{F}\varphi &:= \mathsf{true}\, \mathsf{U}\, \varphi && \text{“finally”},\\
\mathsf{G}\varphi &:= \neg\mathsf{F}\neg\varphi && \text{“globally”}.
\end{aligned}
$$

Consider a transition system with states 2^{AP}, such that the set of atomic propositions AP coincides with that of an LTL formula φ. The evolution of this transition system when starting in state s generates words $w = w_0 w_1 ...$, where $w_0 = s$, over the alphabet 2^{AP}. The function

$$\text{Words} : 2^{AP} \to (2^{AP})^\omega$$

associates a state with all words that can be generated by starting in this state and following the transitions of the system. The semantics of LTL is extended to a state-based notion in the following way:

$$s \vDash \varphi \iff \forall w \in \text{Words}(s) : w \vDash \varphi \ .$$

9.2 Other Temporal Logics

Whenever there are multiple futures possible in a moment, the linear understanding of time considers every future individually. In a branching time interpretation, the future is seen as a tree that branches out whenever a moment allows for more than one future. Computation Tree Logic (CTL) is a temporal logic with this understanding of time. CTL can reason over a branching future with the quantifiers $\exists$ (there is some future) and $\forall$

(for all possible futures). While the expressiveness of CTL overlaps with that of LTL, properties exists that can only be expressed in one but not the other (Baier and Katoen 2008).

The understanding of time as linear or branching is not the only concept that influences the construction of a temporal logic. While LTL and CTL are only concerned with the ordering of time, there are also temporal logics that include the concept of a clock. In such logics properties like "in less than 10 time units" or "for the next 10 time units" can be expressed. Quantitative and qualitative properties are of interest when verifying probabilistic systems and systems that associate rewards with certain behaviour. Probabilistic Computation Tree Logic incorporates quantitative measures of certainty with the addition of a probabilistic operator to CTL. The certainty of temporal logic properties can also be quantified on the semantic level by changing the satisfaction relation (Baier and Katoen 2008). The boolean question "Does formula φ hold?" can therefore be transformed into a quantitative inquiry such as "With which probability does formula φ hold?" in the probabilistic setting.

9.3 Automata for LTL Objectives

For every LTL formula, an ω-automaton can be constructed such that the automaton accepts all words that satisfy the formula and rejects all words that do not satisfy the formula. Formula-to-automaton translation is a key component of LTL model checking because it allows to systematically combine the system model and the specification it is verified against. Tools for the automatic translation of LTL formulae to various types of ω-automata have beed developed, e.g. by Křetínský et al. (2018), Duret-Lutz et al. (2016) and Gastin and Oddoux (2001).

Table 1 presents five basic LTL formulae together with corresponding ω-automata. The formulae are specified over words over an alphabet of valuations of the propositional atoms ϕ and θ. Transition labels of the automata are propositional formulas. A transition is taken if the current symbol (i.e. valuation) of the run-inducing word satisfies its associated propositional formula. The chosen acceptance conditions are one-pair Streett tuples for all but the safety automaton which expresses acceptance sufficiently through an incomplete transition relation.

9.4 Fragments of LTL

The complexity of an LTL model checking procedure is governed by the cost of the translation of the specification formula to an automaton and the analysis of the synchronized product of the automaton and system model. The size of a deterministic finite automaton (DFA) corresponding to an LTL formula is doubly exponential in the number of subformulae in the worst case (Baier and Katoen 2008). The worst-case time complexity

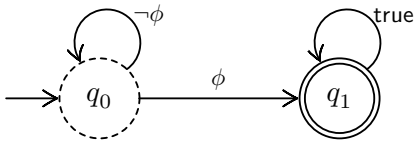
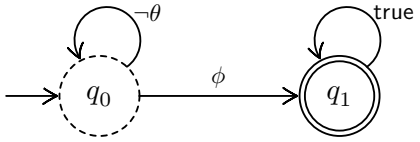
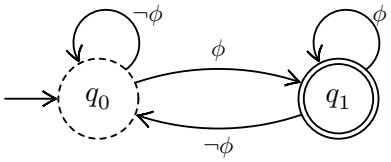
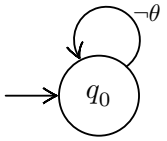
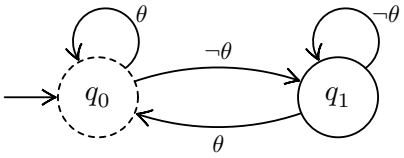
Reachability: $\mathsf{F}\phi$. Eventually satisfy ϕ. One-pair Streett condition: $(\{q_0\}, \{q_1\})$	
Reachability/Avoidance: $\neg\theta \,\mathsf{U}\, \phi$. Avoid satisfying θ until a ϕ is satisfied. One-pair Streett condition: $(\{q_0\}, \{q_1\})$	
Recurrence: $\mathsf{GF}\phi$. Satisfy ϕ again and again. One-pair Streett condition: $(\{q_0\}, \{q_1\})$	
Safety: $\mathsf{G}\neg\theta$. Forever avoid satisfaction of θ. Acceptance expressed through incompleteness of δ.	
Eventual Safety: $\mathsf{FG}\neg\theta$. Eventually, θ must never be satisfied. One-pair Streett condition: $(\{q_0\}, \emptyset)$	

Table 1: Description of five basic linear time objectives and their translations into ω-automata with one-pair Streett acceptance condition. The highlighting of states from the acceptance sets is analogous to Figure 2.

of a DFA-based model checking procedure for LTL is therefore at least doubly exponential in the size of the specification, assuming that the procedure has to visit each state in the product of automaton and system model at least once. Restricting specifications to a fragment of LTL with reduced expressivity can limit the exponential blow-up in the automaton translation or simplify the analysis procedure. If designed carefully, such LTL fragments can still include many properties that are relevant in practical applications.

The class of LTL formulae with form

$$\Big(\bigwedge_{i=1\ldots n} \mathsf{GF}\phi_i \Big) \;\rightarrow\; \Big(\bigwedge_{i=1\ldots m} \mathsf{GF}\mu_i \Big) \,,$$

where ϕ_i and μ_i are propositional formulae, was named General Reactivity(1) (GR(1)) and popularized by Piterman et al. (2005). The complexity of model checking procedures with GR(1) specifications is polynomial in the size of the specification which is a substantial reduction from the doubly exponential complexity of full LTL (Piterman et al. 2005). An extended variant of GR(1) contains all formulae

$$\Big(\bigwedge_{i=1\ldots n} \varphi_i \Big) \;\rightarrow\; \Big(\bigwedge_{i=1\ldots m} \psi_i \Big) \,,$$

where φ_i and ψ_i are LTL formulae representable by deterministic automata with Büchi acceptance condition (Piterman et al. 2005). Model checking procedures for GR(1) are generally compatible with extended GR(1). In both fragments, a deterministic one-pair Streett automaton of polynomial complexity in the size of the formula (GR(1)) or the number of Büchi conditions (extended GR(1)) can be constructed from the specification (Kesten et al. 2005).

LTL formulae are interpreted over words of infinite length. For some formulas, satisfiability of a word can be decided already after a finite prefix. Any infinite continuation of the word after this prefix has no impact on the satisfiability of the formula. A language $\mathcal{L} \subseteq \Sigma^\omega$ over the alphabet Σ is called a co-safety language if and only if every word $w \in \mathcal{L}$ has a prefix $x \in \Sigma^*$ such that $x \cdot y \in \mathcal{L}$ for all $y \in \Sigma^\omega$, where $\cdot$ is the concatenation operator (Kupferman and Vardi 2001). The fragment of co-safe LTL contains all formulae whose language is a co-safety language. A co-safe LTL formula can be identified syntactically: All formulae that can be written in positive normal form (where negation occurs only in front of atomic propositions) and use only the temporal operators X and U are co-safe (Prasad Sistla 1994). Co-safety languages can be represented by deterministic ω-automata with an acceptance condition that requires reaching a state in a set of final states. Model checking with co-safe LTL specifications can therefore be reduced to a reachability analysis problem with respect to these final states (Kupferman and Vardi 2001). Note that every one-pair Streett automaton with $E \cup F = Q$ and $\delta(q, \sigma) = q$ for all $q \in F$ and for all $\sigma \in \Sigma$ defines a co-safety language. Therefore, the reachability and reachability/avoidance objectives in Table 1 are co-safe.

Finally, the class of X-free LTL formulae is introduced, i.e. the class of all LTL formulae in which the X operator does not appear. Two words w and w' over an alphabet Σ are called stuttering equivalent if and only if there exist two integer sequences $i_0 < i_1 < ...$ and $j_0 < j_1 < ...$ and a word $v_0 v_1 ... \in \Sigma^\omega$ such that

$$w = (v_0)^{i_0}(v_1)^{i_1}... \text{ and } w' = (v_0)^{j_0}(v_1)^{j_1}... \ .$$

If a word satisfies a X-free LTL formula, all stuttering equivalent words also satisfy this formula (Baier and Katoen 2008).

Problem Formulation

The problem posed by Svoreňová et al. (2017) is considered: Compute the the set of initial states of a discrete-time linear stochastic system such that there exists a control strategy that guarantees satisfaction of a GR(1) objective specification with probability 1.

10 System Setup

The definition of linear stochastic systems from Section 5.1 is recalled. An LSS is a discrete-time dynamical system, in which traces $\boldsymbol{x} = \boldsymbol{x}_0\boldsymbol{x}_1...$ evolve according to the difference equation

$$\boldsymbol{x}_{t+1} = \boldsymbol{A}\boldsymbol{x}_t + \boldsymbol{B}\boldsymbol{u}_t + \boldsymbol{w}_t \ , \tag{2}$$

where $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, $\boldsymbol{B} \in \mathbb{R}^{n\times m}$. $\boldsymbol{x}_t \in U \subset \mathbb{R}^n$ is the state of the trace at time t, $\boldsymbol{u}_t \in U \subset \mathbb{R}^m$ is a control input and $\boldsymbol{w} \in W \subset \mathbb{R}^n$ is a random perturbation. For convenience, it is assumed that X, U and W are convex polytopes. The random vector distribution is uniform and non-zero everywhere in W. A trace is controlled by a strategy $S_{\mathcal{T}} : X^+ \to U$ that maps finite prefixes of traces to control inputs. Note that such an LSS can be interpreted as a Markov decision process with infinitely many states X and infinitely many actions U.

A set of linear predicates $\Pi = \{\pi_k\}_{k\in K}$ is defined over the state space. Each linear predicate π_k is associated with a halfspace $H_k = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{u}_k \cdot \boldsymbol{x} \leq c_k\}$. The function $\pi : X \to 2^\Pi$ returns the set of fulfilled predicates for a given state vector $\boldsymbol{x}$. When applied element-wise to a trace, π generates a word over the alphabet 2^Π which can be interpreted by a temporal logic specification.

11 Problem Statement

Both the verification and synthesis problems are posed for the linear stochastic system and the specification.

11.1 Satisfiability Analysis

First, the verification of system (2) with respect to a temporal logic objective φ is considered. The goal is to determine for which initial states of the system a control strategy exists such that the objective φ can be fulfilled with probability 1. The result of this almost-sure analysis is the computed set of initial states X_{init}.

The specification φ uses the linear predicates Π as atomic propositions. Objectives are limited to the extended GR(1) fragment of LTL, i.e. all formulas that can be translated

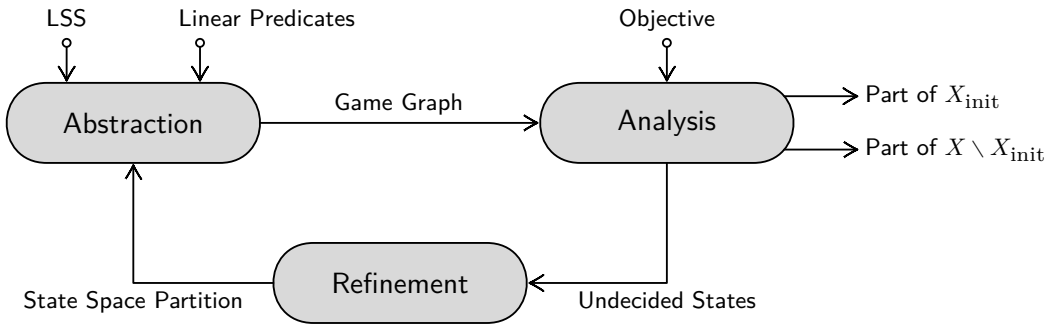


Figure 3: A flowchart representation of the solution approach described in section 12. X_{init} is the set of states for which a controller can guarantee satisfaction of the objective when a trace originates from within. Adapted from Fig. 1 of Svoreňová et al. (2017) with inspiration from Fig. 1 of Lahijanian et al. (2015).

to a deterministic ω-automaton with a one-pair Streett acceptance condition. If a formula allows a co-safe interpretation in addition to an infinite interpretation, both may be considered. In the infinite interpretation the trace is never allowed to leave the state space, whereas in the co-safe interpretation the trace can go anywhere once the objective has been satisfied.

A notable special case is reachability analysis with the co-safe objective $\varphi = \mathsf{F}\phi$, where ϕ is a propositional formula over Π. This specification is satisfied when a trace reaches the region defined by ϕ. Reachability analysis for the given problem setup was studied in detail by Svoreňová et al. (2017) and is of great importance here as well. As shown in Section 22.1, reachability is a fundamental building block of refinement procedures for more complex problems.

11.2 Controller Synthesis

The second problem considered is controller synthesis. The analysis identifies initial states from which traces can be controlled such that the given objective is fulfilled with probability 1. The aim of controller synthesis is to construct a controller that achieves this.

12 Solution Approach

The analysis and synthesis problems for the given LSS setup have been previously solved by Svoreňová et al. (2017) with an iteratively refined, game-based abstraction. Their solution procedure is depicted in Figure 3 and summarized in this section. The individual steps are then reviewed in detail in the following chapter.

First, a 2½-player game abstraction of the LSS is constructed based on a state space partition and its dynamics. In the game, one player controls the evolution of a trace

while the other player represents the uncertainty introduced by the abstraction model. For computational convenience, convex polytopic partitions are chosen. The initial state space partition is induced by the set of linear predicates.

Next, the objective is translated to a deterministic ω-automaton and the synchronous product of this automaton and the game graph is constructed. The resulting game's winning condition is modelled on the one-pair Streett acceptance condition of the automaton. The product game is then solved once for an adversarial player 2 and once for a cooperative player 2. With both solutions, regions of the state space can be identified as parts of X_{init} or $X \setminus X_{\text{init}}$. However, some regions can remain undecided because of a too coarse abstraction. Therefore, refinement is applied to the state space partition. Based on the game solutions and system dynamics, a new state space partition is constructed and the procedure returns to the abstraction phase and iterates.

The procedure is constructed such that after every iteration, a partial solution to the problem is obtained (Svoreňová et al. 2017). This means that once a region of the state space has been identified as a part of X_{init} or $X \setminus X_{\text{init}}$, this identification is provably correct and will not change in subsequent analyses or due to refinement activity. If the procedure terminates, the analysis problem from 11.1 is solved. However, refinement heuristics can only provide progress guarantees in some circumstances and there are systems for which this procedure will provably never terminate.

Finally, a controller is synthesized based on the game graph and the analysis results.

Abstraction and Analysis

The abstraction and analysis phases of the solution procedure outlined in Section 12 are described in this chapter, followed by a brief discussion of the controller synthesis problem. The solution was first proposed and implemented by Svoreňová et al. (2017). It is reviewed here.

First, a set of polytopic operators capturing important aspects of the dynamics is introduced in Section 13. A 2½-player game graph is then constructed from the linear stochastic system in Section 14. This game graph is synchronized with a deterministic ω-automaton obtained from the GR(1) specification in Section 15. Section 16 demonstrates how to obtain a solution to the analysis problem from the synchronized game. Based on this solution, an almost-sure winning control strategy for player 1 is synthesized in Section 17.

Throughout this chapter the simple, 1-dimensional LSS

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \boldsymbol{u}_t + \boldsymbol{w}_t \ , \tag{3}$$

where $\boldsymbol{x}_t \in X = [\,0,\, 4\,]$, $\boldsymbol{w}_t \in W = [\,-0.1,\, 0.1\,]$ and $\boldsymbol{u}_t \in U = [\,-1,\, 1\,]$ for all t, is used to illustrate the construction of the solution.

13 Dynamics Operators

A meaningful game-based abstraction of a linear stochastic system must reflect its dynamics. Here, operators on the state- and control space are defined. Each operator expresses an aspect of characteristic one-step behaviour of the system dynamics. All operator introductions are complemented by descriptions of their computation based on the polytopic operations from Section 6.

Because all operators are based on the dynamical properties of the LSS, they require knowledge of the LSS parameters, i.e. $\boldsymbol{A}$, $\boldsymbol{B}$, X, W, U. The association between an operator and its LSS is not captured explicitly by the presented notation and should always be clear from context that the operator is used in.

13.1 Posterior

The posterior (Post) is a forward-looking operator. Given an origin region $X' \subseteq X$ and control input region $U' \subseteq U$, it computes the set of states which are reachable from the origin region under the control inputs with non-zero probability. The returned region is therefore the one-step reachable set

$$\mathrm{Post}(X',\, U') := \{\boldsymbol{x} \in \mathbb{R}^n \mid \exists \boldsymbol{x}' \in X',\, \exists \boldsymbol{u}' \in U',\, \exists \boldsymbol{w} \in W : \boldsymbol{x} = \boldsymbol{A}\boldsymbol{x}' + \boldsymbol{B}\boldsymbol{u}' + \boldsymbol{w}\} \ .$$

The posterior allows the computation of an extended state space $X_{\text{ext}} = X \cup \text{Post}(X, U)$, which is the union of the original state space and its one-step reachable set.

Post, as defined above, takes set-valued arguments. The notation of this operator and all following operators is additionally overloaded for vector-valued state- and control-space arguments as well as sets of elements of a state space decomposition in the following way: Let $\boldsymbol{x}' \in X$, $\boldsymbol{u}' \in U$ and $X_j \subseteq X$ for all $j \in J$. Then

$$\begin{aligned} \text{Post}(\boldsymbol{x}', U') &\coloneqq \text{Post}(\{\boldsymbol{x}'\}, U') \ , \\ \text{Post}(X', \boldsymbol{u}') &\coloneqq \text{Post}(X', \{\boldsymbol{u}'\}) \text{ and} \\ \text{Post}(\{X_j\}_{j\in J}, U') &\coloneqq \text{Post}\left(\bigcup_{j\in J} X_j, U'\right) \ . \end{aligned}$$

For polytopic inputs, Post can be computed with the Minkowski sum as

$$\text{Post}(X', U') = \boldsymbol{A}X' \oplus \boldsymbol{B}U' \oplus W \ .$$

13.2 Predecessors

As the name suggests, predecessors are backward-looking operators, computing origin regions with specific properties for a given state space target and control input. The predecessor (Pre) and robust predecessor (PreR) are defined as

$$\begin{aligned} \text{Pre}(X', U', \tilde{X}) &\coloneqq \{\boldsymbol{x}' \in X' \mid \exists \boldsymbol{u}' \in U' : \text{Post}(\boldsymbol{x}', \boldsymbol{u}') \cap \tilde{X} \neq \emptyset\} \text{ and} \\ \text{PreR}(X', U', \tilde{X}) &\coloneqq \{\boldsymbol{x}' \in X' \mid \exists \boldsymbol{u}' \in U' : \text{Post}(\boldsymbol{x}', \boldsymbol{u}') \subseteq \tilde{X}\} \ , \end{aligned}$$

where the second argument $U' \subseteq U$ is a control-space region, the third argument $\tilde{X} \subseteq X_{\text{ext}}$ is a target region in the extended state space and the first parameter $X' \subseteq X$ is a region of the state space to which the returned predecessor is restricted and exists mainly for convenience.

From any state in a predecessor set, the specified target region is reachable with non-zero probability in one step of the system evolution for some control input in U'. The control inputs that enable these transitions can be different for every state in the predecessor origin region. The probability of reaching the target region in case of the robust predecessor is 1. The robust predecessor is therefore robust in the sense that the target will be reached exclusively from the computed origin region under the given control inputs, irrespective of the stochastic dynamics. Trivially, it holds that

$$\text{PreR}(X', U', \tilde{X}) \subseteq \text{Pre}(X', U', \tilde{X}) \ .$$

For target regions where $\tilde{X} \ominus W = \emptyset$, the PreR is always empty. This can easily be seen from the operator's computations based on polytopic operators:

$$\mathrm{Pre}(X', U', \tilde{X}) = X' \cap \Big(\tilde{X} \oplus -(\boldsymbol{B}U' \oplus W)\Big)\boldsymbol{A} \text{ and}$$
$$\mathrm{PreR}(X', U', \tilde{X}) = X' \cap \Big((\tilde{X} \ominus W) \oplus -\boldsymbol{B}U'\Big)\boldsymbol{A} \ .$$

Finally, the precise predecessor is defined. This operator only makes sense when used when applied to elements of a state space partition:

$$\mathrm{PreP}(X', U', \{X_j\}_{j\in J}) \coloneqq \Big\{\boldsymbol{x} \in X' \;\Big|\; \exists \boldsymbol{u}' \in U' : \mathrm{Post}(\boldsymbol{x}, \boldsymbol{u}') \subseteq \bigcup_{j\in J} X_j \text{ and } \forall j \in J : \mathrm{Post}(\boldsymbol{x}, \boldsymbol{u}') \cap \bigcup_{j\in J} X_j \neq \emptyset\Big\} \ .$$

Every state in the resulting origin set fulfills the robust predecessor property with respect to the entire target region $\bigcup_{j\in J} X_j$ for some $\boldsymbol{u}' \in U'$, while simultaneously fulfilling the predecessor property with respect to every individual part of the target region for the same $\boldsymbol{u}'$. In other words, for any state in a PreP set, a control input in U' exists such that both a state in $\bigcup_{j\in J} X_j$ is reached almost surely after one step and the probability of ending up in any one of the target region parts $\{X_j\}_{j\in J}$ is non-zero. The precise predecessor can be computed with

$$\mathrm{PreP}(X', U', \{X_j\}_{j\in J}) = X' \cap \Big(\Big(\bigcap_{j\in J}(X_j \oplus -W) \setminus \bigcup_{j\in I\setminus J}(X_j \oplus -W)\Big) \oplus -\boldsymbol{B}U'\Big)\boldsymbol{A} \ ,$$

where I is the index set enumerating the partition of the extended state space.

13.3 Attractors

Analogous to the predecessor and robust predecessor operators, the attractor (Attr) and robust attractor AttrR are defined as

$$\mathrm{Attr}(X', U', \tilde{X}) \coloneqq \{\boldsymbol{x} \in X' \mid \forall \boldsymbol{u}' \in U' : \mathrm{Post}(\boldsymbol{x}, \boldsymbol{u}') \cap \tilde{X} \neq \emptyset\} \text{ and}$$
$$\mathrm{AttrR}(X', U', \tilde{X}) \coloneqq \{\boldsymbol{x} \in X' \mid \forall \boldsymbol{u}' \in U' : \mathrm{Post}(\boldsymbol{x}, \boldsymbol{u}') \subseteq \tilde{X}\} \ ,$$

i.e. predecessor properties hold for all control inputs in U' for the attractors, not just some. Therefore,

$$\mathrm{Attr}(X', U', \tilde{X}) \subseteq \mathrm{Pre}(X', U', \tilde{X}) \ ,$$
$$\mathrm{AttrR}(X', U', \tilde{X}) \subseteq \mathrm{PreR}(X', U', \tilde{X}) \text{ and}$$
$$\mathrm{AttrR}(X', U', \tilde{X}) \subseteq \mathrm{Attr}(X', U', \tilde{X}) \ .$$

The attractor operators can be computed from the predecessor operators as

$$\mathrm{Attr}(X', U', \tilde{X}) = X' \setminus \mathrm{PreR}(X', U', X_{\mathrm{ext}} \setminus \tilde{X}) \ ,$$
$$\mathrm{AttrR}(X', U', \tilde{X}) = X' \setminus \mathrm{Pre}(X', U', X_{\mathrm{ext}} \setminus \tilde{X}) \ .$$

Although these operators are not used in the construction of the game graph, they play an important role in the refinement procedures presented later.

13.4 Actions

The last operators defined here are control space operators, i.e. the output of these operators is a subset of U, not X. The action (Act) and robust action (ActR), given by

$$\begin{aligned}\mathrm{Act}(X', \tilde{X}) &= \{\boldsymbol{u} \in U \mid \mathrm{Post}(X', \boldsymbol{u}) \cap \tilde{X} \neq \emptyset\} \text{ and} \\ \mathrm{ActR}(X', \tilde{X}) &= \{\boldsymbol{u} \in U \mid \mathrm{Post}(X', \boldsymbol{u}) \subseteq \tilde{X}\} \ ,\end{aligned}$$

return the sets of control inputs with which the probability of transitioning to the target region $\tilde{X}$ from origin region X' in one step is non-zero and 1, respectively. These are probabilistic variants of the operators $U^{X \to Y}$ (corresponding to $\mathrm{Act}(X, Y)$) and $U^{X \Rightarrow Y}$ (corresponding to $\mathrm{ActR}(X, Y)$) defined by Yordanov and Belta (2009) for a deterministic setting.

Both operators can be computed with the Minkowski sum and Pontryagin difference as

$$\begin{aligned}\mathrm{Act}(X_i, X_j) &= U \cap (X_j \oplus -(\boldsymbol{A}X_i \oplus W))\boldsymbol{B} \text{ and} \\ \mathrm{ActR}(X_i, X_j) &= U \cap (X_j \ominus (\boldsymbol{A}X_i \oplus W))\boldsymbol{B} \ .\end{aligned}$$

Because only full-dimensional polytopes are considered here (see Section 6), control-space regions of lower dimension than U are not computable by the action operators. With this constraint, it is possible that $\mathrm{ActR}(\boldsymbol{x}, \tilde{X}) = \emptyset$ for some $\boldsymbol{x} \in \mathrm{PreR}(X', U, \tilde{X})$, even though this should not be possible if lower-dimensional regions were allowed. This artifact of the computational constraints plays an important role in Section 16.2.

Completing the set of action operators is the concrete action (ActC). As for the precise predecessor, a state space partition is required for this operator to be meaningful:

$$\mathrm{ActC}(X', \{X_j\}_{j \in J}) \coloneqq \Big\{ \boldsymbol{u} \in U \;\Big|\; \mathrm{Post}(X', \boldsymbol{u}) \subseteq \bigcup_{j \in J} X_j \text{ and } \forall j \in J : \mathrm{Post}(X', \boldsymbol{u}) \cap X_j \neq \emptyset \Big\} \ .$$

Under the returned control inputs only $\tilde{X}$ can be reached in one step from states in X' (first condition) and for every element of the partitioned target region a state in X' exists such that the probability of reaching this element is non-zero. It must be noted that the conditions of PreP and ActC are quite different from one another: The precise predecessor requires some control input for every of its states for which both conditions (subset and part intersection) are fulfilled. The concrete action requires that for every of its control vectors the subset condition is fulfilled for every origin state while the part intersections

can be fulfilled independently by different states in X' as long as every part is reachable from some state. Hence,

$$\mathrm{ActC}(X',\, \{X_j\}_{j\in J}) \cap \mathrm{ActC}(X',\, \{X_j\}_{j\in J'}) = \emptyset$$

for all $J \neq J'$. The same does not hold for precise predecessors, which can overlap for different target sets. Concrete actions are computable directly from Act with

$$\mathrm{ActC}(X',\, \{X_j\}_{j\in J}) = \bigcap_{j\in J} \mathrm{Act}(X',\, X_j) \setminus \bigcup_{j\in I\setminus J} \mathrm{Act}(X',\, X_j)\ ,$$

where I is again the index set enumerating the elements of the partition of X_{ext}. Note that Svoreňová et al. (2017) use U_i^J to denote the set $\mathrm{ActC}(X_i,\, \{X_j\}_{j\in J})$, an abbreviation that is adopted here as well when space constraints demand it.

14 Game Graph

To analyse the LSS, its infinite-member state- and control spaces must be replaced by finite counterparts and every trace realizable in the LSS must also be realizable in the resulting discrete abstraction. Additionally, the abstraction may allow additional behaviour. If a temporal logic specification is found to be satisfied by the abstraction, it will also be satisfied by the original system whose behaviour is a subset of the abstraction's.

The abstraction model chosen by Svoreňová et al. (2017) is a probabilistic game graph $\mathcal{G}$ with 2-players, constructed from the dynamics operators from Section 13. In contrast to the derivation of Svoreňová et al. (2017), the game is build directly here, without the definition of an intermediate non-deterministic transition system which is extending to a 2½-player game in order to reintroduce the stochasticity of the LSS.

14.1 Player 1

The state space discretization requires the grouping of all state space vectors into a finite set of disjunct regions. As seen in Section 6, convex geometry has many advantageous computational properties, therefore a convex, polytopic partition of the extended state space X_{ext} is chosen. The inclusion of $X_{\mathrm{ext}} \setminus X$ in addition to the state space simplifies the handling of transitions out of the state space during construction. In the abstraction-analysis-refinement iterations of the solution procedure (Figure 3), this partition is subject to change. A sensible partition of X to initiate the procedure is based on the equivalence relation

$$\boldsymbol{x} \sim_{\Pi} \boldsymbol{x}' \iff \pi(\boldsymbol{x}) = \pi(\boldsymbol{x}')\ , \tag{4}$$

induced by the linear predicates Π, where $\pi : X \to 2^{\Pi}$ associates a state vector with the predicates it fulfills. The partition is polytopic, convex and allows a straightforward

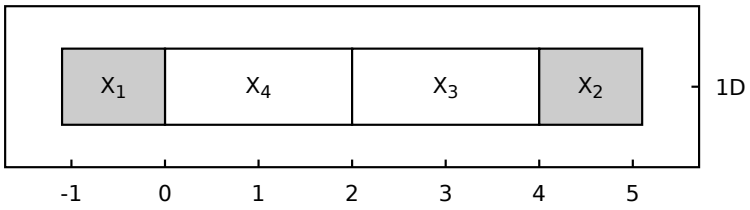


Figure 4: The initial state space partition for the example system (3), induced by the equivalence relation $\sim_{\{\pi_0\}}$ where π_0 is a linear predicate associated with halfspace $\{\boldsymbol{x} \in \mathbb{R}^n \mid -\boldsymbol{x} \leq 2\}$. X_1 and X_2 are the outer parts from the decomposition of $X_{\text{ext}} \setminus X$.

connection to the temporal logic specification later. The region $X_{\text{ext}} \setminus X$ can be partitioned arbitrarily into convex polytopes. It only exists as a convenient transition target and is not subjected to refinement.

The player 1 states

$$G_1 = \{X_i\}_{i \in I}$$

of the game abstraction $\mathcal{G}$ are based on this decomposition of X_{ext}. These states are directly identified with the corresponding polytopes of the state space partition, which are enumerated by the index set I. In a geometric context X_i refers to the polytope while in a game-theoretic context X_i refers to the associated player 1 state.

Figure 4 shows the initial state space partition for the example system (3) when equipped with a linear predicate π_0 associated with the halfspace $\{\boldsymbol{x} \in \mathbb{R}^n \mid -\boldsymbol{x} \leq 2\}$. The equivalence relation $\sim_{\{\pi_0\}}$ partitions the state space into two parts $X_3 = [\,2,\, 4\,]$ and $X_4 = [\,0,\, 2\,]$ such that $\pi(\boldsymbol{x}) = \{\pi_0\}$ for all $\boldsymbol{x} \in X_3$ and $\pi(\boldsymbol{x}) = \emptyset$ for all $\boldsymbol{x} \in X_4$. Two additional outer polytopes $X_1 = [\,-1.1,\, 0\,]$ and $X_2 = [\,4,\, 5.1\,]$ arise from the convex decomposition of $X_{\text{ext}} \setminus X$.

The state space partition induces an equivalence relation $\sim_{\boldsymbol{x}}$ over the control space for any given state vector $\boldsymbol{x}$ with

$$\boldsymbol{u} \sim_{\boldsymbol{x}} \boldsymbol{u}' \iff \forall j \in I : \Big((\text{Post}(\boldsymbol{x},\, \boldsymbol{u}) \cap X_j = \emptyset) \leftrightarrow (\text{Post}(\boldsymbol{x},\, \boldsymbol{u}') \cap X_j = \emptyset) \Big) ,$$

i.e. two control vectors are equivalent if and only if the same set of state space partition elements is reachable under them after one step of system evolution when starting in $\boldsymbol{x}$. In order to obtain actions for a state X_i based on this relation it must be extended to a polytope-based notion. However, not all states in a given X_i necessarily produce the same relation, so a common $\sim_{X_i}$ does not emerge naturally. Instead, it is defined as

$$\boldsymbol{u} \sim_{X_i} \boldsymbol{u}' \iff \forall j \in I : \Big((\text{Post}(X_i,\, \boldsymbol{u}) \cap X_j = \emptyset) \leftrightarrow (\text{Post}(X_i,\, \boldsymbol{u}') \cap X_j = \emptyset) \Big)$$

and used to generate the player 1 actions

$$Act_1 = \left\{ i \to J \;\middle|\; i \in I,\, J \subseteq I \right\} ,$$

named after the index of the origin state X_i and the indices of reachable target states $\{X_j\}_{j \in J}$, governed by $\sim_{X_i}$. No actions are defined for the outer states. Note that every set of reachable target states has a unique associated game action and control space region. The region is given by the concrete action dynamics operator ActC which reflects the action-generating relation $\sim_{X_i}$.

For polytope X_4 of the example LSS (3) with the decomposition from Figure 4, the following non-empty concrete actions are computed:

$$\begin{aligned} \mathrm{ActC}(X_4, \{X_3, X_4\}) &= [\,0.1,\, 1\,] \;, \\ \mathrm{ActC}(X_4, \{X_1, X_3, X_4\}) &= [\,-0.1,\, 0.1\,] \text{ and} \\ \mathrm{ActC}(X_4, \{X_1, X_4\}) &= [\,-1,\, -0.1\,] \;. \end{aligned}$$

Therefore, 3 actions are derived for the associated player 1 state of the game graph: $4 \to \{1, 4\}$, $4 \to \{1, 3, 4\}$ and $4 \to \{3, 4\}$.

14.2 Player 2

In the original LSS, after a control input has been selected, the next state is determined stochastically according to the evolution equation and the probability distribution over W. In the abstraction, a single probability distribution that describes the transitions of all $\boldsymbol{x} \in X_i$ does not exist generally, so the exact probabilities of reaching another target state when player 1 has selected an action are unknown until the trace and the control input are exactly localized. In particular, because of a possible mismatch between $\sim_{\boldsymbol{x}}$ and $\sim_{X_i}$, not every state in the action's target set may be reachable for every $\boldsymbol{x} \in X_i$. A simple probabilistic transition after a player 1 action to the next player 1 state is therefore not possible. The trace has to be localized in X_i and the exact control input determined, so that the probability distribution over the set of target states is well defined for a transition. Hence, a second player with the power to localize the trace and select a specific control vector from the ActC-regions associated with player 1 actions is required in the abstraction. The game can then transition to any of the reachable states by sampling the resulting probability distribution.

Because every target set of state space partition elements has exactly one associated player 1 action, player 2 states

$$G_2 = \left\{ (X_i,\, J) \;\middle|\; i \in I \text{ and } J \subseteq I \right\}$$

are simply defined as tuples of player 1 states and actions. The transition relation

$$\delta_{\mathcal{G}}\Big(X_i, i' \to J'\Big)\Big((X_i,\, J)\Big) = \begin{cases} 1 & \text{if } i = i' \text{ and } J = J' \\ & \quad \text{and } \operatorname{ActC}(X_i,\, \{X_j\}_{j\in J}) \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

matches each existing player 1 state-action pair with its corresponding player 2 state deterministically.

Player 2's actions are supposed to determine the probability distribution of the actual system transition. A finite set of actions is desired but there are potentially infinitely many because the probability distributions will be different for every $\boldsymbol{x} \in X_i$ and $\boldsymbol{u} \in \operatorname{ActC}(X_i,\, \{X_j\}_{j\in J})$ that player 2 can choose from. Fortunately, in the context of almost-sure analysis, all non-zero probabilities can be considered equivalent. Intuitively this can be understood by considering repeated attempts to make some event happen. As long the probability of the event happening is non-zero in every attempt, it will happen eventually with probability 1, i.e. almost surely. The exact probability is irrelevant as long as it does not drop to 0. This property of almost-sure analysis reduces the choice of the probability distribution to a choice of the support set of the probability distribution. Therefore, (finitely many) player 2 actions

$$Act_2 = \Big\{(i,\, J) \to K \;\Big|\; i \in I,\, K \subseteq J \subseteq I\Big\}\ ,$$

where K is the set of target state indices from the support of the probability distribution, can be defined without losing relevant behaviour of the system for the purposes of almost-sure analysis. The equivalence of all probability distributions with the same support sets also means that, for convenience, uniform distributions can be chosen everywhere for the player 2 transition relation

$$\delta_{\mathcal{G}}\Big((X_i,\, J), (i',\, J') \to K\Big)\Big(X_k\Big) = \begin{cases} \dfrac{1}{|K|} & \text{if } i = i' \text{ and } J = J' \text{ and } k \in K \\ & \quad \text{and } \operatorname{PreP}(X_i,\, U_i^J,\, \{X_k\}_{k\in K}) \neq \emptyset \\ 0 & \text{otherwise}\ , \end{cases}$$

where $U_i^J = \operatorname{ActC}(X_i,\, \{X_j\}_{j\in J})$ is the control input associated with the player 1 action $i \to J$ that lead to the player 2 state $(X_i,\, J)$ from player 1 state X_i. The dynamics operator used to express the transition condition is the precise predecessor, which corresponds to the origin regions in X_i for which probability distribution support sets $\{X_k\}_{k\in K}$ are identical for some $\boldsymbol{u} \in U_i^J$.

In the example system (3), consider state X_4 and its action $4 \to \{1, 3, 4\}$ computed in Section 14.1. This action leads (deterministically) to the player 2 state $(X_4,\, \{1, 3, 4\})$, which is associated with the non-empty precise predecessors

$$\begin{aligned} \operatorname{PreP}(X_4,\, U_4^{\{1,3,4\}},\, \{X_1, X_4\}) &= [\,0,\, 0.2\,]\ , \\ \operatorname{PreP}(X_4,\, U_4^{\{1,3,4\}},\, \{X_3, X_4\}) &= [\,1.8,\, 2\,] \text{ and} \\ \operatorname{PreP}(X_4,\, U_4^{\{1,3,4\}},\, \{X_4\}) \quad &= [\,0,\, 2\,]\ . \end{aligned}$$

Therefore, in this player 2 state, the actions $(4, \{1,3,4\}) \to \{1,4\}$, $(4, \{1,3,4\}) \to \{3,4\}$ and $(4, \{1,3,4\}) \to \{4\}$ are available.

14.3 Synopsis

The constructed 2½-player game graph is

$$\mathcal{G} = (G_1, G_2, Act, \delta_{\mathcal{G}}) \ ,$$

with $Act = Act_1 \cup Act_2$.

To summarize, the game is played starting from a player 1 state as follows: First, player 1 selects a set of state space partition elements reachable from the current state. One of these elements will be randomly selected as the next player 1 state. Because the real state of the trace in the LSS to which the play corresponds is unknown in the discretized state space, the distribution from which the successor is sampled is not uniquely defined yet and must be chosen by player 2. Conveniently, only the supports of the probability distributions matter for almost-sure analysis, so player 2 can choose from a (finite) set of uniform distributions with supports from the power set of the reachable states selected by player 1.

Translated to the corresponding trace in the LSS: Player 1 chooses a region of the control space given by one of the non-empty concrete actions of the state space partition element the trace is currently located in. Player 2 then chooses a specific control vector from this region and reveals the exact location of the trace in the state space. The trace steps forward in time according to the evolution equation and it is player 1's turn again.

An excerpt from the game graph constructed for the example system (3) is shown in Figure 5. It shows all actions for the player 1 state X_4 (computed in 14.1) as well as all player 2 successor states and their actions (partially computed in 14.2).

15 Product Game

The game graph constructed in the previous section has no winning condition. In order to carry out an analysis with respect to some specification, a winning condition has to be introduced. A temporal logic formula can be translated into a deterministic ω-automaton that accepts the same language as the the formula. It is possible to transfer the acceptance condition of this automaton to the game graph by constructing their synchronous product. The resulting product game enforces the synchronized evaluation automaton runs and plays on the game graph.

15.1 Objective Automaton

The GR(1) objective formula φ from Section 11 is formulated over a set of linear predicates Π. The initial decomposition of the state space (4) is induced by an equivalence relation

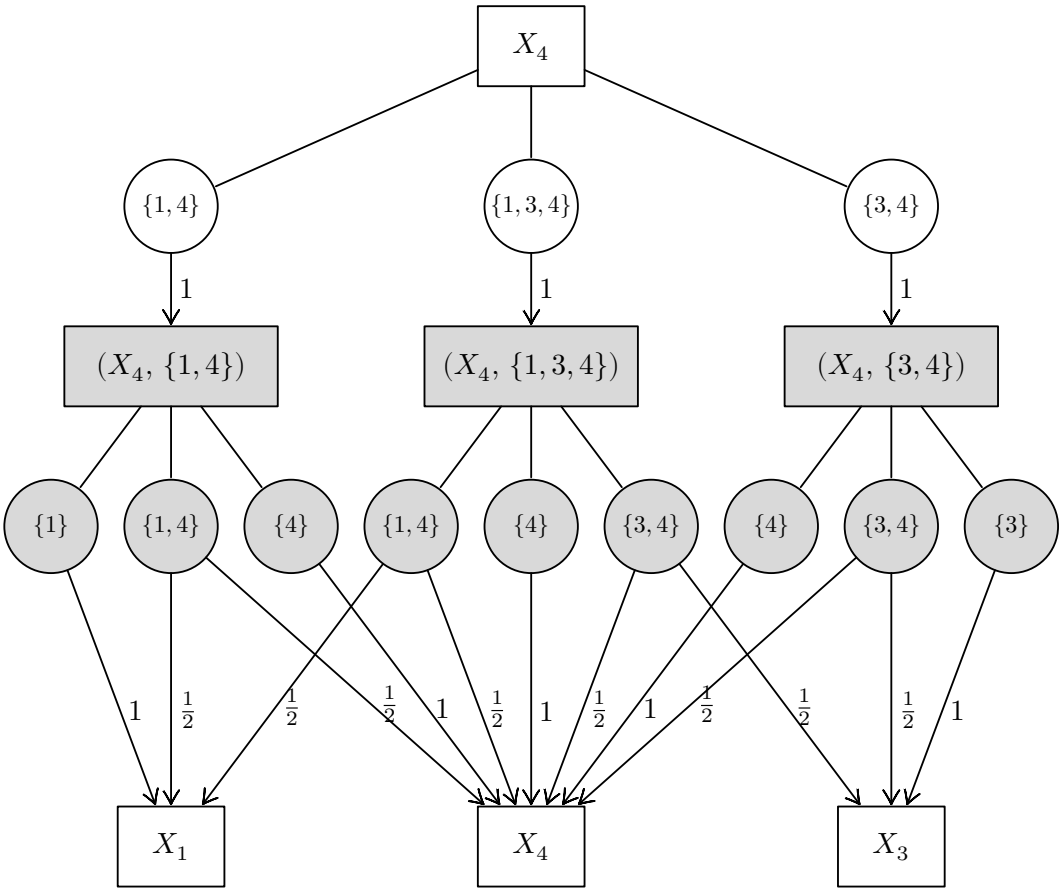


Figure 5: A subset of the game graph abstraction for the example system (3) with the partition from Figure 4. Rectangles denote states and circles denote actions. Player 1's states and actions are coloured white, player 2's grey. The probabilistic transitions after actions are labeled with their respective transition probabilities. The two depicted states X_4 are the same and only shown separately to improve readability.

governed by this set of linear predicates. Every element X_i of the initial state space partition therefore fulfills and rejects exactly the same predicates from Π. The function π can thus be extended to the partition without ambiguity, associating partition elements with the set of predicates they satisfy. Player 1 states of the game are identified uniquely with state space partition elements, so every play of $\mathcal{G}$ induces a word over the alphabet 2^{Π}. Using this alphabet, an ω-automaton

$$\mathcal{A} = (Q, 2^{\Pi}, \delta_{\mathcal{A}}, q_0, \mathcal{C}_{\mathcal{A}})$$

accepting the language of φ is constructed. A single Streett pair $\delta_{\mathcal{A}} = (E_{\mathcal{A}}, F_{\mathcal{A}})$ can be chosen as the acceptance condition due to the restriction of objectives to the GR(1) fragment of LTL.

15.2 Synchronized Product

To enforce the evaluation of the automaton in synchronization with plays on the game

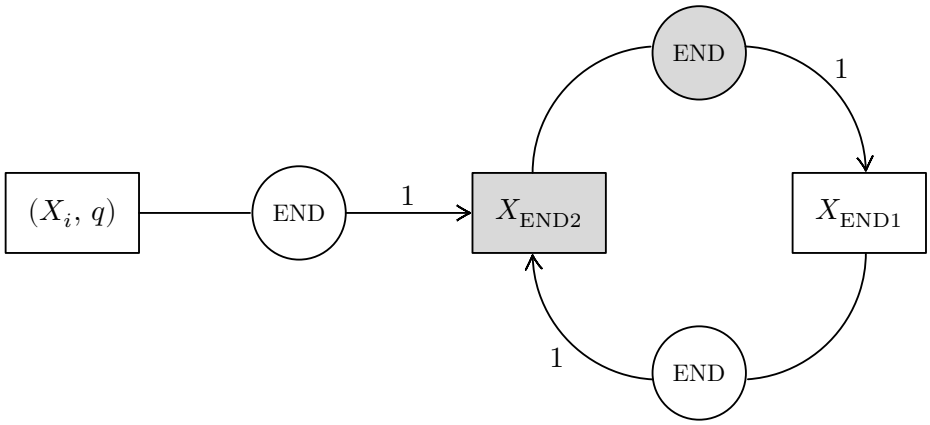


Figure 6: A player 1 state $(X_i,\, q)$ of the product game $\mathcal{P}$, connected to the dead-end loop.

graph, their synchronous product

$$\mathcal{P} = (P_1, P_2, Act, \delta, \mathcal{C})$$

is constructed. $\mathcal{P}$ is a 2½-player game with a one-pair Streett winning condition $\mathcal{C}$ modelled after $\mathcal{C}_{\mathcal{A}}$. The states of player 1 and 2 are given by $P_1 = G_1 \mathbin{\check{\times}} Q$ and $P_2 = G_2 \mathbin{\check{\times}} Q$ respectively, where $\mathbin{\check{\times}}$ is the normal Cartesian product but with flattened output tuples to reduce visual clutter. The set of actions Act is taken directly from the game graph $\mathcal{G}$. The synchronous evolution of automaton and game is implemented by the transition relation

$$\delta\Big((X_i,\, q), i \to J\Big)\Big((X_i,\, J,\, q')\Big) = \begin{cases} \delta_{\mathcal{G}}\Big(X_i, i \to J\Big)\Big((X_i,\, J)\Big) & \text{if } \delta_{\mathcal{A}}(q, \pi(X_i)) = q' \\ 0 & \text{otherwise} \end{cases}$$

for player 1 and

$$\delta\Big((X_i,\, J,\, q), (i,\, J) \to K\Big)\Big((X_k,\, q')\Big) = \begin{cases} \delta_{\mathcal{G}}\Big((X_i,\, J), (i,\, J) \to K\Big)\Big(X_k\Big) & \text{if } q = q' \\ 0 & \text{otherwise} \end{cases}$$

for player 2. Player states that cannot be reached with non-zero probability are removed from sets P_1 and P_2 of the product game. Note that a transition in the automaton is taken once for every subsequent player 1 and 2 turns, corresponding to one step of trace evolution in the underlying LSS, with the transition always occurring after a player 1 action. The one-pair Streett winning condition is given by $\mathcal{C} = (E,\, F)$ with $E = (G_1 \cup G_2) \mathbin{\check{\times}} E_{\mathcal{A}}$ and $F = (G_1 \cup G_2) \mathbin{\check{\times}} F_{\mathcal{A}}$.

15.3 Dead-End States

If $\mathcal{A}$ is an incomplete ω-automaton, $\mathcal{P}$ is incomplete as well and contains dead-end states in P_1, i.e. states without any outgoing actions. Because a word is rejected immediately

by an incomplete automaton when a non-existing transition would be required to advance a run, these dead-end states can be modified to complete the game in the following way: First, a player 1 state X_{END1}, a player 2 state X_{END2} and an action END are added to P_1, P_2 and Act, respectively. These states are combined with the action END to form a deterministic loop that traps any entering plays by defining

$$\delta(X_{\mathrm{END1}}, \mathrm{END})(s) = \begin{cases} 1 & \text{if } s = X_{\mathrm{END2}} \\ 0 & \text{otherwise} \end{cases} \quad \text{and}$$

$$\delta(X_{\mathrm{END2}}, \mathrm{END})(s) = \begin{cases} 1 & \text{if } s = X_{\mathrm{END1}} \\ 0 & \text{otherwise} \end{cases} .$$

Then any dead-end state (X_i, q) from P_1 is deterministically connected to this loop using the action END as depicted in Figure 6. To ensure that a play stuck in the loop will never lead to player 1 winning the game, states X_{END1} and X_{END2} are added to the set E of the one-pair Streett winning condition $\mathcal{C} = (E, F)$.

Dead-end states also occur due to the outer states from $X_{\mathrm{ext}} \setminus X$ which were explicitly constructed without actions in the game graph $\mathcal{G}$. These are connected to the same dead-end loop used for handling incomplete automata. Traces that leave the state space therefore immediately violate the specification.

15.4 Co-Safe Interpretation

An exception to the handling of outer states in the previous section must be made: For objectives interpreted in a co-safe setting, traces that have already fulfilled their objective are free to go anywhere, even outside of the state space. It is possible to accommodate this interpretation in the product game construction with a postprocessing step analogous to the one that resolves dead-end states.

A new player 1 state X_{SAT1} and player 2 state X_{SAT2} is introduced together with an action SAT. Final states in the product game, i.e. player 1 states whose actions trigger satisfaction of the co-safe objective, are first stripped of their existing actions and then deterministically redirected into a loop constructed exactly as in Figure 6 but using X_{SAT1} instead of X_{END1}, X_{SAT2} instead of X_{END2} and SAT instead of END. The only states occurring infinitely often in traces trapped in this loop are X_{SAT1} and X_{SAT2} which are not included in either of the acceptance sets of the one-pair Streett condition. Trapped traces therefore guarantee that player 1 wins the game almost surely, independent of what happens after the objective has been satisfied.

The following requirements are expected of one-pair Streett acceptance conditions (E, F) with co-safe interpretation: F contains all final states of the automaton or game and must not be empty. All other states are members of E. Using this convention, automata and games with these properties can be interpreted in both the infinite framework, where

traces must never leave the state space, and, after the above modifications, in the co-safe framework, where this requirement is lifted once a final state has been reached.

16 Analysis

The product game has to be analysed twice in order to decide which elements of the state space partition are subsets of X_{init} and which are subsets of $X \setminus X_{\text{init}}$. First, the game is solved under the assumption that player 2 is an adversary. With the results of this analysis one can identify game states in which player 1 can win almost-surely independent of the strategy of player 2 ("yes"-states). Then, the game is solved under the assumption that player 2 cooperates with player 1. Based on this analysis, game states in which player 1 cannot win almost-surely, independent of the player 2 strategy can be identified ("no"-states).

The game solutions are projected back onto the state space partition through the association of state space partition elements with player 1 game states in the initial state q_0 of the automaton. In general, not all states will be identified as yes- or no-states by the analysis procedure. If the abstraction is too coarse, a set of "maybe"-states remains whose corresponding state space partition elements can only be categorized after additional refinement.

16.1 Product Game Solution

A solution to the game for both the adversarial and cooperative setting can be found with the same algorithm. It is based on a fixed-point iteration solver for parity-3 games (Svoreňová et al. 2017).

Based on the successor operator

$$\text{Succ}(s, a) = \text{Supp}(\delta(s, a))$$

for game $\mathcal{P}$, two conditions

$$C_1(X) = \{(s, a) \in P \times Act \mid \text{Succ}(s, a) \subseteq X\} \text{ and}$$
$$C_2(X, Y) = \{(s, a) \in P \times Act \mid \text{Succ}(s, a) \subseteq X \text{ and } \text{Succ}(s, a) \cap Y \neq \emptyset\}$$

for state-action pairs of the product game graph are defined, where $P = P_1 \cup P_2$. $C_1(X)$ contains state-action pairs of the product game whose successor states are all contained in X. $C_2(X, Y)$ contains state-action pairs whose successor states are all contained in X while at least one successor state is contained in Y. Using these conditions, two sets of predecessor operators are defined. One for the adversarial setting

$$
\begin{aligned}
\mathrm{Pre}_1^{\mathrm{adv}}(X) &= \{s \in P_1 \mid \exists a \in Act : (s, a) \in C_1(X)\} \\
&\quad \cup \{s \in P_2 \mid \forall a \in \mathrm{Act} : (s, a) \in C_1(X)\} \ , \\
\mathrm{Pre}_2^{\mathrm{adv}}(X, Y) &= \{s \in P_1 \mid \exists a \in Act : (s, a) \in C_2(X, Y)\} \\
&\quad \cup \{s \in P_2 \mid \forall a \in \mathrm{Act} : (s, a) \in C_2(X, Y)\} \ , \\
\mathrm{Pre}_3^{\mathrm{adv}}(X, Y, Z) &= \{s \in P_1 \mid \exists a \in Act : (s, a) \in C_2(X, Y) \cup C_1(Z)\} \\
&\quad \cup \{s \in P_2 \mid \forall a \in Act : (s, a) \in C_2(X, Y) \cup C_1(Z)\}
\end{aligned}
$$

and one for the cooperative setting

$$
\begin{aligned}
\mathrm{Pre}_1^{\mathrm{coop}}(X) &= \{s \in P_1 \cup P_2 \mid \exists a \in Act : (s, a) \in C_1(X)\} \ , \\
\mathrm{Pre}_2^{\mathrm{coop}}(X, Y) &= \{s \in P_1 \cup P_2 \mid \exists a \in Act : (s, a) \in C_2(X, Y)\} \ , \\
\mathrm{Pre}_3^{\mathrm{coop}}(X, Y, Z) &= \{s \in P_1 \cup P_2 \mid \exists a \in Act : (s, a) \in C_2(X, Y) \cup C_1(Z)\} \ .
\end{aligned}
$$

Substituting these sets of operators into Algorithm 1 for $\mathrm{Pre}_1^{\mathrm{set}}$, $\mathrm{Pre}_2^{\mathrm{set}}$ and $\mathrm{Pre}_3^{\mathrm{set}}$ results in procedures to compute $\mathrm{Almost}^{\mathrm{adv}}(\mathcal{P})$ and $\mathrm{Almost}^{\mathrm{coop}}(\mathcal{P})$. Based on these solutions,

$$
\begin{aligned}
P_{\mathrm{yes}} &= P_1 \cap \mathrm{Almost}^{\mathrm{adv}}(\mathcal{P}) \ , \\
P_{\mathrm{no}} &= P_1 \setminus \mathrm{Almost}^{\mathrm{coop}}(\mathcal{P}) \text{ and} \\
P_? &= P_1 \setminus (P_{\mathrm{yes}} \cup P_{\mathrm{no}})
\end{aligned}
$$

are defined. For any play starting in a state from P_{yes}, player 1 has an almost-sure winning strategy even if player 2 plays as an adversary. No such strategy exists when a play is initiated from a state in P_{no}, even if player 2 is cooperative. For states in $P_?$ player 1 can currently only win almost-surely with the cooperation of player 2. The abstraction is too coarse to determine if an almost-sure winning strategy also exists in an adversarial setting.

Let

$$
\begin{aligned}
X_{\mathrm{yes}}^q &= \{X_i \mid (X_i, q) \in P_{\mathrm{yes}}\} \ , \\
X_{\mathrm{no}}^q &= \{X_i \mid (X_i, q) \in P_{\mathrm{no}}\} \text{ and} \\
X_?^q &= \{X_i \mid (X_i, q) \in P_?\}
\end{aligned}
$$

for all $q \in Q$. The notation of these sets is overloaded such that they denote the union of their member polytopes in geometric contexts, analogous to the use of sets of state space partition elements as arguments of the dynamics operators. The regions $X_{\mathrm{yes}}^{q_0}$ and $X_{\mathrm{no}}^{q_0}$ are then subsets of X_{init} and $X \setminus X_{\mathrm{init}}$ and a partial solution to the analysis problem posed in Section 11.1.

Input: $\mathrm{Pre}_1^{\mathrm{set}}$, $\mathrm{Pre}_2^{\mathrm{set}}$, $\mathrm{Pre}_3^{\mathrm{set}}$ of game $\mathcal{P} = (P_1, P_2, Act, \delta, (E,\, F))$
Output: $\mathrm{Almost}^{\mathrm{set}}(\mathcal{P})$

$P \leftarrow P_1 \cup P_2$
$D \leftarrow P \setminus (E \cup F)$
$X, \bar{X}, Y, \bar{Y}, Z, \bar{Z}$: Set
$\bar{X} \leftarrow P$
$\bar{Y} \leftarrow \emptyset$
$\bar{Z} \leftarrow P$
do
  $X \leftarrow \bar{X}$
  do
    $Y \leftarrow \bar{Y}$
    do
      $Z \leftarrow \bar{Z}$
      $\bar{Z} \leftarrow (F \cap \mathrm{Pre}_1^{\mathrm{set}}(X)) \cup (E \cap \mathrm{Pre}_2^{\mathrm{set}}(X,\, Y)) \cup (D \cap \mathrm{Pre}_3^{\mathrm{set}}(X,\, Y,\, Z))$
    while $Z \neq \bar{Z}$
    $\bar{Y} \leftarrow Z$
    $\bar{Z} \leftarrow P$
  while $Y \neq \bar{Y}$
  $\bar{X} \leftarrow Y$
  $Y \leftarrow \emptyset$
while $X \neq \bar{X}$
return X

Algorithm 1: A solver for the product game $\mathcal{P}$ with one-pair Streett acceptance condition $\mathcal{C} = (E,\, F)$ in either an adversarial ($\mathrm{set} = \mathrm{adv}$) or cooperative ($\mathrm{set} = \mathrm{coop}$) setting. The required $\mathrm{Pre}^{\mathrm{set}}$-operators are defined in Section 16.1.

16.2 Correctness and Termination

The (partial) correctness of the procedure presented in this chapter is proven by Svoreňová et al. (2017). Given an LSS and a sequence of state space partitions, ordered such that each partition of the sequence is a sub-partition of the previous, the volumes of the solution sets $X_{\mathrm{yes}}^{q_0} \subseteq X_{\mathrm{init}}$ and $X_{\mathrm{no}}^{q_0} \subseteq X \setminus X_{\mathrm{init}}$ are increasing monotonically. If $X_{\mathrm{yes}}^{q_0} \cup X_{\mathrm{no}}^{q_0} = X$, the procedure terminates. However, even if perfect refinement procedures were available, it is not guaranteed that the algorithm terminates for every LSS as the following examples demonstrate.

Consider the 1-dimensional LSS

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \boldsymbol{u}_t + \boldsymbol{w}_t \ ,$$

where $\boldsymbol{x}_t \in [0, 3]$, $\boldsymbol{u}_t \in [-1.5, 1.5]$ and $\boldsymbol{w}_t \in [-0.5, 0.5]$ for all times t. A co-safe reachability objective with target region $[1, 2]$ can be satisfied from every $\boldsymbol{x} \in [0, 3]$ with control input $\boldsymbol{u} = 1.5 - \boldsymbol{x}$. However, this control input is the only control input that leads to almost-sure satisfaction and it is different for every state of the state space. Therefore, a solution in form of a finite partition of the state space cannot exist.

But even if a finite state space partition does exist, the algorithm may still not terminate in practice due to computational constraints. To see this, consider the 1-dimensional LSS

$$\boldsymbol{x}_{t+1} = 1.5 \cdot \boldsymbol{x}_t + \boldsymbol{u}_t + \boldsymbol{w}_t \ ,$$

where $\boldsymbol{x}_t \in [-2, 2]$, $\boldsymbol{u}_t \in [-2, 2]$ and $\boldsymbol{w}_t \in [-1, 1]$ for all times t. A basic safety property is specified, i.e. traces are not allowed to leave the state space at any time. The partition

$$X_1 = [-2, -1], \ X_2 = [-1, 0], \ X_3 = [0, 1], \ X_4 = [1, 2]$$

is proposed for the state space. Using interval arithmetic, one can easily show that the application of control inputs 2 in X_1, 1 in X_2, -1 in X_3 and -2 in X_4 keeps traces starting from anywhere in the state space safe:

$$\begin{aligned}
\text{Post}(X_1, 2) &= 1.5 \cdot [-2, 1] + 2 + [-1, 1] = [-2, 1.5] \subseteq X \ , \\
\text{Post}(X_2, 1) &= 1.5 \cdot [-1, 0] + 1 + [-1, 1] = [-1.5, 2] \subseteq X \ , \\
\text{Post}(X_3, -1) &= 1.5 \cdot [0, 1] - 1 + [-1, 1] = [-2, 1.5] \subseteq X \ , \\
\text{Post}(X_4, -2) &= 1.5 \cdot [1, 2] - 2 + [-1, 1] = [-1.5, 2] \subseteq X \ .
\end{aligned}$$

In Section 13.4 it was noted that the action operators are unable to reproduce all possible behaviour of the dynamics because any polytope used in the abstraction has to be full-dimensional. It is therefore not possible to associate player 1 actions with single control vectors as proposed above. In practice, any control region must be able to contain a ball of some diameter $\epsilon > 0$ in order to be recognized as non-empty. Consider a trace in state $\boldsymbol{x}_t = -2 + \frac{2}{3}\delta$, for a small $\delta \geq 0$. From the condition for safe continuation of this trace one obtains

$$\begin{aligned}
& 1.5\Big(-2 + \frac{2}{3}\delta\Big) + \boldsymbol{u}_t + [-1, 1] \subseteq [-2, 2] \\
\Leftrightarrow \quad & [-4 + \delta + \boldsymbol{u}_t, -2 + \delta + \boldsymbol{u}_t] \subseteq [-2, 2] \\
\Rightarrow \quad & \boldsymbol{u}_t \geq 2 - \delta
\end{aligned}$$

due to the lower interval bound. For states $\boldsymbol{x}_t = -2 + \frac{2}{3}\delta$ with $\delta < \epsilon$, the remaining control region satisfying this condition is not able contain an ϵ-ball. It is therefore not possible to recognize these states as members of X_{init} or $X \setminus X_{\text{init}}$ due to the restrictions

of the abstraction. This issue will be referred to as "ϵ-limit behaviour" throughout this work.

16.3 Product Game Simplification

In Section 15.4, the co-safe interpretation of objectives is achieved by redirecting final states from the product game into a special loop which guarantees player 1 to win any play that enters it. If partial knowledge of P_{yes} and P_{no} is available from a previous abstraction-analysis-refinement iteration, it can be used to simplify the product game in a similar way, rendering parts of the game graph unreachable. The size of the product game can then be reduced by removing these states which reduces the cost of subsequent analysis efforts. Because it is guaranteed that any set of states previously decided to be part of P_{yes} has been identified correctly, redirecting the outgoing transitions of these states into the special winning loop does not affect the correctness of analysis results obtained from the modified game even if the state space partition was refined in the meantime. The same can be done to states recognized as members of P_{no} with redirection into the loop designed for dead-end states in Section 15.3.

While this simplification can reduce the computational demands of product game construction and analysis significantly and does not affect the correctness of solutions obtained from the modified game, it has an unfortunate side-effect. If a subset of the product game is not constructed due to the introduced redirection, no analysis results will subsequently be available for states in this subset. Results from previously recognized yes- and no-states can be transferred to such states from the previous analysis, but maybe-states in the removed subset will remain in an undecided state. The amount of information available to refinement methods or controller synthesis procedures is therefore reduced which may affect the quality of their results. Should the full information be required, the product game has to be constructed without any simplification and then analysed.

17 Controller Synthesis

For every trace starting in a state recognized as a member of X_{init}, a strategy exists that leads to player 1 winning almost-surely. A witness strategy can be constructed based on the result from the analysis of the product game in the following way (Svoreňová et al. 2017): At every step of the trace in the LSS, a control input is chosen such that the corresponding play in the product game only visits states from the set $\text{Almost}^{\text{adv}}(\mathcal{P})$. The control inputs are obtained from the ActC-regions associated with player 1 actions targeting only states in P_{yes}. These are selected in a round-robin fashion every time a trace visits a player 1 state. This is a pure strategy that requires a finite memory to keep track of the round-robin selection of actions.

In general, memoryless strategies exist for one-pair Streett objectives of 2½-player games, because they can be converted into parity-3 objectives for which such strategies are known to exist (Chatterjee and Henzinger 2012). Deeper analysis of the game is required to derive these strategies. A memoryless strategy for problems with robust solutions based on a geometric distance metric induced by the LSS dynamics is briefly discussed in Section 27.2. Otherwise, the controller synthesis problem is not a focus of this work.

Abstraction Refinement

The game-based abstraction requires a state space partition from which to derive player 1 states and actions. The initial partition described in Section 14.1 is governed by the set of linear predicates Π. This partition is convenient for the synchronous product construction because it creates a unique association between its parts and the linear predicates satisfied by them. However, this partition will generally be too coarse to fully solve the analysis problem for the given objective, with product game states remaining in the undecided set $P_?$ after the first analysis.

In this chapter, heuristic procedures are presented that refine the state space partition. Ideally, these procedures should enable a solution or at least a partial solution after some iterations of the abstraction-analysis-refinement cycle (Figure 3).

18 Refinement Guidance

If a given state space partition has been found to be too coarse, how should one determine which of its elements require refinement and how should this refinement look like? Answers to these question are given by the idea of counterexample-guided abstraction refinement (CEGAR, Clarke et al. 2000). In classical CEGAR, the analysis procedure generates a counterexample in the form of a trajectory in the abstraction that results in non-satisfaction of the specification. If the counterexample is found to be spurious, i.e. if it requires behaviour that is possible in the abstraction but not the original system, refinement is applied that eliminates this behaviour from the abstraction.

The analysis procedure described above does not return specific counterexample plays for states in which player 2 has a strategy to win with non-zero probability. It does however return the set $P_?$, containing all states in which player 1 cannot win almost-surely against an adversarial opponent, but winning is possible when player 2 is cooperative. The states in $P_?$ are therefore exactly those states with are associated with spurious behaviour in the abstraction, i.e. behaviour where player 1 might be able to win, but is unable to because of coarse abstraction. Without a specific counterexample play or trace, knowledge of the game graph and the LSS dynamics have to be combined with the analysis results to determine how to refine. Based on $P_?$, one can restrict refinement to the state space partition elements

$$\{X_i \mid \exists q \in Q : X_i \in X_?^q\} \ .$$

For other other elements of the partition an almost-sure winning strategy for player 1 either is either known to or not exist for all associated player 1 states of the product game and initial states of the original LSS. This is guaranteed by the partial correctness of the analysis procedure (Section 16.2).

Svoreňová et al. (2017) separated their refinement procedures into two categories named positive and negative. The goal of positive refinement is to enlarge the total state space volume associated with states in P_{yes} while negative refinement aims to enlarge the volume associated with P_{no}. Here, an additional category called neutral refinement is introduced. The purpose of procedures from this category is not to enlarge P_{yes} or P_{no} immediately, but to transfer control over the behaviour of a trace from player 2 to player 1 such that "bad" transitions can be avoided for longer. The value of creating such a "neutral" environment without inevitable bad transitions is derived from the stutter equivalence of traces evaluated with LTL formulas that do not contain the next operator (e.g. GR(1) formulas). In X-free LTL, only the order of events can be specified and not the temporal distance between them. For almost-sure verification, this means that there is no need to satisfy an objective in an efficient manner. Player 1 can take any number of turns to reach a state that has a desirable transition as long as the number of turns is finite and player 2 is unable to force an unwanted transition during these turns. Even if the desired transition is only possible with a small but non-zero probability, player 1 can still win almost-surely as long as the play remains in the neutral environment and the corresponding state can be reached over and over again after finite times. With each try, the total probability that the desired transition occurs approaches 1. Neutral refinement procedures aim to break patterns in the game graph that allow player 2 to win, i.e. the aforementioned bad transitions. Control is thus transferred to player 1 through the creation of a neutral environment. An almost-sure winning player 1 strategy for the objective may not immediately exist after neutral refinement but finding one through positive refinement should have become easier.

Even for positive procedures, feedback to refinement of the state space partition will not be immediate in general. Particularly for more complex objectives, multiple refinement steps might be necessary until the first states of the product game are recognized as members of P_{yes} or P_{no} by the analysis. This can be a challenge when fully autonomous selection of refinement procedures is desired. One cannot solely rely on direct feedback from the sets P_{yes}, P_{no} and $P_?$ to evaluate which refinement procedure to apply next since progress cannot be guaranteed in general. Unless a co-safe interpretation is adopted, where the final states are trivially satisfying, the set P_{yes} will likely be empty initially. The design of refinement procedures has to take this into account.

With multiple refinement procedures are at one's disposal, questions related to their coordination arise: Which procedures should be selected? In which order and how often should these procedures be applied? How often should the system be analysed between refinement steps to assess progress? General answers to these questions are hard to find. The best strategy for refinement coordination will depend on the objective, the properties of the LSS and even on the meaning of "best". One can optimize refinement for the shortest time until a certain percentage of the state space has been decided by the analysis, the smallest number of states space partition elements that have to be generated until

a solution is found, the suitability of the generated game graph for controller synthesis purposes and many other metrics. When performance is of importance, note has to be taken that the application of a refinement procedure invalidates (parts of) the game graph. If multiple refinement procedures are chained without an intermediate analysis (for which the game graph must be constructed inevitably), the invocation of game graph-based procedures may include significant additional computational cost because the game graph has to be recomputed. This gives an advantage to procedures based only on the analysis results and system dynamics as they can be chained without incurring this additional computational cost.

19 Positive Refinement with Robust Operators

In Section 13, 3 robust operators were introduced: PreR, AttrR and ActR. They are "robust" in the sense that they are independent of the stochasticity of the LSS dynamics, treating the uncertainty as an adversarial choice. Thus, robust operators are characterized by a guaranteed exclusivity with respect to their target regions. This guarantee can be used to design refinement procedures that ensure one-step reachability with respect to the target region after application, which is e.g. of interest when looking at safety objectives where the probability of reaching the safe region exclusively must be 1 in every step in order to ensure satisfaction.

In positive refinement, robust dynamics can provide progress guarantees for the overall abstraction-analysis-refinement procedure under some circumstances. This was recognized by Svoreňová et al. (2017), who designed a positive refinement procedure using the robust attractor to identify parts of the state space from which the region associated with states from P_{yes} can be reached in a single step of the dynamics almost-surely. The robustness guarantees that the state space volume associated with P_{yes} is enlarged after every refinement step, as long as a robust transition to the P_{yes}-region is possible. Before recreating this procedure in Section 20.1, a more detailed look at positive robust refinement and its properties is taken.

19.1 A Single-Step Robust Refinement Kernel

Based on Svoreňová et al. (2017)'s approach to positive refinement with the AttrR operator, a function

$$\text{RefineRobust}_{+} : C_n \times R_n \to 2^{C_n} \ ,$$

is defined, where C_n is the set of all convex polytopes in $\mathbb{R}^n$ and R_n is the set of all polytopic regions in $\mathbb{R}^n$. $\text{RefineRobust}_{+}(Y, Z)$ encapsulates the refinement of an origin polytope Y such that region Z can be reached exclusively and almost-surely after one step of the LSS dynamics from some non-empty set of elements of the returned Y-partition

Input: Origin polytope Y, target region Z
Output: Partition of Y according to RefineRobust_+

```
1  if ActR(Y, Z) ≠ ∅ then
2      return Y
3  end if
4  U' = GetControl(Y, Z)
5  if U' = ∅ then
6      return Y
7  end if
8  A = AttrR(Y, U', Z)
9  return Convexify(A) ∪ Convexify(Y \ A)
```

Algorithm 2: An implementation of RefineRobust_+ as defined in Section 19.1, based on the robust attractor operator. The implementation of GetControl is discussed in Section 19.2.

and for some non-empty control input region. This guarantees that in the game graph constructed after application of this refinement kernel, player 1 has an action for some state associated with the partition of Y that leads almost-surely and exclusively to Z after one turn. If no subset of Y with this property can be determined or Z can already be reached robustly from Y, Y is returned unchanged.

RefineRobust_+ is used as a building block for other, more complex, refinement methods throughout this chapter. It finds application in the holistic positive refinement in Section 20.1, the safety refinement in Section 20.3 and the reachability procedures in Section 22. An implementation based on the robust attractor is given in Algorithm 2. The case of a robust transition already being possible is covered by lines 1-3. In line 4, a control region is determined for which a robust transition to the target is possible for some states in Y. If no such control region is found, the origin polytope is returned unchanged (lines 5-7). Otherwise, the robust attractor with respect to the target region and control region is computed and the origin partitioned accordingly.

The helper function

$$\text{Convexify} : R_n \to 2^{C_n}$$

partitions any polytopic region into a set of disjunct convex polytopes. By definition, $\text{Convexify}(\emptyset) = \emptyset$. The implementation of

$$\text{GetControl} : R_n \times R_n \to R_m \ ,$$

where m is the dimension of the control space, is discussed in the next section.

Input: Origin polytope Y, target region Z

Output: Control region enabling a one-step robust transition from Y to Z or $\emptyset$

```
1   P ← PreR(Y, U, Z)
2   if P = ∅ then
3       return ∅
4   end if
5   V ← Vert(Hull(P))
6   for i = 1 ... 3n do
7       p ← random point in P
8       V ← V ∪ {p}
9   end for
10  A ← {ActR(v, Z) | v ∈ V}
11  C ← ∅
12  for all A′ ∈ 2^A do
13      U′ ← ⋂_{a∈A′} a
14      if U′ ≠ ∅ and |A′| > |C| then
15          C ← A′
16      end if
17  end for
18  return ⋂_{c∈C} c
```

Algorithm 3: An implementation of GetControl, discussed in Section 19.2. $n \in \mathbb{N}$ is the dimensionality of the LSS's state space X.

19.2 Control Region Selection

In Algorithm 2, a control region U' must be determined in the call GetControl(Y, Z) such that AttrR(Y, U', Z) on line 8 returns a non-empty region (if such an AttrR-region exists). In their positive refinement scheme, Svoreňová et al. (2017) applied the robust attractor refinement only to state space partition elements as the origin and target regions. In order to obtain a non-empty AttrR, they selected a control region based on the ActC associated with a suitable player 1 action. This procedure is not ideal for use with RefineRobust$_+$ because the origin and target regions accepted as input by RefineRobust$_+$ can be arbitrary polytopic regions of the state space and are not bound to the current state space partition and game graph. Player 1 actions are therefore generally not available.

Instead, Algorithm 3 is proposed which is independent of the game abstraction, using only geometric operations and the LSS dynamics. It is based on a Monte Carlo approach, sampling a few state vectors from inside the origin region and computing the control region usable for a single-step robust transition to the target for each sample state. These control

regions are then clustered and the cluster that most points "agree" with is returned. Since the sample states should be spread relatively uniformly over the polytope, the control region that most states can agree on should yield the largest AttrR-region in RefineRobust$_+$.

First, the robust predecessor of Z in Y with respect to the entire control space is computed in line 1. If a robust transition to Z from somewhere in Y is possible, the origin states must lie in this PreR. Then, a number of random state vectors from within the PreR-region are sampled in lines 6-9. The vertices of the hull of this region are added to the set of sampled vectors as well (line 5). Experience shows that this improves the efficacy of RefineRobust$_+$, as these points are commonly associated with highly agreeable control regions. The ActR operator with respect to target region Z is applied to each sample point in line 10, resulting a set of control regions with which a robust transition to the target is possible from at least one state in Y. In lines 11 to 17, the ActR-regions are clustered and the cluster with most origin state vector members determined. Its associated control region is returned in line 18. The clustering shown here is exhaustive, but quite expensive and cheaper clustering methods can be substituted if desired. The number of sample state vectors is chosen to be only 3 times the dimensionality of the state space to limit the computational demand.

19.3 Limitations of Robust Refinement

While the guarantees provided by robust refinement are very agreeable, the circumstances in which it can be applied have limitations. The adversarial treatment of probability in the robust framework leads to a conservative view of the system, neglecting the positive potential of the probabilistic dynamics for player 1. Generally, this means that robust procedures tend to overrefine the state space, due to the stricter-than-necessary requirements of robust dynamics. It also restricts which problems can be treated, as the solution to some problems may unavoidably depend on the probabilistic dynamics.

In particular, if a target region Z satisfies $Z \ominus W = \emptyset$, its robust predecessor is always empty, meaning the region cannot be targeted exclusively. But even regions that can be targeted robustly can be problematic in practice if they are small, as the required control for robust reachability has to be very precise and origin regions that can agree on a common control region are limited in size. In such cases a bigger target area must be built up over multiple refinements until efficient refinement is possible. Alternatively, small targets can be enlarged under careful consideration of the probabilistic dynamics. E.g. one can try to find a region Z' with $Z \subseteq Z' \subseteq X$ such that the probability of reaching Z is non-zero for every trace that is steered into Z', i.e. $(Z' \setminus Z) \ominus W = \emptyset$. The extended region $Z' \setminus Z$ may require additional refinement, e.g. such that it fulfills a safety property with respect to Z' or X.

Input: State-space partition element X_i
Output: Partition of X_i

```
1  Y ← {X_i}
2  for all q ∈ Q do
3     if X_i ∈ X_?^q then
4        q' ← δ_A(q, π(X_i))
5        Y' ← ∅
6        for all Y_n ∈ Y do
7           Y' ← Y' ∪ RefineStep(Y_n, q')
8        end for
9        Y ← Y'
10    end if
11 end for
12 return Y
```

Algorithm 4: A generic template for a single-step holistic refinement procedure based only on the system dynamics and analysis results. Implementations of RefineStep based on ideas from Sections 20.1 (positive robust refinement), 20.2 (negative attractor refinement) and 20.3 (safety refinement) are summarized in Table 2.

20 Holistic Refinement

The first set of refinement procedures introduced is referred to as "holistic" methods because they operate on the entire product game. They are single-step methods, i.e. they only take a single step of the LSS dynamics into account when evaluating which states to refine. This restriction limits the complexity of patterns that can be identified but simplifies the procedures.

20.1 Positive Robust Refinement

The first holistic refinement procedure is an adaptation of case 1 of the positive refinement proposed by Svoreňová et al. (2017). It identifies states from which P_{yes} can be reached robustly in a single step and uses the RefineRobust$_+$ kernel from Section 19.1 to partition the associated state space polytopes. This can be done by looking for player 1 states in $P_?$ from which a state in P_{yes} can be reached with probability 1 if both players cooperate, i.e. states $(X_i, q) \in P_?$ with an action $i \to J$ leading to player 2 state (X_i, J, q'), which has a player 2 action $(i, J) \to k$ such that $(X_k, q') \in P_{\text{yes}}$ for all $k \in K$. While this is a viable implementation of the refinement condition, it binds the procedure to the availability of player actions which could be associated with expensive game graph recomputation when the method is not used directly after an analysis.

Algorithm 4 demonstrates how the game graph construction can be avoided. Instead of analysing player actions, the algorithm systematically explores all player 1 states associated with a state space partition element X_i, filters out states that are not in $P_?$ (line 3) and determines the unique successor automaton state for all player 1 actions from the satisfied linear predicates (line 4). X_i is then partitioned with a function RefineStep which, if chosen such that

$$\text{RefineStep}(Y_n,\, q') = \text{RefineRobust}_+(Y_n,\, X^{q'}_{\text{yes}}) \ ,$$

encapsulates both the desired condition and executes the refinement (lines 5 to 9). Because RefineRobust_+ can be implemented solely based on the LSS dynamics (Algorithm 2), no dependence on the availability of the game graph exists.

The properties of the robust refinement kernel guarantee that the procedure enlarges the state space volume associated with P_{yes} whenever a state is successfully partitioned. However, if P_{yes} is empty, no refinement can take place at all. This is not a problem for co-safe specifications where the final states are immediately recognized as satisfying states in the first analysis. For other objectives though, the initial partition will generally not yield any yes-states after the first analysis. This issue is addressed in Section 22, where robust positive refinement will appear again in the context of a reachability decomposition approach.

20.2 Negative Attractor

A second refinement condition identified by Svoreňová et al. (2017) is the pattern

$$\begin{aligned} & \forall (i \to J) \in Act \ \exists ((i,\, J) \to K) \in Act \ \exists k \in K : X_k \in X^{q'}_{\text{no}} \\ \Leftrightarrow \quad & \forall (i \to J) \in Act \ \exists j \in J : X_j \in X^{q'}_{\text{no}} \ , \end{aligned} \tag{5}$$

where $q' = \delta_{\mathcal{A}}(q, \pi(X_i))$. It matches player 1 states in which an adversarial player 2 can win the game with non-zero probability independent of the action that player 1 chooses. As refinement, they propose a negative procedure that partitions state space partition elements according to the attractor sets $\text{Attr}(X_i,\, U,\, X^{q'}_{\text{no}})$. Plays originating in this attractor region and transitioning to q' are won by player 2 with non-zero probability inevitably, even if player 2 plays cooperatively. The removed attractor region is therefore guaranteed to be recognized as non-satisfying in the next analysis.

Again, it is possible to execute this refinement without having to consult the player actions of the game graph. Reusing the structure of Algorithm 4 and choosing

$$\begin{aligned} \text{RefineStep}(Y_n,\, q') = \ & \text{Convexify}(\text{Attr}(Y,\, U,\, X^{q'}_{\text{no}})) \\ & \cup \text{Convexify}(Y \setminus \text{Attr}(Y,\, U,\, X^{q'}_{\text{no}})) \end{aligned}$$

a) Positive robust refinement

Input: Polytope Y, successor automaton state q'
Output: Partition of Y

1 **return** RefineRobust$_+(Y, X^{q'}_{\text{yes}})$

b) Negative attractor refinement

Input: Polytope Y, successor automaton state q'
Output: Partition of Y

1 $A \leftarrow \text{Attr}(Y, U, X^{q'}_{\text{no}})$
2 **return** $\text{Convexify}(A) \cup \text{Convexify}(Y \setminus A)$

c) Safety refinement

Input: Polytope Y, successor automaton state q'
Output: Partition of Y

1 **return** RefineRobust$_+(Y, X \setminus X^{q'}_{\text{no}})$

Table 2: Implementations of the function RefineStep invoked by Algorithm 4 based on ideas from Sections 20.1 (a, positive robust refinement), 20.2 (b, negative attractor refinement) and 20.3 (c, safety refinement). All implementations only depend on the analysis results and the LSS dynamics but not the player actions of the product game graph.

results in the negative refinement procedure of Svoreňová et al. (2017).

20.3 Safety

The negative attractor refinement presented in the previous section does not generally resolve all occurrences of the problematic pattern (5). It only identifies and refines regions where player 2 wins inevitably with non-zero probability. A state (X_i, q) of the product game might satisfy condition (5) but also have a player 1 action after whose selection states from $X^{q'}_{\text{no}}$ with $q' = \delta_{\mathcal{A}}(q, \pi(X_i))$ can be avoided almost-surely if player 2 cooperates, resulting in a safe transition from the perspective of player 1. In such cases, refinement should be applied that allows player 1 to enforce this safe transition even if player 2 plays as an adversary. A procedure of this kind belongs in the category of neutral refinement as it transfers control from player 2 to player 1 but neither aims to explicitly enlarge the state space volume associated with P_{yes} nor the volume associated with P_{no}.

Algorithm 4 with the choice

$$\text{RefineStep}(Y_n, q') = \text{RefineRobust}_+(Y_n, X \setminus X^{q'}_{\text{no}})$$

results in a procedure that improves safety for player 1 by splitting state space partition elements associated with unsafe states with RefineRobust_+, targeted at a safe region. The procedure can only guarantee the existence of a safe player 1 action for some parts of the generated partition so multiple iterations might be necessary to ensure safety for all states. It is also possible that some regions of the state space cannot be made safe due to the ϵ-limit behaviour seen in Section 16.2.

Table 2 summarizes the choices for the function RefineStep of Algorithm 4 from this and the previous two sections.

20.4 Loop Removal

The motivation for neutral refinement was the stutter equivalence of plays when subjected to almost-sure analysis with respect to X-free LTL. In a neutral environment, player 1 can take any finite amount of turns to arrive at a good transition. But if an adversarial player 2 can force a play into an infinite, non-satisfying loop, then player 1 cannot win with probability 1 even if states from P_{no} are avoided almost-surely in every turn. Such loops in the game graph may involve multiple elements of the state space partition or multiple states of the automaton, in which case deep analysis of the product game is required for their identification. Much easier to find are self-loops, i.e. individual states of the game graph in which player 2 can trap a trace indefinitely.

The same observation was made by Yordanov et al. (2012) in the context of hybrid systems with non-probabilistic, piece-wise linear dynamics and LTL objectives. They separated self-loops of the abstraction into transient loops, which can be escaped in finite time in the original system by repeated application of the same control input, and non-transient loops, which cannot be broken by repeated application of the same control input. Their solution procedure did not incorporate abstraction refinement. Instead, they replaced non-transient self-loops in the game abstraction with corresponding non-looping player 1 actions, using the stutter equivalence property of X-free LTL. Here, instead of modifying the game graph pre-analysis, the problem of self-loops is approached with refinement. The goal is to refine the state space partition such that for all states of the game graph, player 1 has an action available after which player 2 is not able to enforce looping with probability 1. The refinement procedure in Algorithm 5 is proposed to remove potentially problematic self-loops in the product game graph.

Given an element X_i of the state space partition, Algorithm 5 searches for an associated player 1 state $(X_i, q) \in X^q_?$ which has a safe player 1 action $i \to J$ that leads to a player 2 state $(X_i, i \to J, q)$ in which a player 2 action $(i, J) \to \{i\}$ exists whose only target is the origin player 1 state (X_i, q). The procedure only looks at player 1 states whose outgoing actions don't change the automaton state, otherwise stutter equivalence does

Input: State-space partition element X_i
Output: Partition of X_i

```
1  for all q ∈ Q do
2    if X_i ∈ X_?^q and δ_A(q, π(X_i)) = q then
3      if ∃J ≠ {i} : ((i, J) → {i} ∈ Act and {X_j}_{j∈J} ∩ X_no^q = ∅) then
4        J: witness from previous condition
5        U_i^J ← ActC(X_i, {X_j}_{j∈J})
6        P ← PreP(X_i, U_i^J, X_i)
7        return Convexify(P) ∪ Convexify(X_i \ P)
8      end if
9    end if
10 end for
11 return {X_i}
```

Algorithm 5: Self-loop removal refinement for an element X_i of the state space partition.

not apply. Unsafe actions that lead to any state in P_{no} with non-zero probability are ignored as they are not considered to be neutral. The safety refinement of Section 20.3 should be applied to these states first. Self-loops initiated deterministically by player 1 with an action $i \to \{i\}$ are also ignored since player 1 already has full control over the looping behaviour for these actions.

If any player 2 action fulfills the self-loop condition, its corresponding origin polytope X_i is partitioned according to the precise predecessor associated with the player 2 action. The precise predecessor set contains all states in X_i from which a trace can potentially enter the self-loop. After extraction of these states, player 1 gains additional control over the looping behaviour while the trace localization power of player 2 is weakened. The refinement does not guarantee that the resulting states have no new self-loops, so multiple applications of the procedure may be required to ensure this.

Note that if a state space part X_i fulfills

$$(\text{Post}(X_i, U) \cap X_i) \ominus W = \emptyset \ ,$$

no self-loop can exist since any precise predecessor of $\text{Post}(X_i, U) \cap X_i$ will be empty. This property can be used to filter candidate states for the procedure relatively cheaply when the game graph has not been recomputed directly after another refinement step. It also ensures that all self-loop behaviour is eliminated eventually, namely when the property is satisfied by all states of the state space partition.

21 A Reduction Approach

Holistic refinement procedures seek patterns in the entire product game graph. The procedures presented in Section 20 only consider a single step of the system dynamics due to the high computational cost of obtaining deeper insights into the game graph. But without the consideration of multi-step behaviour, refinement progress is limited to a local scale. Due to the high computational cost of construction and analysis of the product game, patterns suitable for non-local refinement are expensive to identify. Computational demands can be reduced through the extraction of meaningful subsets of the game graph. This is particularly interesting in the context of positive refinement. Every satisfying player 1 strategy based on a subset of the product game can also be realized in the full product game as long as the power of player 2 has not been reduced by the subset selection.

One possible subset extraction method is to remove all but one action from every player 1 state of the game graph. This subset selection only limits the abilities of player 1, so almost-sure satisfying strategies obtained from it can be transferred to the full game. The reduction allows to remove player 1 from the game entirely so that it turns into an MDP. The lower computational complexity of MDP analysis and uncontrolled dynamics improves the feasibility of multi-step analysis. A refinement procedure based on this reduction can be constructed in the following way: First a control region is assigned to every element of the state space partition. This turns the controlled dynamics of the LSS into piece-wise dynamics without control. An MDP abstraction is constructed on top of the piece-wise system. The state space is refined based on insights from the MDP. Finally, the state space partition of the MDP system is transferred back to the original system.

Experience gathered with a prototype implementation of such a refinement scheme revealed drawbacks. The choice of a control region for every state space partition element is non-trivial. A certain amount of foresight is required in order to obtain an effective refinement procedure since badly chosen control regions significantly limit its potential. The non-continuous piece-wise dynamics causes the refinement to produce jagged regions, which result in a very fine partitioning of the state space (see Figure 7(a) where this issue is illustrated for positive robust refinement). Such jaggedness quickly snowballs into an explosion of the number of (small) partition elements, which can offset the complexity reduction gained from the MDP abstraction. Due to these issues, the approach was abandoned in favour of another subset extraction technique.

Instead of reducing the number of actions per state, one can directly reduce the number of states. This makes sense when the remaining subset of states and actions forms a meaningful sub-problem of the entire product game. It is not uncommon to encounter objectives made up of multiple separable sub-objectives, e.g. a safety property in conjunction with a recurrence objective, which is itself a sequence of reachability problems. If these sub-objectives can be separated in the product game and solved individually, one

can expect that a solution for the composite objective emerges. For refinement, this is advantageous because the sub-problems will often provide more immediate analysis feedback after refinement. The analysis of simple sub-problems might be possible without the need to construct a game abstraction altogether. E.g., safety properties can be verified with only the ActR-operator. Parallel refinement of the sub-problems is generally possible but restricted by the fact that state space partition is shared between the individual sub-problems in the product game.

22 Transition Refinement

A decomposition approach is presented that extracts a series of reachability problems from the product game. These sub-problems are solved with positive robust refinement, which is computationally inexpensive even when multi-step dynamics is taken into account. Solutions of the sub-problems are transferred back to the product game through refinement. When all reachability problems have been solved, a solution for the original problem emerges.

22.1 Transition-Based Reachability Decomposition

Because of the stuttering equivalence of paths in X-free LTL, the product game $\mathcal{P}$ can be decomposed into a set of co-safe reachability/avoidance problems. The idea behind this decomposition is motivated by an example:

Assume the recurrence objective $\mathsf{GF}\varphi$ is specified for some LSS, translated to a one-pair Streett automaton as shown in Table 1. One possible satisfying path through the automaton is $(q_0 q_1)^\omega$. Because recurrence is a X-free specification, any stuttering equivalent path $q_0^+ q_1^+ q_0^+ q_1^+ ...$ is also satisfying, so remaining in any of the two automaton states for a finite number of steps before transitioning has no consequences with respect to the acceptance condition. Based on this observation, two reachability problems are extracted from the product game: First, $\mathcal{R}_{q_0 \to q_1}$ where player 1 has to reach a q_1-associated player state of the product game almost-surely, starting from a state (X_j, q_0) and only visiting q_0-associated states until a target state is reached. Second, $\mathcal{R}_{q_1 \to q_0}$ which is defined analogously but with the roles of q_0 and q_1 exchanged.

Let $X_{\text{init}}^{\mathcal{R}_{q_0 \to q_1}}$ and $X_{\text{init}}^{\mathcal{R}_{q_1 \to q_0}}$ be the sets of satisfying initial states of the LSS for problems $\mathcal{R}_{q_1 \to q_0}$ and $\mathcal{R}_{q_1 \to q_0}$, respectively. Then a player 1 strategy $S^1_{q_0 \to q_1}$ exists that leads any trace starting in $X_{\text{init}}^{\mathcal{R}_{q_0 \to q_1}}$ almost-surely and in finite time to the state space region where an associated play of the product game switches from a q_0- to a q_1-associated state, i.e. the region defined by φ. An analogous strategy $S^1_{q_1 \to q_0}$ exists for traces starting in $X_{\text{init}}^{\mathcal{R}_{q_1 \to q_0}}$ with a target region defined by $\neg\varphi$. Based on these strategies, an almost-sure winning player 1 strategy for full product game can be assembled. Let $\boldsymbol{x}$ be a trace starting in some

state (X_i, q_0). Player 1 plays with $S^1_{q_0 \to q_1}$ until a state (X_j, q_1) is reached. Player 1 then switches to the strategy $S^1_{q_1 \to q_0}$ until a state (X_k, q_0) is reached. Then player 1 switches back to strategy $S^1_{q_0 \to q_1}$ and keeps switching whenever the automaton state changes in the play. Because automaton runs induced by a composite strategy alternating in this fashion are stuttering equivalent to $(q_0 q_1)^\omega$, the recurrence objective is satisfied almost-surely. Note that each sub-problem-strategy is guaranteed to achieve its reachability goal in finite time as the sub-problems are interpreted in the co-safe setting.

The example illustrates how a solution to a verification problem involving a X-free LTL specification can emerge from solutions to a series of reachability sub-problems extracted from the product game based on transitions of the objective automaton. Generally, it is not sufficient to just consider a single path and its stuttering equivalents. The above strategy construction fails when a trace enters the region $X \setminus X_{\text{init}}^{\mathcal{R}_{q_0 \to q_1}}$ while the corresponding play transitions from a q_0-associated to a q_1-associated state. However, since the decomposition targets individual automaton transitions and not entire paths, strategies for all possible paths through the automaton can be constructed once a solution for every reachability sub-problem is available.

For product game $\mathcal{P}$ and objective automaton $\mathcal{A}$, a reachability/avoidance sub-problem

$$\mathcal{R}_{q \to q'} = (X^{q \to q'}_{\text{reach}}, X^{q \to q'}_{\text{refine}}, X^{q \to q'}_{\text{avoid}})$$

for an automaton transition from q to q' can be constructed as follows: First, identify

$$X^{q \to q'}_{\text{reach}} = \{X_i \mid (X_i, q) \in P_1 \text{ and } X_i \notin X^q_{\text{no}} \text{ and } (\delta_{\mathcal{A}}(q, \pi(X_i)) = q' \text{ or } X^q_{\text{yes}})\} \ ,$$

the state space partition elements whose union has to be reached. These are state space parts where a transition to the target automaton state happens with any of the next player 1 actions of the product game. Also included are all partition elements that were already recognized as satisfying for the origin q in a previous analysis of $\mathcal{P}$. Second,

$$X^{q \to q'}_{\text{refine}} = \{X_i \mid (X_i, q) \in P_1 \text{ and } X_i \notin X^q_{\text{no}} \text{ and } \delta_{\mathcal{A}}(q, \pi(X_i)) = q\} \ ,$$

the state space partition elements that do not trigger an automaton transition with their player 1 actions. These elements will be the subject of the refinement. Due to stuttering equivalence, any finite number of transitions inside this region can be made without affecting satisfaction of the objective. And finally,

$$X^{q \to q'}_{\text{avoid}} = \{X_i\}_{i \in I} \setminus \left(X^{q \to q'}_{\text{reach}} \cup X^{q \to q'}_{\text{refine}}\right) \ ,$$

the elements where a transition to any other automaton state happens with the next player 1 action, as well as all elements from the no-set of the last product game analysis. These states are to be avoided. The three sets are disjunct except for the special case $q = q'$, where $X^{q \to q'}_{\text{reach}} \cap X^{q \to q'}_{\text{refine}} \neq \emptyset$.

Input: $X^{q\to q'}_{\text{reach}}$, $X^{q\to q'}_{\text{refine}}$ and i_{max}
Output: Sub-partition of $X^{q\to q'}_{\text{refine}}$

```
1  T ← X^{q→q'}_reach
2  Y ← X^{q→q'}_refine
3  i ← 0
4  while Y \ T ≠ ∅ and i < i_max do
5     Y' ← ∅
6     for all Y_n ∈ Y \ T do
7        Y' ← Y' ∪ RefineRobust_+(Y_n, T)
8     end for
9     Y ← Y'
10    do
11       T' ← T
12       for all Y_n ∈ Y \ T do
13          if ActR(Y_n, T) ≠ ∅ then
14             T' ← T' ∪ {Y_n}
15          end if
16       end for
17    while T' ≠ T
18    i ← i + 1
19 end while
20 return Y
```

Algorithm 6: Basic robust reachability solver for a problem $\mathcal{R}_{q\to q'}$ obtained from the transition-based product game decomposition. Input i_{max} limits the number of iterations carried out by the procedure.

To summarize, the goal of sub-problem $\mathcal{R}_{q\to q'}$ is to refine the state space partition elements in $X^{q\to q'}_{\text{refine}}$ such that $X^{q\to q'}_{\text{reach}}$ can be reached almost-surely and in finite time while avoiding the region $X^{q\to q'}_{\text{avoid}}$. Note that the individual reachability/avoidance sub-problems can overlap, i.e. a state space partition element can be the subject of refinement of more than one sub-problem. The decomposition of the product game graph is therefore not disjunct.

22.2 Robust Reachability Refinement

When analysing a problem $\mathcal{R}_{q\to q'}$ from the reachability decomposition, the product game construction can be skipped because co-safe almost-sure reachability can be decided with a backwards-search starting from the final states of the corresponding game abstraction. However, this does not reduce the exponential computational demands of the game graph

construction. In order to build an inexpensive, multi-step refinement procedure based on the decomposition approach, an additional restriction has to be introduced.

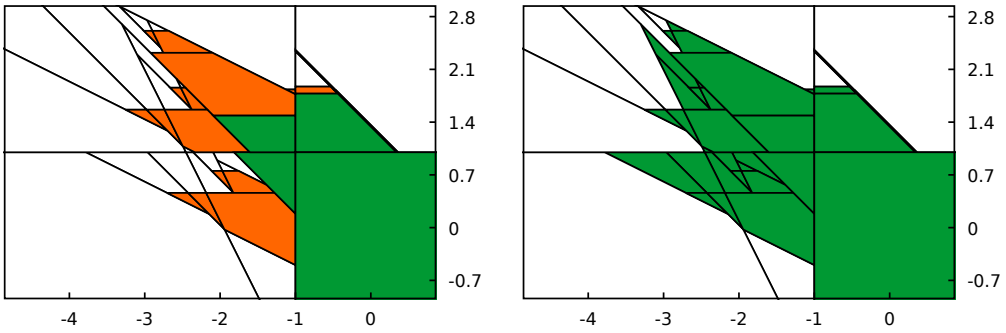
The advantages of robust dynamics for positive refinement were discussed in Section 19. A single-step positive refinement procedure based on the RefineRobust$_+$ kernel was then presented in Section 20.1. The major advantage of the robust framework is that reachability can be decided geometrically with the ActR operator and no construction of analysis of an abstraction model is required. The co-safe reachability/avoidance subproblems can therefore be analysed in the framework of robust dynamics without any abstractions, using only geometric operations. Alternating single-step robust refinement and robust analysis can therefore be performed with polynomial complexity, since PreR, AttrR and ActR can be computed in polynomial time. This allows the construction of an inexpensive, robust multi-step refinement procedure. It is presented in Algorithm 6. First, the reachability/avoidance problem $\mathcal{R}_{q\to q'}$ for the automaton transition from q to q' is set up. Polytopes in $X^{q\to q'}_{\text{refine}}$ are refined with respect to the target region $X^{q\to q'}_{\text{reach}}$ using the RefineRobust$_+$ kernel (lines 5-9). The target region is then expanded based on a ActR-condition that determines robust reachability (lines 10-17). The two steps are iterated until all polytopes satisfy the reachability objective or a given number of iterations has been exceeded (line 4). The generated sub-partition of $X^{q\to q'}_{\text{reach}}$ is then used to refine the state space partition of the original system.

The efficacy of Algorithm 6 can be improved with a few modifications. For example, the expansion of the target region can be accelerated by immediately recognizing "small" polytopes of the partition as satisfying. A polytope Z is considered to be small if $Z \ominus W$ is empty. These states cannot be targeted exclusively and can therefore also do not have self-loops. Experience shows that such states should be refined further only if they are unsafe, i.e. if $\text{Act}(Z, X^{q\to q'}_{\text{avoid}}) = U$. Small states are otherwise unlikely to void an almost-sure reachability guarantee and can therefore usually be skipped in the robust refinement even if robust reachability cannot be established.

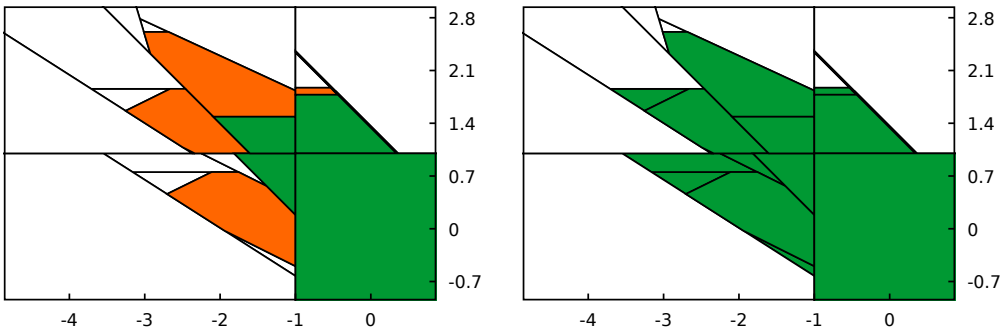
Overrefinement, leading to the creation of many (unnecessary) small polytopes, is a problem in general. Figure 7(a), left, illustrates how jagged target regions (green) can lead to the creation of many small polytopes by the RefineRobust$_+$ kernel (orange). This behaviour snowballs with subsequent iterations and affects smaller and smaller scales over time, contributing to the state space explosion and high computational cost of abstraction in the original problem after refinement. To counteract this issue, postprocessing can be applied to the output of RefineRobust$_+$. This postprocessing can either seek an over- or underapproximation to the region of RefineRobust$_+$. Overapproximations have the potential to additionally accelerate progress but the guarantees of robust refinement are generally lost. Underapproximations lead to slower progress but have the advantage that the robust guarantees are preserved.

Figure 7(b) shows an overapproximation of the refinement from 7(a) obtained from post-

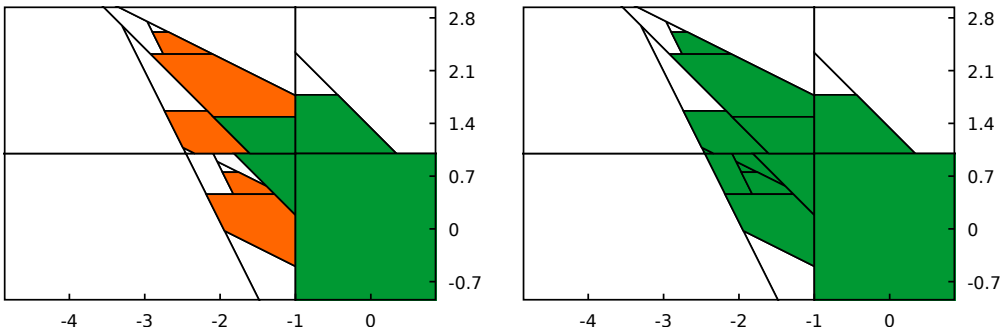
(a) no postprocessing



(b) convex hull



(c) small state suppression, one iteration



(d) small state suppression, two iterations

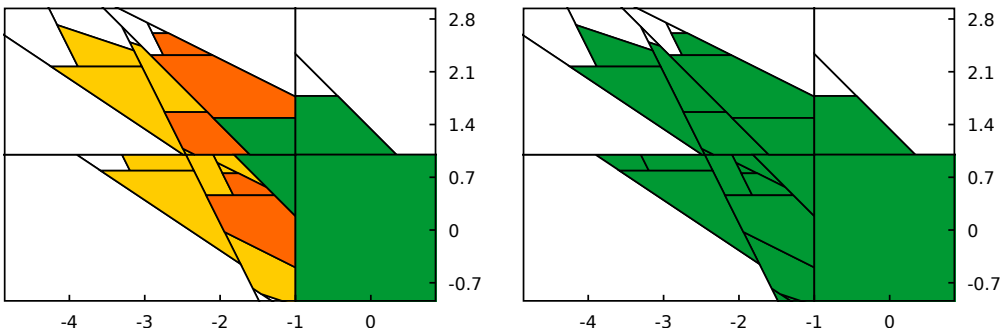


Figure 7: Left column: Illustration of how a jagged positive refinement target (green) causes small-scale overrefinement and how postprocessing of the refinement kernel affects the generated polytopes (1st application orange, 2nd yellow). The analysis progress of the original system (double integrator from Section 26) after refinement is shown on the right (green). Discussed in Section 22.2.

processing of the AttrR-region of the RefineRobust$_+$ step with a simple convex hull computation. In the example, the overapproximation has worked out and progress is made in the original system (right). Figure 7(c), left, shows an underapproximation that removes all small polytopes from the AttrR-region of the RefineRobust$_+$ step. The benefit of this postprocessing is not just the reduction of the number of polytopes in the AttrR-region but the reduction of its jaggedness. This additionally reduces the number of polytopes that have to be generated for the convex partitioning of the remaining region of the refined polytope. As seen in the right panel, less progress is made in the original system but this can be compensated for by increasing the number of iterations in the robust reachability solver. Figure 7(d), left, shows the same problem but with two instead of one robust refinement iterations applied with small polytope-suppression postprocessing. Subsequent analysis of the refined original system (right) reveals more progress than both no and convex hull postprocessing after one step and no overrefinement of small scales is discernible.

22.3 Layered Robust Reachability Refinement

A multi-step refinement scheme for reachability problems can be realized by iterating the robust refinement procedure more than once. Another possibility is to apply a layer decomposition based on the robust predecessor, the operator that encapsulates one-step robust reachability. A layered approach was originally used by Svoreňová et al. (2017) in the full probabilistic setting using Pre as the layer-generating operator and is adopted here in the robust framework to further improve the performance of the robust reachability solver. The idea is to generate layers around the target region using a PreR recurrence relation such that k-th layer is robustly reachable from every state in layer $k+1$. Refining each layer to enable such transitions in the game abstraction, a strategy can be constructed that leads a trace toward the target by moving from layer to layer inwards. The layered refinement procedure solves multi-step reachability by solving a series of single-step reachability problems. If single-step reachability can be guaranteed for every single-step layer problem, multi-step reachability can be guaranteed for the original reachability target.

The robust reachability problem of the k-th layer of problem $\mathcal{R}_{q\to q'}$ is given by

$$\mathcal{L}^k_{q\to q'} = (X^{k,q\to q'}_{\text{reach}},\, X^{k,q\to q'}_{\text{refine}},\, X^{q\to q'}_{\text{avoid}})$$

where the reach- and refine-polytope sets are generated by

$$\begin{aligned} X^{k,q\to q'}_{\text{reach}} &= X^{k-1,q\to q'}_{\text{refine}} \cup X^{k-1,q\to q'}_{\text{reach}} \quad \text{and} \\ X^{k,q\to q'}_{\text{refine}} &= \left\{ X_i \cap \text{PreR}(X,\, U,\, X^{k,q\to q'}_{\text{reach}}) \mid X_i \in X^{q\to q'}_{\text{refine}} \right\}. \end{aligned}$$

The recurrence is initialized with assignments $X^{0,q\to q'}_{\text{reach}} = X^{q\to q'}_{\text{reach}}$ and $X^{0,q\to q'}_{\text{refine}} = \emptyset$. An example for the layer-generation is given in Figure 8.

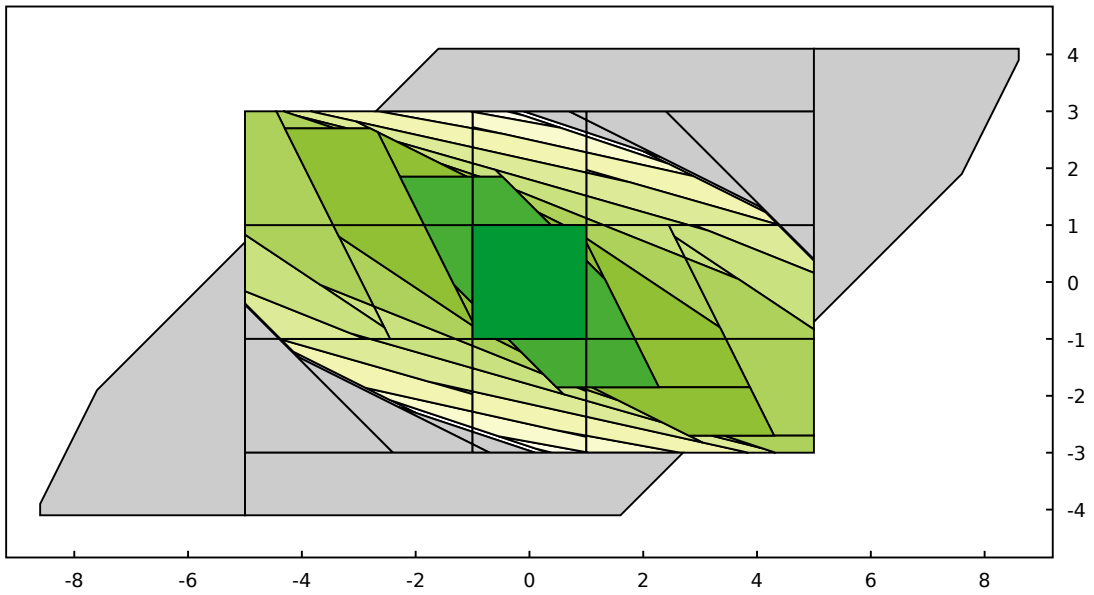


Figure 8: Layers (shaded green) generated with the PreR operator and a 5% shrunk U for the double integrator system from case study 26. 8 layers are generated until convergence. The partition consists of 106 parts in X.

The advantage of the layer decomposition is that the individual layer systems are generally simpler than the original reachability problem. There are less polytopes to refine in each system and the layers are fully independent, which allows parallelization. The PreR-based structure prescribed by the layer composition has a stabilizing effect on the refinement because it limits the occurrence of jaggedness by removing the connection between the expanding target region and the refinement kernel. Optimizations from standard robust refinement can also be used to solve the layer problems, such as skipping of small-but-safe polytopes and postprocessing of $\mathrm{RefineRobust}_+$. Experience shows that refinement performance benefits from shrinking the layer-generating control space polytope slightly to combat the occurrence of ϵ-limit behaviour on the outer edges of the layers. Removing the extreme control inputs at the (closed) edges of the control space has been found to reduce the generation of small state space polytopes at the edges of the layer regions in practice.

22.4 Transition Selection

If a solution for every transition of the objective automaton has been found through refinement, the LSS can be fully analysed with respect to the objective specification due to the strategy construction outlined in Section 22.1. However, the individual reachability sub-problems are not completely independent as they initially share the state space partition of the original LSS. While it is possible to solve the sub-problems independently and then combine the resulting state space partitions into a single partition, this combination is likely to generate many additional polytopes. Solving the reachability systems

sequentially, using only one state space partition that is handed from refinement procedure to refinement procedure, can be beneficial as the refinement applied to achieve one transition may provide a partial solution to a following sub-problem.

Satisfying paths through the objective automaton can be plentiful even when considering stuttering equivalence. While it is generally not sufficient for the determination of X_{init} to pick one satisfying path and only refine with respect to the transitions that occur along this specific path, this approach to selecting transition for refinement can lead to faster availability of partial analysis results than a breadth-first exploration of the automaton starting from its initial state.

An Interactive Implementation

Part of this work has been the creation of a web-based application that allows interactive exploration of the considered LSS abstraction-analysis-refinement procedure. Its purpose is both educational and experimental. The application supports customizable problem setups in 1 and 2 dimensions and gives the user control over the state space refinement and analysis as well as insights into the product game and system dynamics. This chapter motivates its development and presents the main features.

23 The Case for Interactive Exploration

Visualizations play an important role in scientific research. Graphical representations of data leverage the cognitive abilities of the human brain to process information quickly and detect patterns. The visual interpretation of theoretical concepts can provide valuable perspectives and aid in building intuition for typical behaviour and characteristics of a problem.

Interactivity can resolve the dilemma that arises when large amounts of information have to be displayed together. Instead of showing all information at once, an interactive visualization can provide only a partial view of a problem at any given time while allowing the user to selectively add and remove information as desired. All information can be made accessible this way without overloading the visualization with content. Interactive exploration is useful in educational contexts, helping beginners in a field to grasp important concepts and relationships while also encouraging experimentation. The ability to quickly set up a problem and explore, it results not just in a lowered barrier of entry but can also support understanding through immediate and visual feedback to user inputs. But the usefulness of interactive visualization extends to research as well. The ability to inspect a system of interest in detail can bring forth new approaches to solve a given problem, support the process of understanding and interpretation of experimental results and assist in testing ideas and resolving practical issues.

The focus of this work has been the development of new refinement heuristics for the LSS verification problem. Insights gained from graphical representations of the LSS in direct connection with its abstraction model have aided this task substantially. Illustrations of the connection between the system dynamics and the game construction helped to understand changes in the game graph in response to refinement. The ability to interactively inspect the game graph in its entirety and relate its states and actions to the geometry of the state- and control space has been valuable for the recognition of interesting patterns and the understanding of previously incomprehensible behaviour.

Of course, visualizations on a 2-dimensional screen limit the range of problems that can reasonably be considered. Systems of dimension 3 and higher have to be projected into

2-dimensions for geometric representations which results in complex and potentially ambigous representations. The complexity of problems must not be too high, as the computational demands grow quickly with the problem size and interactivity only makes sense when loading times are short. The restriction to low-dimensional systems and simple objectives is not seen as an issue for the purpose of designing refinement procedures or the educational value of an interactive implementation. Linear dynamics is generally well understood with only little potential of unexpected behaviour in higher dimensions and the geometric dimensionality is only indirectly visible in the abstraction model. Patterns encountered in game graphs derived from 1- and 2-dimensional systems are unlikely to be fundamentally different than those emerging in higher dimensions. System objectives are often variations and combinations of a basic set of standard objectives such as reachability, safety or recurrence. A refinement procedure that can handle these basic building blocks can reasonably be expected to be able to handle objectives constructed from them, e.g. through a decomposition approach.

24 Platform Selection

To implement an interactive system exploration tool, the web browser was chosen as a platform. Browsers are tailored to the display of visual interfaces and have built-in vector graphics capabilities. They are arguably the most accessible platform of the modern age, as browsers are available for virtually device with a screen. Interactivity is achieved through the JavaScript programming language which is natively tied to the browser's interface representation, the document object model. JavaScript is an easy-to-use language which offers good performance due to JIT compilation. Debugging tools are shipped with every modern browser. Using node.js, JavaScript can also be run outside of the browser.

However, for the purposes of this work, there was also an important disadvantage associated with the choice of the browser. Native browser applications are restricted to JavaScript without the possibility to interface with other programming languages via direct-call APIs or wrappers. The only way to connect JavaScript to the outside world from within the browser is through a client-server infrastructure. This was particularly an issue given that JavaScript libraries for control problems of hybrid systems could not be found. E.g. MPT (Herceg et al. 2013) is written in MATLAB and TuLiP (Filippidis et al. 2016) in Python, so neither is directly accessible from within the browser. While node.js can resolve these issues, it limits the available graphical capabilities and accessibility.

Nevertheless, the browser was still chosen. The range of problems under consideration is limited to relatively simple, low-dimensional problems due to the representation- and responsiveness requirements of interactive visualization. Time was invested to implement the required operations from convex geometry as well as game- and automaton data

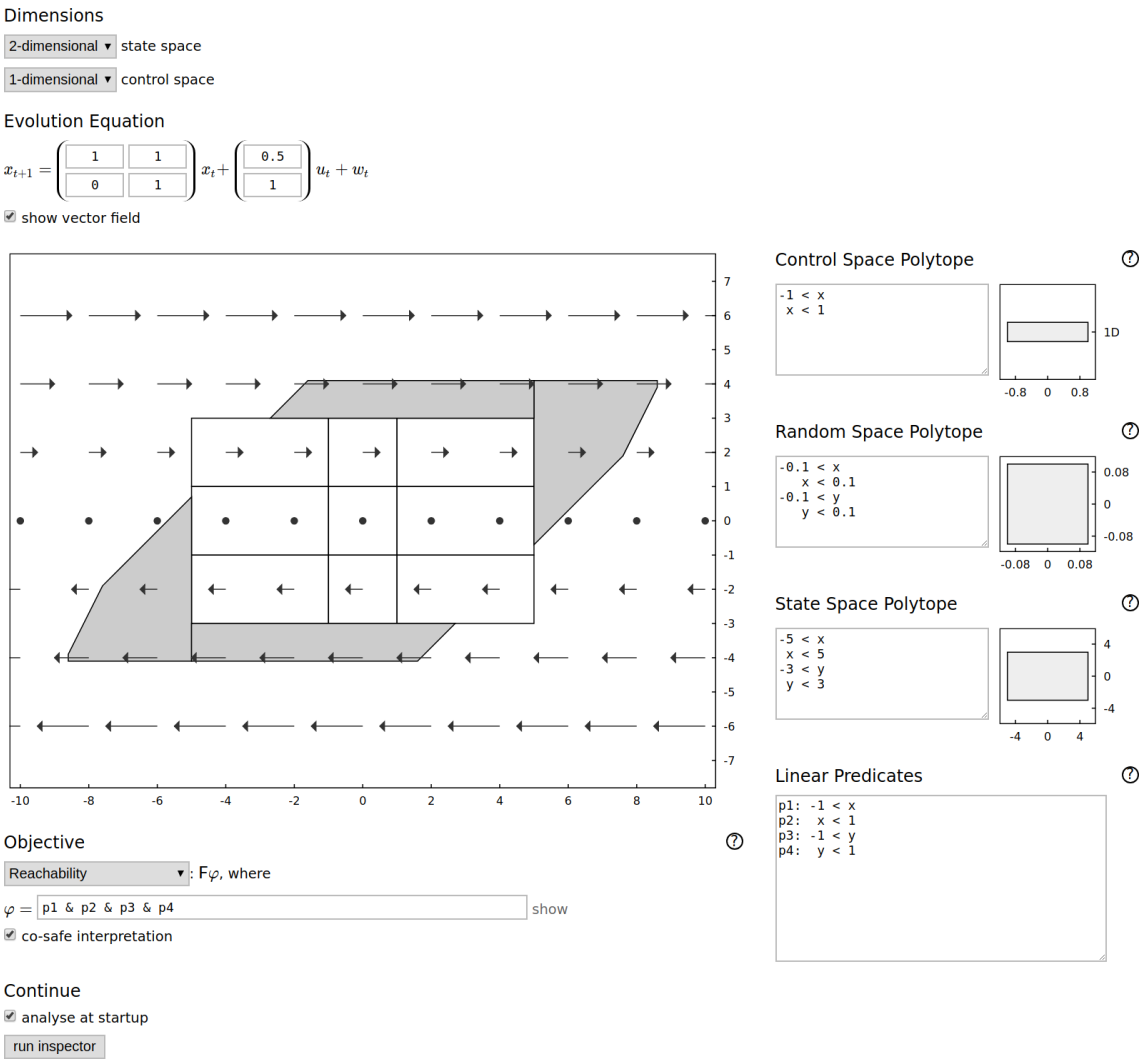


Figure 9: The problem setup screen of the interactive application.

structures. This also provided a valuable learning opportunity for the author of this thesis and has deepened his understanding of the given problem and its solution procedure.

25 Features

Figure 9 shows the problem setup screen of the created application. The user is able to specify the state-, random- and control space in 1 and 2 dimensions as well as the evolution equation and the set of linear predicates used in the initial partitioning of the state space. A preview of the state space configuration provides instantaneous feedback to user input. The objective can be chosen from a list of predefined GR(1) formulae, LTL-to-automaton translation is not implemented. While the structure of the temporal logic formulae is fixed, they contain placeholder variables that can be defined by the user by combining the linear predicates with connectives from propositional logic.

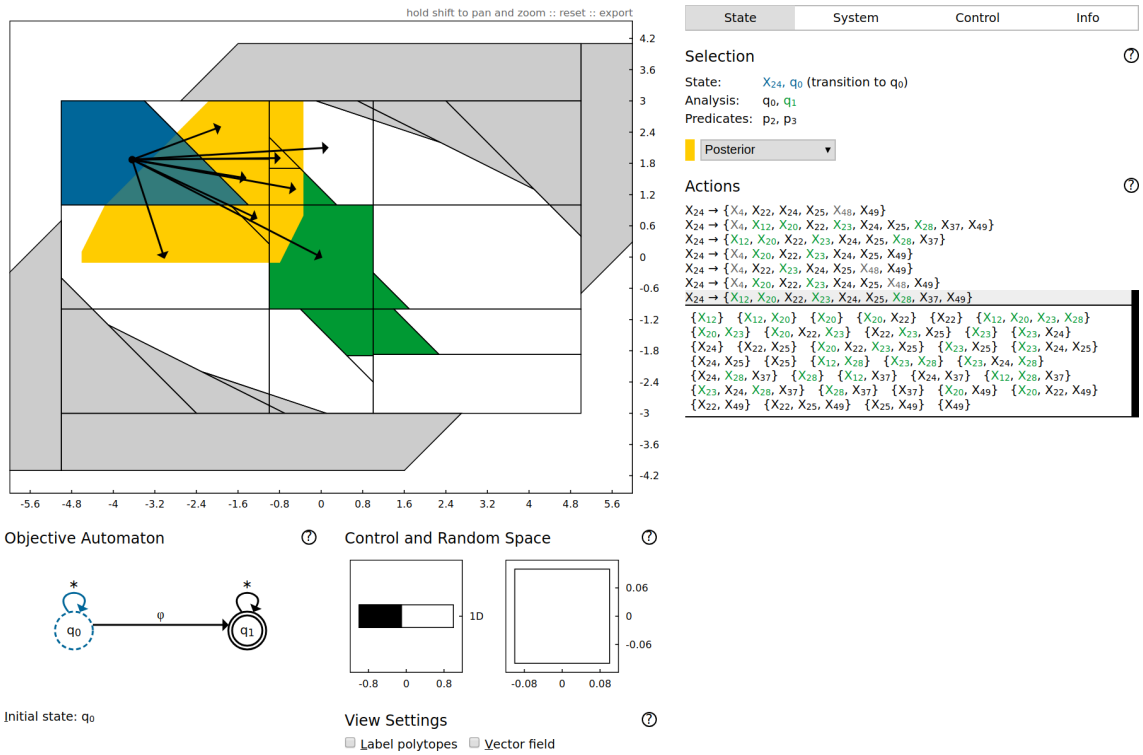


Figure 10: The problem exploration screen of the interactive application.

Figure 10 shows the exploration screen, which is shown once the problem setup has been submitted by the user. Its main interface element is a visualization of the state space and its partition. The view is customizable by panning and zooming and can be exported as an SVG image. The selection of state space partition elements reveals information about the associated product game states in the "State" tab to the right of the system view. Player actions from the game graph are shown there and ActC and PreP regions can be highlighted. The product game can be explored further with the selection of a state from the objective automaton, displayed below the state space view. State-space elements are coloured according to the membership of the corresponding player 1 states in P_{yes}, P_{no} and $P_?$.

Accessible in the tabs on the right are also controls for analysis and refinement procedures. The presented holistic refinement algorithms from Section 20 as well as the robust refinement for automaton transitions in the product game from Section 22 are implemented and can be applied to the system with custom configurations. A snapshot system can save the current state of the system abstraction and restore it later, allowing the user to try and compare different refinement approaches. Brief reports with performance metrics and information about the product game are logged after every refinement and analysis invocation. Traces can be sampled with a synthesized round-robin controller and the individual trace steps inspected.

Built-in help texts provide assistance and explanations of the interface to the user. They are available in the appendix of this document.

Case Studies

The presented refinement procedures are applied in 2 case studies for comparison and performance assessment. Particular attention is given to the robust positive refinement for reachability problems. Multiple configurations are compared in the first case study, a basic reachability problem with 2-dimensional double integrator dynamics. This exact problem was previously examined in detail by Svoreňová et al. (2017). A second case study is concerned with a more complex objective with 3 automaton states and a 2-dimensional control space.

All case studies were run on a machine with an Intel i5-4500U Haswell CPU (1.6 GHz base clock, 2.6 GHz turbo) and 8 GB of main memory. The JavaScript routines from the interactive system explorer were run inside node.js without parallelism.

26 Double Integrator

The double integrator case study of Svoreňová et al. (2017) is replicated. The system under consideration evolves according to the difference equation

$$\boldsymbol{x}_{t+1} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \boldsymbol{x}_t + \begin{bmatrix} 0.5 \\ 1 \end{bmatrix} \boldsymbol{u}_t + \boldsymbol{w}_t \ ,$$

where $\boldsymbol{x}_t \in X = [-5, 5] \times [-3, 3]$, $\boldsymbol{u}_t \in U = [-1, 1]$ and $\boldsymbol{w}_t \in W = [-0.1, 0.1]^2$. The problem is therefore mixed-dimensional, with a 2-dimensional state space and 1-dimensional control space. The objective is to eventually reach the polytope $[-1, 1]^2$ almost-surely, i.e. a reachability problem with formula

$$\mathsf{F}(\pi_1 \wedge \pi_2 \wedge \pi_3 \wedge \pi_4) \ ,$$

where the linear predicates π_1, π_2, π_3 and π_4 correspond to halfspaces governed by the inequalities $x \leq 1$, $-x \leq 1$, $y \leq 1$ and $-y \leq 1$. The co-safe interpretation is chosen and the reachability automaton from Table 1 used. The initial state space decomposition and the extended state space are visualized in Figure 11.

Svoreňová et al. (2017) have demonstrated that positive robust refinement is viable for this problem, so it is used here as a testbed for different configurations of robust refinement techniques. Double integrator dynamics were also used as an example by Aydin Gol et al. (2015) but in a non-probabilistic setting.

26.1 Negative Refinement

Before utilizing positive refinement procedures, negative attractor refinement is applied. The initial negative region is made up of the outer polytopes of the system. 3 iterations

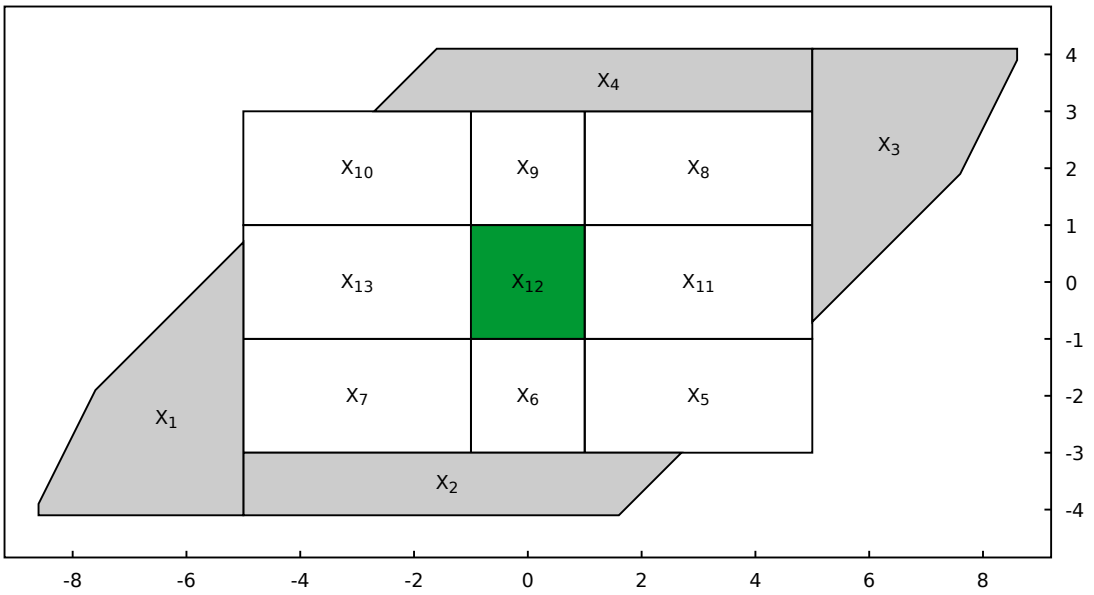


Figure 11: The double integrator test system and its initial partition. Polytopes X_1 to X_4 (grey) are outer states from the decomposition of $X_{\text{ext}} \setminus X$. The reachability target is X_{12} (green).

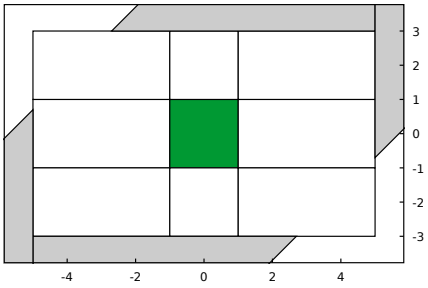
of the refinement-abstraction-analysis cycle are shown in Table 3. As guaranteed by the refinement method, the attractor regions are immediately recognized as members of $X_{\text{no}}^{q_0}$. After 3 iterations, the attractor has converged and no additional progress is made.

In addition to a depiction of the state space partition after each refinement step in Table 3, the elapsed time since problem initialization and the number of polytopes in the current state space partition are stated above the figures. Also reported are the size of the 2½-player game abstraction as player 1 and 2 state and action counts, the elapsed time during refinement application, game graph construction and analysis as well as the volume fraction of the regions $X_{\text{yes}}^{q_0}$ (yes), $X_{\text{no}}^{q_0}$ (no) and $X_{?}^{q_0}$ (maybe) with respect to the state space polytope X. The table shows that refinement, abstraction and analysis in all iterations took less than 1 second. The initial analysis and the negative refinement required 1 second of run time in total.

26.2 Positive Robust Refinement

The first positive refinement procedure tested on the double integrator is configured to resemble that of Svoreňová et al. (2017) (see their Chapter 5 and Figure 6). Their robust refinement kernel was driven by control regions selected based on the player 1 actions of the game and they applied the kernel 3 times between each analysis. The robust predecessor of the recognized yes-region was used to refine the system additionally. Here, only the kernel refinement is applied but with 2 applications between each analysis as the sampling-based control region selection is generally more effective than an action-based selection and progress per iteration is therefore similar.

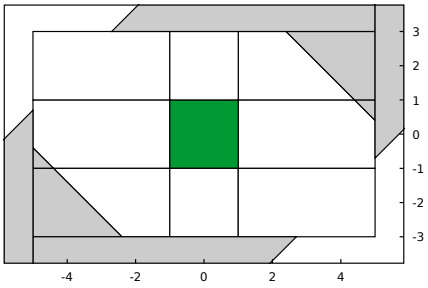
1) 0:00 elapsed, 9 polygons



Player 1:	28 states, 44 actions
Player 2:	26 states, 216 actions
Refinement:	-
Game Graph:	0:00
Analysis:	0:00

6.7% yes, 0.0% no, 93.3% maybe

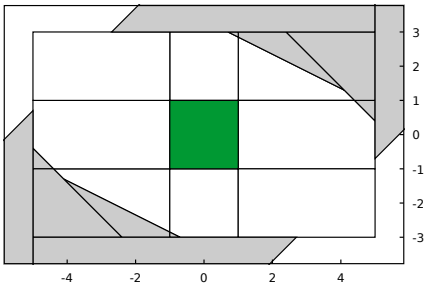
2) 0:00 elapsed, 13 polygons



Player 1:	36 states, 60 actions
Player 2:	38 states, 362 actions
Refinement:	0:00
Game Graph:	0:00
Analysis:	0:00

6.7% yes, 11.3% no, 82.1% maybe

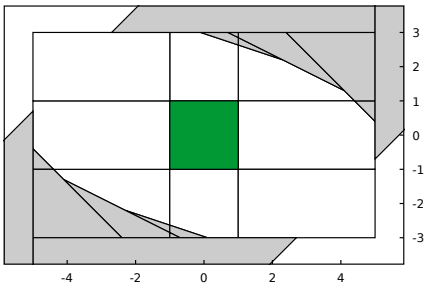
3) 0:01 elapsed, 17 polygons



Player 1:	44 states, 80 actions
Player 2:	50 states, 592 actions
Refinement:	0:00
Game Graph:	0:00
Analysis:	0:00

6.7% yes, 16.1% no, 77.3% maybe

4) 0:01 elapsed, 21 polygons

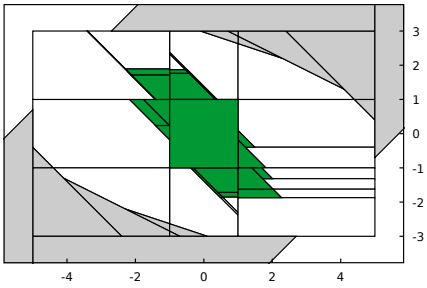


Player 1:	52 states, 104 actions
Player 2:	66 states, 794 actions
Refinement:	0:00
Game Graph:	0:00
Analysis:	0:00

6.7% yes, 17.1% no, 76.2% maybe

Table 3: Initial analysis and negative attractor refinement for the double integrator system. The refinement procedure converged after 3 iterations. Polytopes in $X^{q_0}_{\text{yes}}$ are coloured green, polytopes in $X^{q_0}_{\text{no}}$ grey and white polytopes are from $X^{q_0}_{?}$. See Section 26.1 for discussion.

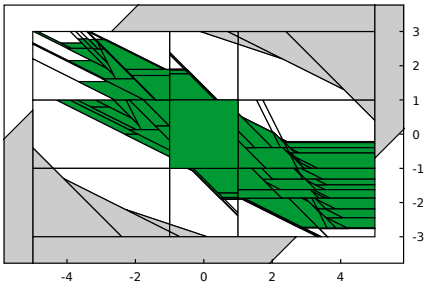
5) 0:04 elapsed, 48 polygons



Player 1:	106 states, 476 actions
Player 2:	407 states, 6188 actions
Refinement:	0:00
Game Graph:	0:03
Analysis:	0:00

13.9% yes, 17.1% no, 68.9% maybe

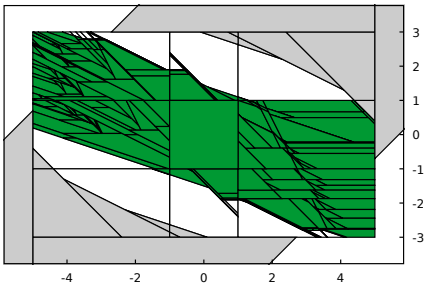
6) 2:07 elapsed, 166 polygons



Player 1:	342 states, 3723 actions
Player 2:	3519 states, 79332 actions
Refinement:	0:02
Game Graph:	1:59
Analysis:	0:03

36.5% yes, 17.1% no, 46.4% maybe

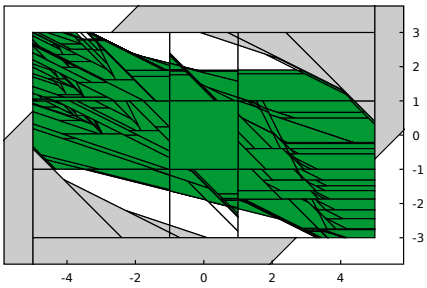
7) 5:59 elapsed, 301 polygons



Player 1:	612 states, 5902 actions
Player 2:	5459 states, 111705 actions
Refinement:	0:09
Game Graph:	3:40
Analysis:	0:03

54.9% yes, 17.1% no, 28.0% maybe

8) 9:41 elapsed, 362 polygons



Player 1:	734 states, 3096 actions
Player 2:	2458 states, 54578 actions
Refinement:	0:07
Game Graph:	3:33
Analysis:	0:02

67.0% yes, 17.1% no, 15.9% maybe

Table 4: Refinement of transition $q_0 \to q_1$ of the double integrator system using 4 iterations of positive robust single-step refinement, applied twice in each iteration without robust target expansion. See Section 26.2 for discussion.

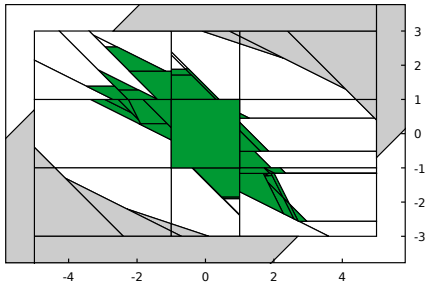
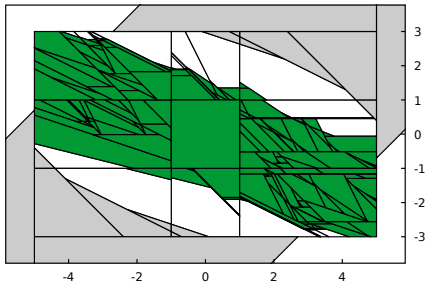
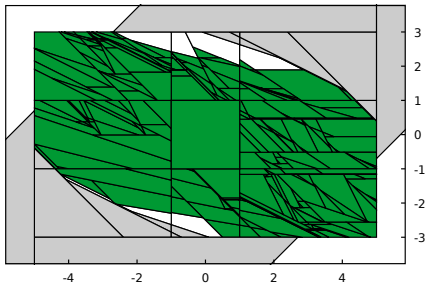
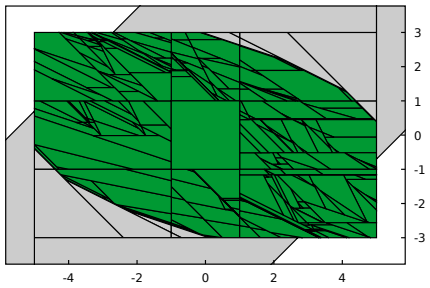
5) 0:11 elapsed, 68 polygons	Player 1:	146 states, 885 actions
	Player 2:	796 states, 13251 actions
	Refinement:	0:00
	Game Graph:	0:10
	Analysis:	0:00
	18.4% yes, 17.1% no, 64.5% maybe	
6) 4:32 elapsed, 236 polygons	Player 1:	482 states, 6293 actions
	Player 2:	6003 states, 141782 actions
	Refinement:	0:06
	Game Graph:	4:08
	Analysis:	0:06
	53.9% yes, 17.1% no, 29.0% maybe	
7) 6:09 elapsed, 333 polygons	Player 1:	676 states, 4114 actions
	Player 2:	3567 states, 53441 actions
	Refinement:	0:06
	Game Graph:	1:29
	Analysis:	0:02
	77.0% yes, 17.1% no, 5.9% maybe	
8) 6:18 elapsed, 361 polygons	Player 1:	732 states, 1367 actions
	Player 2:	682 states, 7436 actions
	Refinement:	0:01
	Game Graph:	0:08
	Analysis:	0:00
	82.5% yes, 17.1% no, 0.3% maybe	

Table 5: Refinement of transition $q_0 \to q_1$ of the double integrator system using 4 iterations of positive robust two-step refinement with robust target expansion. See Section 26.2 for discussion.

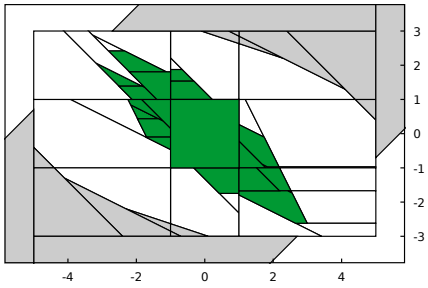
The progress from 4 iterations of refinement together with the familiar performance metrics are shown in Table 4. They are labeled iterations 5 to 8 to indicate that the procedure continues after the negative refinement. The size of the state space partition doubles after the first iteration of refinement and then more than triples in the next iteration. A large amount of small states are created to the upper left and lower right of the target region. The asymmetry of the partitioning outside of the robust attractor region is due to the implementation of the geometric operations. Here, the partitioning in the lower right corner is favourable and leads to additional progress that is not seen in the upper left. This allows for faster progress in the upper right half of the state space in the 2 subsequent iterations whereas many small states are created in the upper left. The polygon count exceeds 300 after 3 iterations of positive refinement and increases only modestly by 61 in the last. After the final iteration, 84% of the state space volume has been categorized as a subset of X_{init} or $X \setminus X_{\text{init}}$. The elapsed time is just under 10 minutes with most of the time spent in the construction of the game abstraction.

Compared to the results of Svoreňová et al. (2017) it can be noted that the amount of progress after 4 iterations is similar, but the partition generated here has more states. This can partially be attributed to the omission of the PreR refinement which seems to reduce overall jaggedness around the AttrR-regions in their demonstration.

The decomposition into robust reachability problems for specific automaton transitions enables multi-step refinement with an expanding target region. The double integrator problem is already a reachability problem and co-safe, therefore the decomposition is trivial here and the transition $q_0 \to q_1$ is chosen. A single-step method was applied twice between analyses in the previous experiment, now a two-step method is applied once. 4 iterations of this refinement procedure are shown in Table 5. Immediately in the first iteration, progress is faster than with the single-step scheme with 20 additional polytopes generated. The second iteration more than triples the partition size which leads to a costly game graph abstraction taking 4 minutes. Progress after this iteration however is comparable to that of the third iteration of the single-step method. The next 2 iterations grow the partition size by 125, with relatively short game graph construction times due to game simplification taking place based on the fast progress in the first 2 iterations (the number of game actions decreases after the second positive refinement). Only 0.3% of the state space volume remain undecided after 4 iterations with an elapsed time that is more than 3 minutes shorter than that of the single-step method.

The two-step refinement is applied again with two modifications: First, any state Y with $Y \ominus W$ is not further refined unless the probability of reaching the no-region is non-zero for all control inputs. Second, the RefineRobust_+ partition is postprocessed such that polytopes Y where $Y \ominus W$ is empty are removed from the AttrR-region, as shown in Figure 7(d). 4 iterations of this modified refinement are presented in Table 6. As expected, progress per iteration is slower than without postprocessing but the state space explosion is significantly reduced because the refinement is able to handle jaggedness

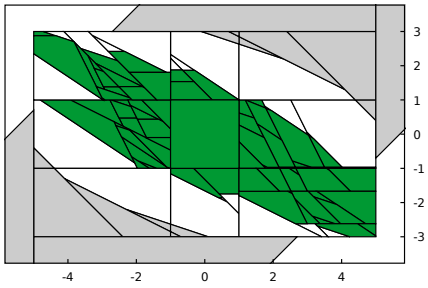
5) 0:05 elapsed, 51 polygons



Player 1:	112 states, 507 actions
Player 2:	435 states, 6147 actions
Refinement:	0:00
Game Graph:	0:04
Analysis:	0:00

18.5% yes, 17.1% no, 64.3% maybe

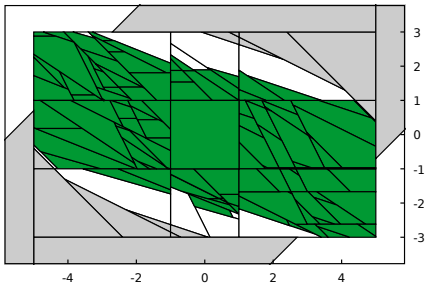
6) 0:26 elapsed, 99 polygons



Player 1:	208 states, 1360 actions
Player 2:	1221 states, 21859 actions
Refinement:	0:02
Game Graph:	0:18
Analysis:	0:01

42.4% yes, 17.1% no, 40.4% maybe

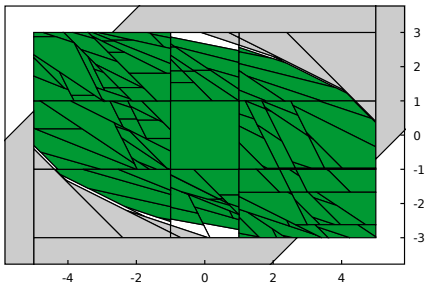
7) 0:41 elapsed, 134 polygons



Player 1:	278 states, 1291 actions
Player 2:	1069 states, 16174 actions
Refinement:	0:02
Game Graph:	0:13
Analysis:	0:00

65.6% yes, 17.1% no, 17.3% maybe

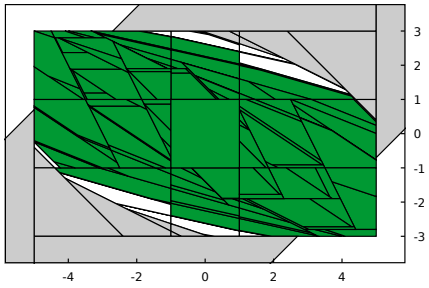
8) 0:48 elapsed, 161 polygons



Player 1:	332 states, 962 actions
Player 2:	671 states, 8157 actions
Refinement:	0:01
Game Graph:	0:05
Analysis:	0:00

80.1% yes, 17.1% no, 2.7% maybe

Table 6: Refinement of transition $q_0 \to q_1$ of the double integrator system using 4 iterations of positive robust two-step refinement with robust target expansion and small state suppression. See Section 26.2 for discussion.

5)	1:24 elapsed, 166 polygons	

Player 1:	342 states, 4364 actions
Player 2:	4177 states, 79056 actions
Refinement:	0:03
Game Graph:	1:16
Analysis:	0:05
76.3% yes, 17.1% no, 6.6% maybe	

Table 7: Refinement of transition $q_0 \rightarrow q_1$ of the double integrator reachability system using a layer decomposition generated by the robust predecessor and four-step positive robust refinement with robust target expansion small state suppression. See Section 26.3 for discussion.

without propagating it to smaller scales. After 4 iterations that take only a tenth of the time of the unmodified procedure, less than half the number of partition elements are generated and only 2.7% of the system is left undecided.

Similar partitioning is also achieved if a single iteration with an 8-step procedure is carried out (not shown), although the overall elapsed time is longer because the game construction cannot benefit from intermediate analysis results. Applying an overapproximating postprocessing using the convex hull as shown in Figure 7(b) resulted in varying performance (not shown). While the overapproximation would sometimes work out and the system would continuously progress, the procedure could also get stuck due to the missing progress guarantee.

26.3 Positive Robust Refinement with Layer Decomposition

The layer decomposition offers an alternative approach to multi-step refinement. It is now applied together with the postprocessed, underapproximated robust refinement from the previous section. The robust predecessor is chosen as the layer-generating operator. Because reachability from the outer layers depends on the solution of the inner layers, up to 4 robust refinement steps are applied to each layer in order to increase the likelihood of successful refinement.

Table 7 shows the partitioning and performance metrics for a layered decomposition refinement. The PreR recurrence converged after 7 layers. The size of the resulting state space partition is similar to that of the final iteration of the postprocessed two-step refinement but the elapsed time to solution is longer, since no intermediate analysis results are available for game simplification. It also fails to guarantee almost-sure reachability for some polytopes from the 2 outermost layers. This is likely related to the small polytope suppression as these polytopes are relatively long and thin and any further refinement is

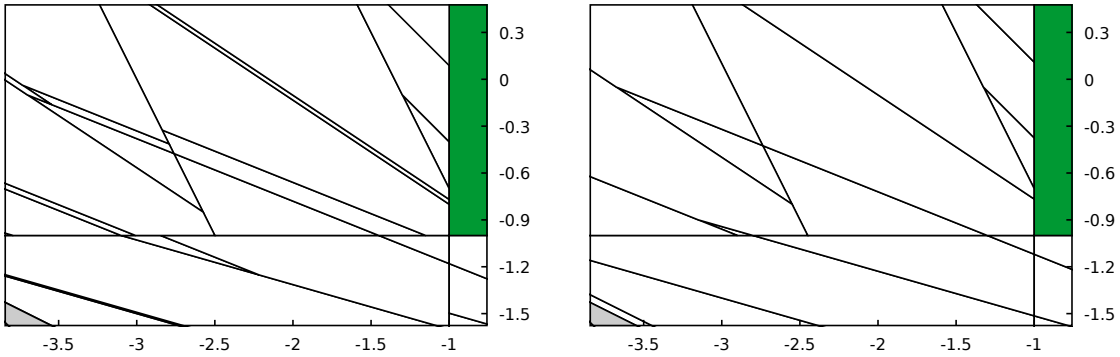


Figure 12: Illustration of the impact of ϵ-limit behaviour (Section 16.2) when using U unmodified to generate layers for the double integrator example (left). The creation of long, thin states along the edges of the layers during the inner refinement of each layer can be combatted by slightly shrinking U as demonstrated by the partition on the right.

rejected by the postprocessing. Long and thin states appear also along the edges of the layer boundaries. This is due to ϵ-limit behaviour at the edge of the PreR-regions.

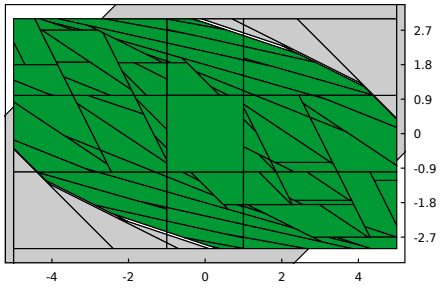
It is possible to avoid this behaviour by shrinking the control region during the layer generation. Figure 12 shows the effects of ϵ-limit behaviour on the partition in the left frame. The panel on the right shows the same layered refinement but the control space was shrunk by 5% during the generation of the layers, here from $[-1, 1]$ to $[-0.95, 0.95]$. This creates a "buffer zone" at the edges of the control space which allow the inner refinement steps to avoid the ϵ-limit behaviour. Due to the smaller resulting PreR-layers, an additional layer is required until the generator converges. The layers were shown in Figure 8, which demonstrates that a small border between the outermost layer and the no-region remains due to the shrunk U.

Table 8 shows the complete refinement results of this layer decomposition. Due to the elimination of the ϵ-limit behaviour, 37 fewer states have been created compared to the unmodified PreR decomposition. The number of states that arise just from the layer decomposition alone is 106 (see Figure 8), meaning that 23 additional polytopes were generated to enable the desired layer-to-layer transitions. The number of player 1 and 2 actions in the game graph has been reduced significantly and this is reflected in a much smaller run time of the game construction. After the analysis, 99% of the state space is decided with 29 seconds elapsed since initialization. This is faster than two-step refinement with postprocessing which profited from game simplification due to intermediate analysis results being available. It is also the smallest state space partition of all positive robust refinement configurations tried.

26.4 Comparison and Discussion

Statistics from 8 runs of the 5 presented refinement configurations are given in Table 9. The negative refinement was shared by all procedures, then the different methods were

5) 0:29 elapsed, 129 polygons



Player 1:	268 states, 2565 actions
Player 2:	2415 states, 36246 actions
Refinement:	0:01
Game Graph:	0:25
Analysis:	0:02

81.8% yes, 17.1% no, 1.0% maybe

Table 8: Refinement of transition $q_0 \to q_1$ of the double integrator reachability system using a layer decomposition generated by the robust predecessor with a 5%-shrunk U and four-step positive robust refinement with robust target expansion and small state suppression. See Section 26.3 for discussion.

applied for 4 iterations or 1 iteration if layer decomposition was applied. The runs that were discussed in detail before are the (lower) median runs with respect to the elapsed time after the final iteration.

The most striking observation from this table is the variability in partition size and elapsed time exhibited by the procedures not based on a layer decomposition. Randomization in the RefineRobust$_+$ kernel apparently has a significant effect on the refinement performance. Elapsed times for both refinement methods without postprocessing vary by more than 15 minutes and the state space partition size by up to 100 polytopes. The variability of the non-layered procedure with small state suppression is much less, but still varies by half a minute. In contrast, the layered methods vary only by a few seconds and less than 10 polytopes in the state space partition. The results of the layer decomposition with shrunk control space show no variability at all in the volume of decided state space and less than 3 seconds of variability in the total elapsed time. One can conclude that the layer decomposition has a strong stabilizing effect on the refinement that counters the variability of the randomization of RefineRobust$_+$ and results in methods that deliver consistently good and dependable performance.

Missing in the implementation is the leveraging of robust progress guarantees by the game graph abstraction after robust refinement. This would speed up the methods further as robust reachability could be established without the need for abstraction. Player actions for many polytopes would then not have to be computed and game construction, which is the bottleneck of the entire solution scheme, would be cheaper.

An issue that arose during the performance benchmark was numerical instability when operating on very small polytopes. 3 runs of the single-step procedure had to be repeated due to a faulty game construction detected by the implementation. In every case the fault was a problem in the calculation of the PreP-regions for extremely small polytopes. This is either an instability in the geometry library (not unthinkable considering that it

Iter.		**single-step ×2** min / avg / max	**two-step** min / avg / max	**two-step** **small suppression** min / avg / max	**layers** **four-step** **small suppression** min / avg / max	**shrunk layers** **four-step** **small suppression** min / avg / max
1-4 neg.	polytopes			21 / 21 / 21		
	elapsed			0:01 / 0:01 / 0:01		
	% decided			23.8 / 23.8 / 23.8		
5	polytopes	48 / 50 / 53	68 / 85 / 97	49 / 54 / 60	165 / 168 / 171	128 / 130 / 133
	elapsed	0:04 / 0:04 / 0:05	0:11 / 0:21 / 0:36	0:04 / 0:05 / 0:07	1:19 / 1:24 / 1:27	0:28 / 0:29 / 0:31
	% decided	30.7 / 31.0 / 31.4	35.5 / 38.4 / 39.6	35.5 / 36.7 / 39.2	93.4 / 95.0 / 96.7	99.0 / 99.0 / 99.0
6	polytopes	166 / 194 / 239	227 / 277 / 317	83 / 99 / 114		
	elapsed	1:55 / 4:00 / 11:56	2:16 / 6:06 / 16:36	0:18 / 0:25 / 0:36	-	-
	% decided	53.6 / 55.8 / 58.5	70.7 / 74.8 / 79.4	53.6 / 61.8 / 71.0		
7	polytopes	239 / 280 / 321	300 / 353 / 394	124 / 132 / 137		
	elapsed	4:04 / 7:14 / 17:13	3:10 / 7:24 / 17:59	0:26 / 0:41 / 0:50	-	-
	% decided	69.3 / 72.0 / 73.3	87.8 / 95.6 / 99.5	76.2 / 83.8 / 89.8		
8	polytopes	296 / 351 / 408	333 / 389 / 435	141 / 151 / 163		
	elapsed	6:15 / 12:35 / 27:24	3:26 / 8:53 / 18:07	0:31 / 0:47 / 0:58	-	-
	% decided	81.6 / 84.3 / 85.6	95.5 / 99.2 / 99.8	89.9 / 96.5 / 99.3		

Table 9: Double integrator performance results for 5 configurations of the positive robust refinement procedure. Shown are the minimum, average and maximum number of polytopes in the X-partition, elapsed time after analysis and volume-percentage of the state space that has been identified as part of X_{init} or $X \setminus X_{\text{init}}$ from 8 runs of each procedure. Tables 3 to 8 show the median runs with respect to total elapsed time in detail. Discussion in Section 26.4.

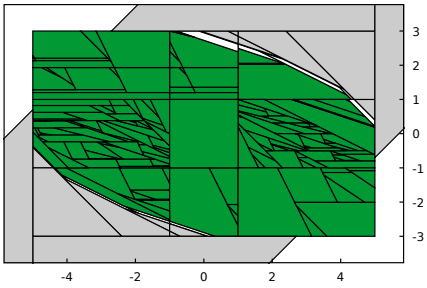
5) 4:04 elapsed, 201 polygons	Player 1: 412 states, 6660 actions
	Player 2: 6438 states, 142464 actions
	Refinement: 0:36
	Game Graph: 3:14
	Analysis: 0:13
	80.8% yes, 17.1% no, 2.1% maybe

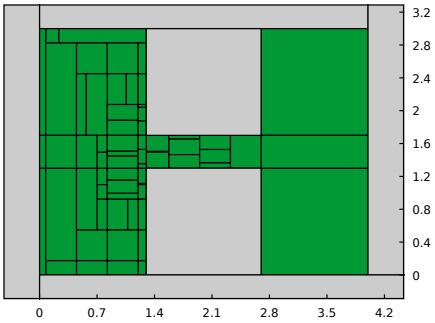
Table 10: Refinement of the double integrator reachability system with 2 applications of safety refinement followed by 5 applications of self-loop removal. See Section 26.5 for discussion.

is a custom development) and/or a general problem with polytope sizes near the smallest representable scale. Additional measures that avoid very fine partitioning could be implemented to combat this. Currently there is a size threshold under which states are not refined further but this does not stop the creation of very small states completely. Instead one could refuse any partitioning that produces a tiny polytope at the cost of refinement progress.

26.5 Neutral Refinement

The discussion of the double integrator system is concluded by a demonstration of the effectiveness of neutral refinement techniques. After the negative refinement (Table 3), safety refinement is applied twice, followed by 5 iterations of self-loop removal without intermediate analysis. The intended purpose of the safety refinement is to contain the ϵ-limit behaviour along the edges of the negative attractor regions, while the removal of self-loops should allow strategies to evolve traces inside the safe region until the target is eventually reached probabilistically. A partition of the state space resulting from such refinement is shown in Table 10 together with the familiar performance metrics. It is evident that the neutral refinement techniques alone are not competitive with layered refinement with respect to both run time and partition size. Notable is that the refinement took much longer than for any of the robust methods due to the self-loop refinement which requires the game graph to be (partially) constructed in between applications. The resulting partition is very fine around $y = 0$ and rather coarse for larger values of $|y|$. This can be explained by the dynamics of the double integrator, which has shear characteristics. The inherent motion of traces prescribed by the transformation with $\boldsymbol{A}$ is faster away from $y = 0$ where it vanishes entirely. Therefore, self-loops are more likely to occur near $y = 0$, where they can only be avoided through external control and a fine partition.

1:35 elapsed, 65 polygons



Player 1:	140 states, 6856 actions
Player 2:	6778 states, 85804 actions
Refinement:	0:02
Game Graph:	1:29
Analysis:	0:04

69.7% yes, 30.3% no, 0.0% maybe

Table 11: The corridor reachability and avoidance problem for the right room solved with layered robust refinement. See section 27.1 for discussion.

27 Corridor

In a second case study, the linear stochastic system

$$\boldsymbol{x}_{t+1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \boldsymbol{x}_t + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \boldsymbol{u}_t + \boldsymbol{w}_t = \boldsymbol{x}_t + \boldsymbol{u}_t + \boldsymbol{w}_t$$

is considered, where $\boldsymbol{x}_t \in X = [0, 4] \times [0, 3]$, $\boldsymbol{u}_t \in U = [-0.5, 0.5]^2$ and $\boldsymbol{w}_t \in W = [-0.1, 0.1]^2$. Linear predicates π_1, π_2, π_3 and π_4 are defined, corresponding to the halfspaces $x \leq 1.3$, $x \geq 2.7$, $y \leq 1.3$ and $y \geq 2.7$, respectively. With these predicates, the state space is divided into two rooms on the left and right, which are connected by a narrow corridor in the center of the state space. The propositional formulas $\phi = \pi_1$ and $\mu = \pi_2$ describe the left and right rooms, respectively, and $\theta = \neg\pi_1 \wedge \neg\pi_2 \wedge (\pi_3 \vee \pi_4)$ defines the walls that line the central corridor.

27.1 Reachability Analysis

First, the system is refined and analysed with respect to a reachability and avoidance specification. The objective is to reach the room on the right while avoiding the walls around the corridor, i.e. $(\neg\theta)\,\mathsf{U}\,\mu$. A corresponding ω-automaton for this LTL formula is given in Table 1. The problem is treated in a co-safe setting.

The challenge of this problem setup is the narrow corridor. Precise control is necessary for a trace to enter and navigate the corridor. This demands fine partitioning around and inside the corridor and restricts the applicability of postprocessing procedures that suppress small polytopes, as these polytopes are specifically required for a solution. Based on the previous good performance in the double integrator example, a PreR-based layer decomposition with 5%-shrunk control region is selected. Each layer is refined with a 4-step robust method without postprocessing. A resulting partition of this refinement configuration is shown in Table 11. The system does not exhibit ϵ-limit behaviour on the

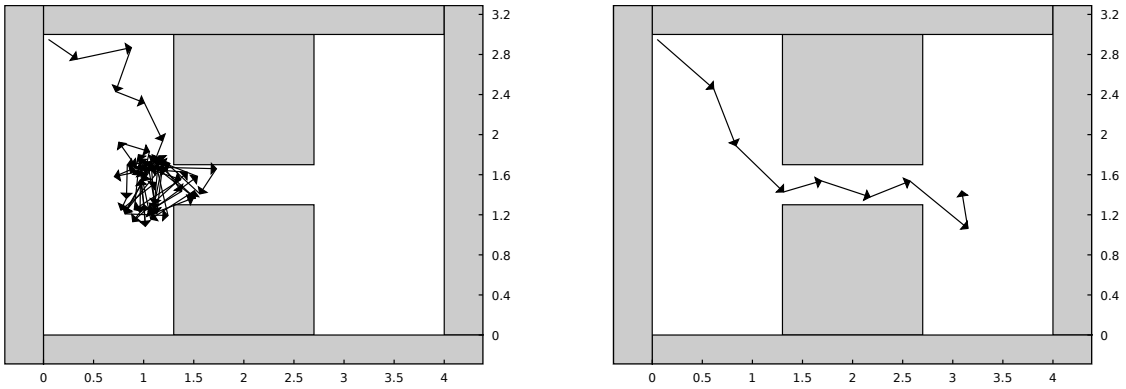


Figure 13: Trace samples for the corridor right-room reachability problem controlled with a round-robin strategy (left) and a layer-based distance minimization strategy (right). The trace are initiated near the top left corner of the state space and terminated after 60 steps or when the corresponding automaton run has transitioned into a final state.

edges of the state space and is completely solved. As expected, the partition is finer at the entrance region of the corridor than in the rest of the left room. Both run time and the size of the product game measured in player actions are similar to that of the double integrator solution in Table 7 despite a state space partition that has only a third of the number of polytopes. This is a consequence of the additional dimension of the control space, which results in more player actions per state. The increased number of polytopic operations required to compute these actions raises the cost of the game construction.

27.2 Reachability Controller

Using the round-robin controller scheme of Svoreňová et al. (2017), a control strategy is synthesized from the solution of the reachability analysis problem (Table 11). A 60-step prefix of a trace controlled by this strategy is shown in Figure 13, left. The trace is initialized in the top left of the left room and moves towards the corridor for a few steps but then gets stuck around the corridor entrance. While the trace does not violate the avoidance specification at any step, it does not manage to efficiently navigate the corridor. The issue is that the round-robin controller scheme has no built-in sense of direction. It has no problem with guaranteeing the safety of traces but in order to navigate the corridor, the trace evolution has to be controlled by "steering" to the right for multiple timesteps consecutively. The lack of coordination between the selection of actions in the round-robin controller means that one might wait for a long time until a suitable sequence of actions is finally applied by the strategy.

An alternative controller construction is proposed based on the idea of layer decomposition and the availability of a robust solution to the current problem. A distance metric governed by the robust predecessor to calculate costs for player 1 actions of the product game graph is introduced and a memory-less strategy synthesized that minimizes the cost

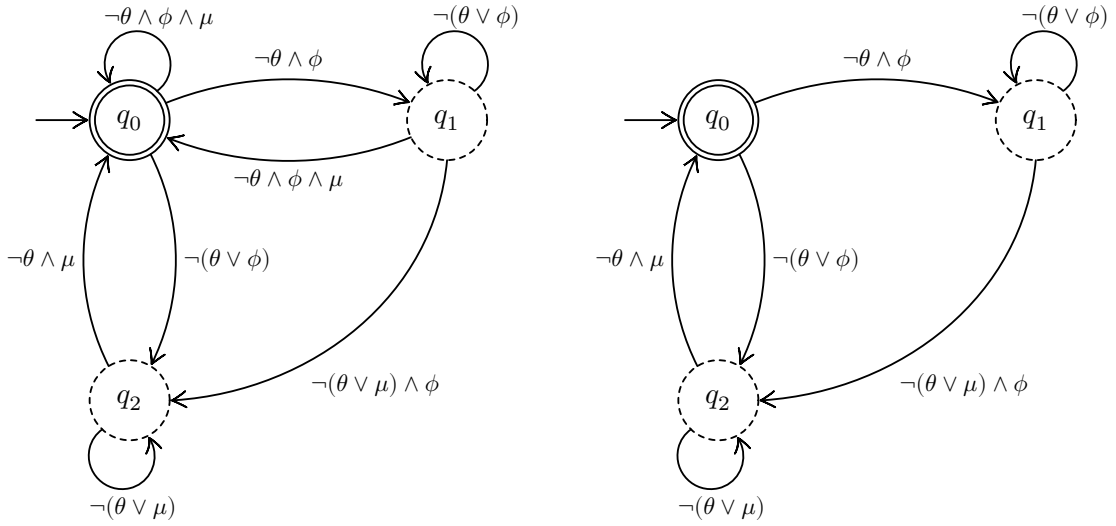


Figure 14: A one-pair Streett automaton corresponding to the GR(1) formula $\mathsf{G}(\neg\theta \wedge \mathsf{F}\phi \wedge \mathsf{F}\mu)$. The automaton transitions have been pruned according to the corridor setup which does not exhibit overlap between the regions defined by ϕ and μ in the right automaton.

over the actions of each state. First a series of layers generated by PreR is computed for the reachability target region. With each new layer spreading outward from the target, a higher cost is assigned to the region covered by the layer. The outer polytopes as well as the region $X \setminus X_{\text{init}}$ is associated with infinite cost. This assignment induces a distance metric on the state space. An action for each state of the game is chosen by valuing the posterior regions associated with all actions based on this distance metric and selecting the one with the minimal volume-weighted cost. Because the layers were generated such that a robust transition from each layer inward to the next is possible, a trace controlled by this strategy will be guided from layer to layer toward smaller cost, reducing the distance to the target until it is reached and the specification satisfied.

A trace controlled with a strategy derived from this layer-based distance minimization scheme is shown in Figure 13, right. The controller is able to guide the trace towards the center and safely through the corridor into the right room. The trace starts in the 7th PreR-layer and reaches the target region in exactly 7 steps.

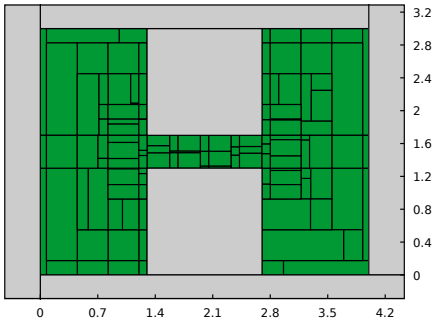
27.3 2-Recurrence and Safety

The corridor system is now verified against a more complex specification. It is required that traces visit both rooms over and over again while staying inside the state space and avoiding the walls of the corridor, expressed by the GR(1) formula

$$\mathsf{G}(\neg\theta \wedge \mathsf{F}\phi \wedge \mathsf{F}\mu) \ .$$

This is an infinite objective. A corresponding ω-automaton for the specification is shown in Figure 14, left. In an accepting run, the automaton state q_0 must be visited infinitely

3:11 elapsed, 122 polygons



Player 1:	339 states, 46768 actions
Player 2:	25139 states, 610511 actions
Refinement:	0:02
Game Graph:	2:09
Analysis:	1:00

69.7% yes, 30.3% no, 0.0% maybe

Table 12: The corridor system solved for a specification requiring both rooms to be visited infinitely often without leaving the state space or entering the "walls" lining the corridor. The partition is produced by layered robust refinement applied to reachability problems corresponding to automaton transitions $q_1 \to q_2$ and $q_2 \to q_0$. See section 27.3 for discussion.

often. State q_1 is a waiting state for the left room and q_2 a waiting state for the right room. Because both rooms ϕ and μ do not overlap in the corridor system, two transitions can be removed from the automaton because their transition conditions are not satisfiable. The pruned automaton is shown in Figure 14, right.

The system is refined with the reachability decomposition approach. The automaton state q_0 has to be visited infinitely often in a run, so it is reasonable to refine such that transitions returning the system into q_0 are rendered possible. In the pruned automaton, these transitions are $q_1 \to q_2$ and $q_2 \to q_0$. The possible satisfying runs enabled by this refinement are runs of the form $(q_0 q_1 q_2)^\omega$ and $(q_0 q_2)^\omega$ as well as all their stutter equivalent runs.

The same layered refinement configuration is used that solved the previous reachability problem. Refinement is applied with respect to both selected transitions, then the system is analysed. The resulting partition and performance statistics are shown in Table 12. Compared to the reachability solution (Table 11), the number of polytopes in the partition has approximately doubled which is to be expected considering that both rooms were refined instead of just the left one. The time required to construct the game graph increased by 45%, while the size of the product game has increased significantly, particularly when comparing the number of player actions. The analysis of the product game takes 15 times as long as that of the reachability problem. Because the game graph has to be constructed only once from the partition and is then multiplied once with every automaton state, these results seem reasonable. The cost of the game graph computation is incurred only once and increases with the size of the partition. The computational demands of the analysis scale with the size of the product game which is governed by both the partition- and automaton sizes. As this example demonstrates, one can expect that the performance of the game solution procedure grows with a growing complexity of the considered specification.

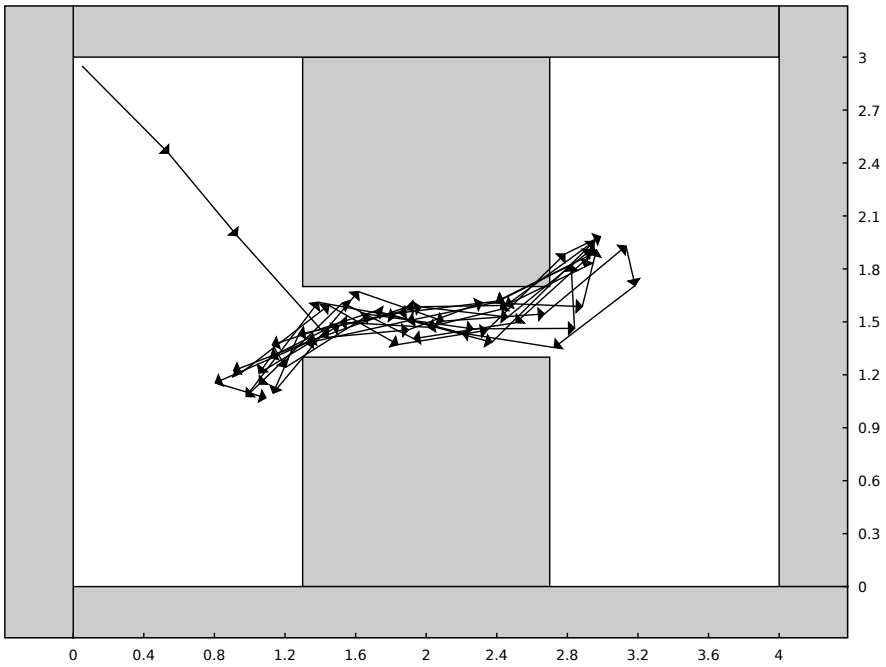


Figure 15: The first 60 steps of a trace sampled using a layer-based distance minimization controller synthesized for a specification that requires to visit both rooms infinitely often while avoiding the walls lining the central corridor. See section 27.3 for discussion.

A controller is again synthesized based on a distance metric induced by PreR-generated layers. When the automaton run corresponding to a trace is in state q_1, the controller selects actions such that the PreR-based distance to the $q_1 \rightarrow q_2$ transition region is minimized, i.e. the controller aims to visit the left room defined by ϕ. When in state q_2, the controller targets the $q_2 \rightarrow q_0$ transition region, i.e. the controller tries to visit the right room defined by μ. In state q_0 the trace is just kept safe, as the next transition immediately leads to either q_1 or q_2. 60 steps of a trace controlled with a strategy of this kind are shown in Figure 15. As desired, the trace does not enter the unsafe regions and visits each room 6 times, navigating back and forth through the corridor.

Conclusion

A brief review of the contents of this work as well as a list of significant contributions is given here. The chapter concludes with a brief outlook onto possible future research directions.

28 Review

This work has been concerned with the computation of the set of initial states of a linear stochastic system such that there exists a control strategy to ensure an objective specified in the extended GR(1) fragment of linear temporal logic with probability 1. The problem was previously posed and solved by Svoreňová et al. (2017), who devised an iterative abstraction-analysis-refinement procedure: A 2½-player game serves as the abstraction model and is constructed based on the dynamics of the LSS. It is combined with the temporal logic specification by translating the latter into a deterministic ω-automaton. Analysing the synchronized product of game and automaton in an adversarial and cooperative setting identifies subsets of the state space for which satisfying strategies exist or cannot exist. If the abstraction is too coarse, state space regions remain undecided and are refined based on information from the system dynamics and analysis. The procedure generates correct solutions at every step but is not guaranteed to terminate.

Building on these ideas laid out by Svoreňová et al. (2017), new refinement procedures were presented in this work. The category of neutral refinement was introduced and filled with safety and self-loop removal refinement procedures. The positive refinement approach was evolved into a more general robust refinement framework and extended to solve multi-step reachability problems. A decomposition of the product game induced by transitions of the objective automaton allows to apply these methods for reachability to problems verified against arbitrary GR(1) specifications. The effectiveness of these refinement procedures was assessed in two case studies. Layered multi-step refinement approaches showed the best and most consistent performance as they reduce jaggedness in the state space partition and thereby stabilize and suppress the state space explosion.

The abstraction-analysis-refinement procedure together with all presented refinement methods was implemented as an interactive application for problems in 1 and 2 dimensions. Its platform is the browser, making it very accessible and therefore suitable for educational and experimentation purposes.

29 Summary of Contributions

- A multi-step refinement framework based on robust dynamics has been developed and successfully tested.
- A decomposition technique induced by the transitions of the objective automaton is described that allows to apply refinement techniques for reachability problems to systems with any specification.
- The class of neutral refinement techniques is introduced. These techniques aim to transfer power in the game graph from player 1 to player 2. They are useful as supportive refinement for other procedures.
- An accessible implementation of the abstraction-analysis-refinement solution scheme in form of an interactive visualization tool is presented.

30 Outlook

In this work, only low-dimensional problems and specifications with small automaton sizes were considered. While it was argued in Section 23 that this restriction does not limit the applicability of obtained results to problems of higher dimensions and with more complex objectives, but it would still be interesting to see how the developed refinement schemes perform in more complex situations.

The limitation to full-dimensional polytopes has been identified as a source of non-termination. It would be interesting to characterize the ϵ-limit behaviour in more detail. A rounding approach could be feasible to resolve the limit behaviour and therefore allow termination for a greater number of problems.

Positive refinement methods that take the probabilistic aspects of the LSS fully into account and do not treat the stochasticity as purely adversarial are still missing.

The relationship between refinement methods and controller synthesis could be further investigated. The robust controller scheme from the corridor case study (Section 27.3) shows that concepts from one domain can be valuable for the other.

The double integrator case study was (partially) solvable through self-loop removal. The alternative approach of Yordanov et al. (2012) to introduce additional actions based on stuttering equivalence however is more sophisticated and can potentially be transferred to the probabilistic setting.

Appendix

The help texts from the interactive application are reproduced here.

View Widgets

These widgets are found below the state space view (see Figure 10).

Widget: Objective Automaton

The one-pair Streett objective automaton. Member states of the acceptance pair (E, F) are depicted with a dashed outline (E) and double outline (F). Click on a state to view the system in that state. Hover over a transition label with the mouse to reveal the associated transition region in the state space view. The symbol * denotes the default transition if no other transition condition matches. Transitions are highlighted blue for the currently selected product game state and red for the selected trace step.

Widget: Control and Random Space

The control- and random space polytopes, U (left) and W (right). ActC regions are highlighted when a player 1 action is selected. Control and random vectors of trace steps are shown when a trace step is selected.

Widget: View Settings

Add or remove information from the state space view:

- Label Polytopes: Display polytope labels X_i.
- Vector Field: Show $\boldsymbol{x}_{t+1} = \boldsymbol{A}\boldsymbol{x}_t$ as a vector field.

State Widgets

These widgets are found in the "State" tab of the sidebar, shown in Figure 17, left. They display information about the game graph.

Widget: Selection

Information about the currently selected state (X_i, q_j) of the product game: Label of the selected state space partition element (choose by clicking in the state space view) and

automaton state (choose by clicking in the automaton view) as well as the successor automaton state based on the satisfied linear predicates. The analysis status for all product game states associated with the state space element is indicated by color.

A dynamics operator is selectable from the dropdown menu which is shown in the state space view.

- Posterior: Post(X_i, U)
- Predecessor: Pre(X, U, X_i)
- Robust Predecessor: PreR(X, U, X_i)
- Attractor: Attr(X, U, X_i)
- Robust Attractor: AttrR(X, U, X_i)

If a player 1 action is selected, the posterior adapts to the concrete action polytope, the backward looking operators do not.

Widget: Actions

Player 1 actions of the game graph denoted as $X_i \to \{X_j\}_{j\in J}$. The colors show the analysis status of the origin and target states, taking into account changes in the automaton state when transitioning. The selection of a player 1 action highlights the associated region $\mathrm{ActC}(X_i, \{X_j\}_{j\in J}) = U_i^J$ in the control space view. The transition is also depicted with arrows between the origin and target polytopes in the state space view.

Player 2 actions are shown when a player 1 action is selected in the form of the support sets $\{X_k\}_{k\in K}$ of the probability distribution of the outgoing transition. Hovering over a player 2 action with the mouse pointer reveals the corresponding $\mathrm{PreP}(X_i, U_i^J, \{X_k\}_{k\in K})$ region in the state space view, where U_i^J is associated with the player 1 action and $K \subseteq J$.

System Widgets

These widgets are found in the "System" tab of the sidebar, shown in Figure 16, right. They provide controls for analysis and refinement of the system as well as the snapshot functionality.

Widget: Analysis

Analyse the system and update the analysis status of all states. While interface remains responsive, only cached data can be displayed until the analysis is finished. Activity is indicated by the loading symbol in the heading. When the analysis is finished, a brief report is written to the log messages (Info tab).

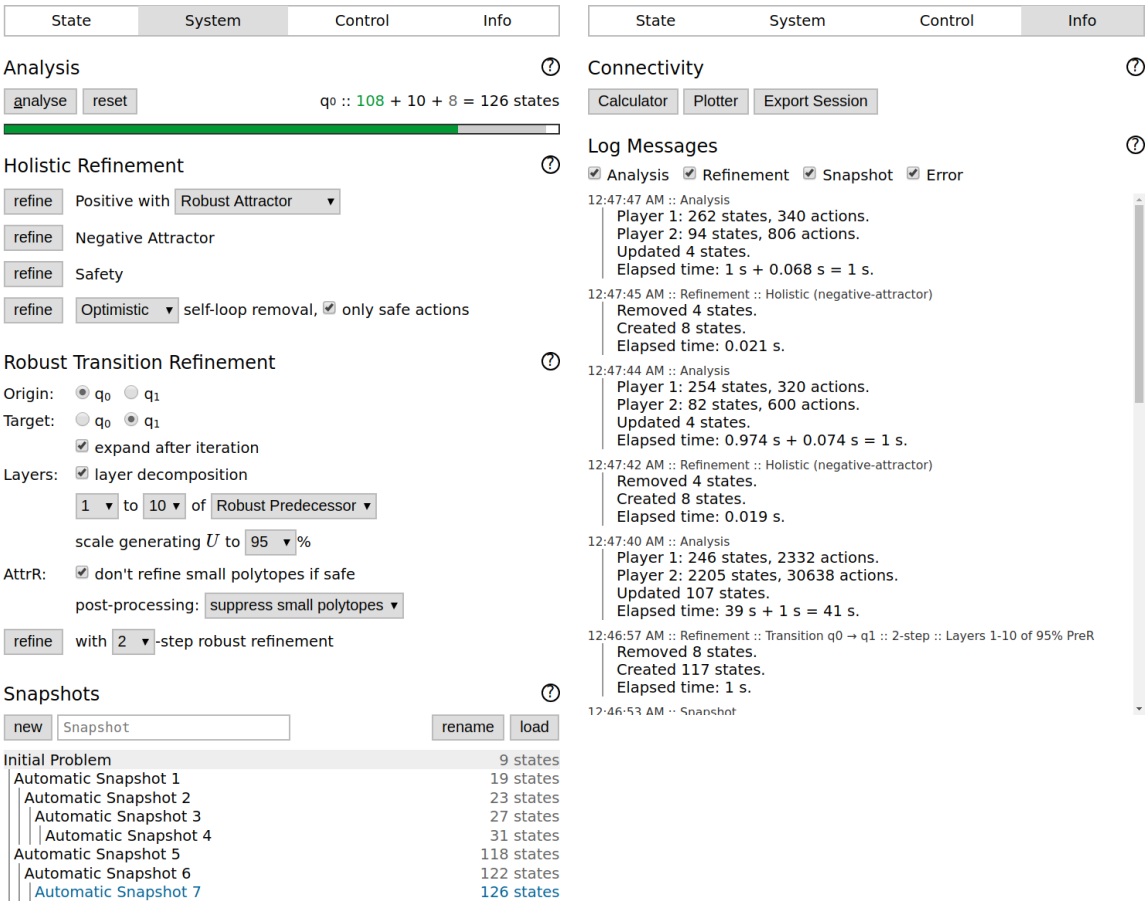


Figure 16: The “System” and “Info” tabs of the sidebar of the interactive implementation (see Figure 10).

Current analysis results can be removed by clicking reset, this allows analysing the system without any product game simplification.

Also displayed are the current number of yes- (green), no- (grey), unreachable- (red) and maybe-states (black). A progress bar shows the state space volume fraction corresponding to each category for the currently selected automaton state. The outer states are excluded from these statistics.

Widget: Holistic Refinement

Refinement based on single-step look-ahead patterns in the entire product game.

- Positive refinement enables robust transitions to the yes-region (robust attractor mode) or refines with respect to the PreR of the yes-region (robust predecessor mode).
- Negative attractor refinement refines with respect to the Attr of the no-region.
- Safety refinement refines robustly with respect to the union of the yes- and maybe-regions.

- Self-loop removal refines with respect to the PreP of player 2 actions that lead to a self-loop. Optimistic removal only refines if player 2 can force a loop for all player 1 actions, while pessimistic removal always refines if a self-loop is found. Unsafe actions can be excluded from the pattern search.

Widget: Robust Transition Refinement

Transition-based refinement in a robust framework. Select origin and target automaton states. A reachability/avoidance problem is extracted based on the corresponding transition. An additional decomposition into layers is possible. Select which layers should be generated as well as the layer-inducing operator. The specified number of refinement iterations is performed on the extracted problem(s) with a positive robust one-step kernel based on the AttrR operator. The following options can be toggled:

- The target region can be automatically expanded between iterations using the progress guarantee of the refinement kernel.
- Small polytopes where $X_i \ominus W = \emptyset$ can be skipped in the refinement if they are safe (i.e. a robust transition exists for the polytope that fulfills the avoidance condition).
- Postprocessing of the AttrR region is available to reduce jaggedness: overapproximation with a convex hull or under-approximation by only retaining the largest polytope of the region or by filtering out small polytopes.
- The control region used for layer generation can be shrunk or expanded to deal with ϵ-limit behaviour.

Widget: Snapshots

Store and restore the state space partition and analysis status of the system in a snapshot. Snapshots are arranged in a tree based on their succession. A snapshot is taken automatically after every analysis and can be taken manually at any time. Progress not saved as a snapshot is lost when loading a previous snapshot.

Control Widgets

These widgets are found in the “Control” tab of the sidebar, shown in Figure 17, left. They provide trace sampling and inspection controls.

Widget: Sample Trace

Generate trace samples in the system. Select the controller used to obtain control vectors from the dropdown menu. Traces are initiated randomly from inside the currently selected

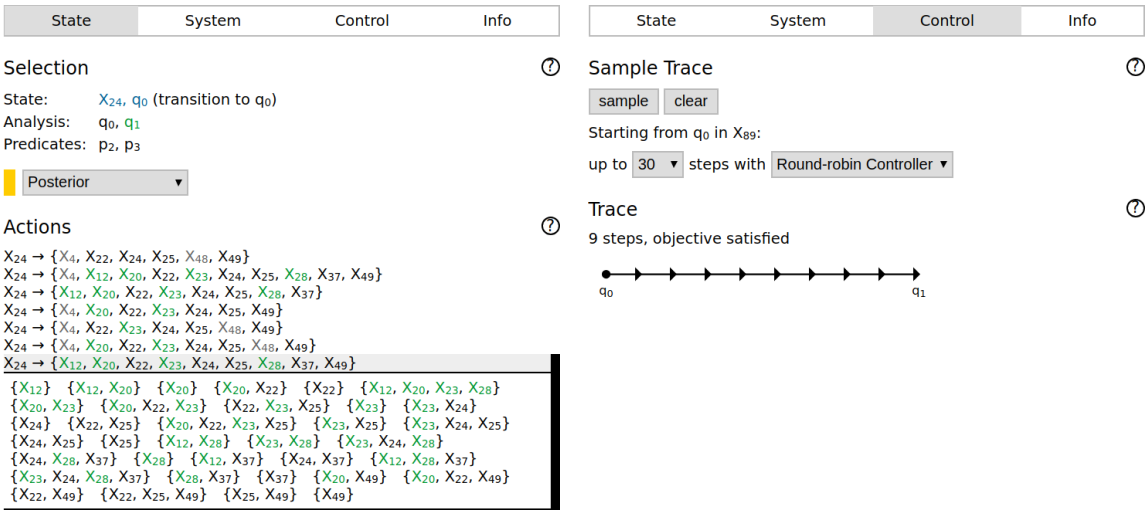


Figure 17: The “State” and “Control” tabs of the sidebar of the interactive implementation (see Figure 10).

product game state or anywhere in the state space for the current automaton state if the polytope selection is empty. Traces are terminated when the outer region is reached, a co-safe objective has been satisfied or the maximum number of steps is exceeded.

Widget: Trace

The individual steps of the current trace. Transitions in the objective automaton in the corresponding play of the product game are shown below the arrows depicting the steps. Hovering over an arrow with the mouse pointer highlights the corresponding step in the state space view as well as the control and random vectors in their corresponding views. The number of steps in the trace sample is given above together with the reason for the termination of the trace.

Info Widgets

These widgets are found in the “Info” tab of the sidebar, shown in Figure 16, right. They are responsible for log-keeping of analysis and refinement reports and connect to additional helper applications developed in the context of this work (not shown).

Widget: Connectivity

Export aspects of the system:

- Calculator: Open the current problem setup in the polytopic calculator application.
- Plotter: Open the current state space partition in the polytopic plotter application.
- Export Session: Download the snapshot tree so that it can be re-imported later from the problem setup screen (make sure to snapshot the current state if desired).

Widget: Log Messages

Type of reports:

- Analysis reports: Information about the size of the product game graph, how many states were updated and timings for the game construction and solution.
- Refinement reports: How many states were refined, how many states were created and timing of the refinement.

Also shown are error messages from the background WebWorker and notifications whenever a snapshot is restored.

References

Abate, A., D'Innocenzo, A. and Di Benedetto, M. D. (2011). Approximate Abstractions of Stochastic Hybrid Systems. *IEEE Transactions on Automatic Control, 56*(11), 2688–2694. doi:10.1109/TAC.2011.2160595.

Aydin Gol, E., Lazar, M. and Belta, C. (2014). Language-Guided Controller Synthesis for Discrete-Time Linear Systems. *IEEE Transactions on Automatic Control, 59*(5), 1163–1176. doi:10.1109/TAC.2013.2295664.

Aydin Gol, E., Lazar, M. and Belta, C. (2015). Temporal logic model predictive control. *Automatica, 56*, 78–85. doi:10.1016/j.automatica.2015.03.029.

Baier, C. and Katoen, J.-P. (2008). *Principles Of Model Checking*. MIT Press.

Balkan, A., Vardi, M. and Tabuada, P. (2018). Mode-Target Games: Reactive Synthesis for Control Applications. *IEEE Transactions on Automatic Control, 63*(1), 196–202. doi:10.1109/TAC.2017.2722960.

Baotić, M. (2009). Polytopic Computations in Constrained Optimal Control. *Automatika, 50*(3-4), 119–134.

Bennion, M. and Habli, I. (2014). A candid industrial evaluation of formal software verification using model checking. In *Companion Proceedings of the 36th International Conference on Software Engineering - ICSE Companion 2014*, pages 175–184.

Chatterjee, K., Chmelík, M. and Daca, P. (2014). CEGAR for Qualitative Analysis of Probabilistic Systems. In Biere, A. and Bloem, R., editors, *Computer Aided Verification. CAV 2014. Lecture Notes in Computer Science, vol 8559*, pages 473–490.

Chatterjee, K. and Henzinger, T. A. (2012). A survey of stochastic ω-regular games. *Journal of Computer and System Sciences, 78*, 394–413. doi:10.1016/j.jcss.2011.05.002.

Clarke, E., Grumberg, O., Jha, S., Lu, Y. and Veith, H. (2000). Counterexample-Guided Abstraction Refinement. In Emerson, E. A. and Sistla, A. P., editors, *Computer Aided Verification. CAV 2000. Lecture Notes in Computer Science*, pages 154–169.

Duret-Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T. and Renault, E. et al. (2016). Spot 2.0 — a framework for LTL and ω-automata manipulation. In *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA'16)*, number 9938 in Lecture Notes in Computer Science, pages 122–129.

Ehlers, R., Lafortune, S., Tripakis, S. and Vardi, M. Y. (2017). Supervisory control and reactive synthesis: a comparative introduction. *Discrete Event Dynamic Systems, 27*(2), 209–260. doi:10.1007/s10626-015-0223-0.

Emerson, E. A. (2008). The Beginning of Model Checking: A Personal Perspective. In Grumberg, O. and Veith, H., editors, *25 Years of Model Checking. Lecture Notes in Computer Science, vol 5000*. Springer.

Filippidis, I., Dathathri, S., Livingston, S. C., Ozay, N. and Murray, R. M. (2016). Control design for hybrid systems with TuLiP: The Temporal Logic Planning toolbox. In *2016 IEEE Conference on Control Applications (CCA)*, pages 1030–1041.

Fukuda, K. (2019). cddlib. https://github.com/cddlib/cddlib. Accessed 2019-05-05.

Gastin, P. and Oddoux, D. (2001). Fast LTL to Büchi automata translation. In Berry, G., Comon, H. and Finkel, A., editors, *Proceedings of the 13th International Conference on Computer Aided Verification (CAV'01)*, number 2102 in Lecture Notes in Computer Science, pages 53-65. Paris, France.

Hahn, E. M., Norman, G., Parker, D., Wachter, B. and Zhang, L. (2011). Game-based abstraction and controller synthesis for probabilistic hybrid systems. In *Proceedings of the 2011 8th International Conference on Quantitative Evaluation of Systems, QEST 2011*, pages 69–78.

Herceg, M., Kvasnica, M., Jones, C. and Morari, M. (2013). Multi-Parametric Toolbox 3.0. In *Proc. of the European Control Conference*, pages 502–510. Zürich, Switzerland.

Kesten, Y., Piterman, N. and Pnueli, A. (2005). Bridging the gap between fair simulation and trace inclusion. *Information and Computation, 200*(1), 35–61. doi:10.1016/j.ic.2005.01.006.

Kloetzer, M. and Belta, C. (2008). A Fully Automated Framework for Control of Linear Systems from Temporal Logic Specifications. *IEEE Transactions on Automatic Control, 53*(1), 287–297. doi:10.1109/TAC.2007.914952.

Křetínský, J., Meggendorfer, T., Sickert, S. and Ziegler, C. (2018). Rabinizer 4: From LTL to your favourite deterministic automaton. In Chockler, H. and Weissenbacher, G., editors, *Computer Aided Verification*, pages 567–577.

Kupferman, O. and Vardi, M. Y. (2001). Model checking of safety properties. *Formal Methods in System Design, 19*(3), 291–314. doi:10.1023/A:1011254632723.

Lahijanian, M., Andersson, S. B. and Belta, C. (2015). Formal Verification and Synthesis for Discrete-Time Stochastic Systems. *IEEE Transactions on Automatic Control, 60*(8), 2031–2045. doi:10.1109/TAC.2015.2398883.

Lin, H. and Antsaklis, P. J. (2014). Hybrid dynamical systems: An introduction to control and verification. *Found. Trends Syst. Control, 1*(1), 1–172. doi:10.1561/2600000001.

Piterman, N., Pnueli, A. and Sa'ar, Y. (2005). Synthesis of Reactive(1) Designs. In Emerson, E. and Namjoshi, K., editors, *Verification, Model Checking, and Abstract Interpretation. VMCAI 2006. Lecture Notes in Computer Science*, pages 364–380.

Prasad Sistla, A. (1994). Safety, liveness and fairness in temporal logic. *Formal Aspects of Computing, 6*(5), 495–511. doi:10.1007/BF01211865.

Rakovic, S., Grieder, P., Kvasnica, M., Mayne, D. and Morari, M. (2004). Computation of invariant sets for piecewise affine discrete time systems subject to bounded disturbances. In *43rd IEEE Conference on Decision and Control (CDC)*, pages 1418–1423 Vol.2.

Svoreňová, M., Černá, I. and Belta, C. (2013). Optimal control of MDPs with temporal logic constraints. In *Proceedings of the IEEE Conference on Decision and Control*, pages 3938–3943.

Svoreňová, M., Křetínský, J., Chmelík, M., Chatterjee, K., Černá, I. and Belta, C. (2017). Temporal logic control for stochastic linear systems using abstraction refinement of probabilistic games. *Nonlinear Analysis: Hybrid Systems, 23*(15), 230–253. doi:10.1016/j.nahs.2016.04.006.

Van Der Schaft, A. J. and Schumacher, J. M. (2000). *An introduction to hybrid dynamical systems*. Springer.

Yordanov, B. and Belta, C. (2009). Temporal logic control of discrete-time piecewise affine systems. In *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pages 3182–3187.

Yordanov, B. and Belta, C. (2010). Formal Analysis of Discrete-Time Piecewise Affine Systems. *IEEE Transactions on Automatic Control, 55*(12), 2834–2840. doi:10.1109/TAC.2010.2072530.

Yordanov, B., Tumova, J., Cerna, I., Barnat, J. and Belta, C. (2012). Temporal Logic Control of Discrete-Time Piecewise Affine Systems. *IEEE Transactions on Automatic Control, 57*(6), 1491–1504. doi:10.1109/TAC.2011.2178328.