Learning of Safety Controllers using Horn Constraints

First Draft

October 6, 2018

Abstract

Infinite-duration two-person games are a popular formalism for synthesizing reactive controllers. In this work, we consider safety games (which arise from safety specifications) and show how safety controllers can be computed using machine learning. At the heart of our approach lies a novel decision tree learning algorithm that learns from Horn constraints and has recently been developed for the verification of concurrent programs. We implement a teacher and show how we can use the Horn learner to synthesize a controller. Furthermore, we show that there is a bounded algorithm that guarantees the learning process to terminate in polynomial time.

1 Introduction

We are looking to synthesize reactive controllers subject to safety specifications. We model the interaction between system and environment as a two-player safety game over possibly infinite graphs. Thus, the usual approach of fixpoint computation doesn't work on this kind of problems. Instead, we use a a teacher learning setting to learn a controller. The teacher has knowledge about the game and the learner proposes hypothesis about a winning set in the game.

Based on the ICE learning framework the teacher responds with counterexamples. Those counterexamples can have four different types: positive, negative, existential implication and universal implication counterexamples. We use a decision tree learner for Horn Samples and need to encoded our counterexamples accordingly to fit those requirements. The learning takes place iteratively. In each iteration the learner proposes a hypothesis to the teacher and the teacher responds with either a counterexample or none. The latter meaning that the hypothesis is indeed a winning set. Otherwise the learner incorporates the counterexample in the next iteration of synthesizing a hypothesis.

One result of this paper is that we can transform a Game Sample that consists of such counterexamples into a Horn Sample that consists of Horn constraints. This allows us to use the already existing Horn learner[3]. The teacher itself is based on the symbolic computation of counterexamples, used in "An Automaton Learning Approach to Solving Safety Games over Infinite Graphs" [4]. But instead of using automata, we use SMT-solver to compute the counterexamples. This form of computation allows us to avoid exponential

blow up by allowing us to not iterate over all possible solution but instead have an symbolic representation of our game.

In detail, we learn a winning set for the controlled system. With the help of a winning set we can determine a strategy by staying inside the winning set. Once in the winning set the controlled system can force to stay inside this set of vertices.

The main results of this paper are how to use a decision tree learner for Horn Samples to compute a winning set. A teacher using an SMT-solver to compute the counterexamples needed for the learner and the transformation of the Game Sample that consists of those counterexamples to a Horn Sample. Finally, we show that using our algorithm with the bounded Horn learner is guaranteed to converge.

2 Overview

We give an overview of all definitions that will be used during this paper. These include the definition of safety games and horn constraints. One of the first things we have to understand is the notion of data points.

We use data points over a domain D to describe points in our graph. Our learner can use Horn constraints to build a tree $\mathcal{T}$, which evaluates data points to a boolean value ($\mathcal{T} : D \to \mathbb{B}$). The learning algorithm constructs a tree using a set of base predicates which evaluate to $true$ or $false$ on data points. We can see, that a boolean function $f : D \to \mathbb{B}$ induces a valuation $v : D \to true, false$, where $x \mapsto_f 1 \iff x \mapsto_v true$.

2.1 Safety Games

First of we start with some definitions needed to understand safety games.

We say that a safety game is a infinite duration game over a graph with a countable set of vertices. Safety games are played on an arena $\mathfrak{A} = (V_0, V_1, E)$, where V_0 and V_1 are disjoint and countable sets of vertices and E defines a directed edge relation $E \subseteq V \times V$. We write successor of a set of vertices as $E(X) = \{y \mid \exists x \in X : (x, y) \in E\}$. The initial vertices of a safety game are defined in the following way. $\mathfrak{G} = (\mathfrak{A}, F, I)$, $\mathfrak{A} = (V_0, V_1, E)$ is an arena, $F \subseteq V$ a set of *safe vertices* and $I \subseteq F$ a set of *initial vertices*. Next, we explain how safety games are played. We have two players, Player 0 and Player 1 and each of them owns a set of vertices. Player 0 owns the vertices that are in V_0 and Player 1 the vertices in V_1. The safety game starts when a initial token is placed on one initial vertex $v_0 \in I$. The player owning the vertex can move the token to one of its successors. Since we are talking about infinite duration games, the process is repeated ad infinitum. This yields a infinite sequence of vertices $v_0 v_1 ...$ that we call a *play* iff $v_0 \in I$ and $(v_i, v_{i+1}) \in E$ for all $i \in \mathbb{N}$. A play is *winning* for Player 0 if $v_i \in F$ for all $i \in \mathbb{N}$. Otherwise the play is *winning* for Player 1.

A strategy for Player σ, $\sigma \in \{0, 1\}$, is a mapping $f_\sigma : V * V_\sigma \to V$. A winning strategy yields a winning play for any play that is created using the strategy. We can get a winning strategy if you look at the *winning set* of a safety game which is defined as follows.

Winning set For a safety game $\mathfrak{G} = (\mathfrak{A}, F, i)$ over the arena $\mathfrak{A} = (V_0, V_1, E)$, a winning set is a set $W \subseteq V$ satisfying

1. $I \subseteq W$
2. $W \subseteq F$
3. $E(\{v\}) \cap W \neq \emptyset$ for all $v \in W \cap V_0$
4. $E(\{v\}) \subseteq W$ for all $v \in W \cap V_0$

A winning strategy for Player 0 would be to move inside the winning set. This can be proven with a induction over the length of plays.

2.2 Teacher and learner interaction

In this section we explain how the teacher and the learner interact with each other. The goal is to compute winnings sets. Winnings sets have the above defined properties. We will follow the ICE-learning approach by learning with counterexamples, meaning if one of the properties is unsatisfied we can generate one counterexample.

We can give positive counterexamples if property $I \subseteq W$ is unsatisfied and a negative counterexample if $W \subseteq F$ is unsatisfied. For the third property we can give existential counterexamples $(x \to (x_1 \vee ... \vee x_k))$ and for the fourth we can give universal counterexamples $(x \to (x_1 \wedge ... \wedge x_k))$. We will show later that it is possible to transform those counterexamples into horn constraints.

The teacher has knowledge about the safety game and can verify winning sets that the learner proposes. The teacher gives the learner counterexamples if the proposed winning set is wrong. The learner makes a conjecture about the winning set using the counterexamples given by the teacher. Then he gives the teacher his constructed winning set. The interaction between teacher and learner takes place in rounds. In each round the learner makes a new conjecture and the teacher either accepts it or gives the learner a counterexample for his conjecture.

The counterexamples of the teacher are not horn clauses. Our goal is it to use a Horn learner, thus we present a way how to transform the counterexamples given by teacher into Horn clauses, usable by the Horn learner.

First, we need to formalize definitions about the counterexamples. Starting with a *Game Sample* that is produced by the teacher, the goal is to have a *Horn Sample* which can be used by the learner.

Definition 2.1. *A* Game Sample *is a set $S = (X, C)$, where X is a set of positive and negative data points and C is a set of formulas over X. Those formulas have the following structure:*

1. $true \to x$
2. $x \to false$
3. $x \to x_1 \vee ... \vee x_k$
4. $x \to x_1 \wedge ... \wedge x_k$

Definition 2.2. *A* Horn Sample *is a set $S = (X, C)$, where X is a set of boolean variables and C is a set consisting of Horn constraints over X. Those formulas have the following structure:*

1. $true \rightarrow x$
2. $x \rightarrow false$
3. $x_1 \wedge ... \wedge x_k \rightarrow x$

Furthermore, we need to define what it means for a sample to be valid. We want a sample to be valid if it doesn't contradict itself, so we take an intuitive definition for validity.

Definition 2.3. *A Horn Sample (X,C) is* valid $\iff$ *there is a valuation v such that $v \vDash C$.*

Definition 2.4. *A Game Sample (X,C) is* valid $\iff$ *there is a valuation v such that $v \vDash C$.*

2.3 Horn constraints and how to use them with data points

In this section we explain how the Horn Sample and the Game Sample are connected. In the Game Sample the Horn constraints are over a set of data points. But in the Horn Sample we have Horn constraints over a set of boolean variables.

A Horn clause (or a Horn constraint) over X is a disjunction of literals over X with at most one positive literal. These are not necessarily boolean formulas since the formulas can go over data points. Let $S := (X, C)$ be a Game sample, where $C = \{c_1, .., c_m\}$ is a set of formulas that are constraints, but instead of variables we have data points, and $D(C) := \{D(c) | c \in C\}$ is a set of data points that appear in the formulas. But we want to evaluate constraints over boolean variables. To do this we need a Sample $S' = (X', C')$, but this time we have Horn constraints over boolean variables. To get this sample we need define a bijective function $v : D(C) \rightarrow X'; d \mapsto x_d$ that maps our data points to variables that we use in our Horn constraints. Additionally, C' has the same Horn constraints like C, but every data point is replaced with its variable.

We can now define a valuation for X', $v_{\mathcal{T}} : X' \rightarrow \mathbb{B}; x_d \mapsto t(v^{-1}(x_d))$. Where t is the evaluation of the data point of x_d on the decision tree.

We say a Horn sample $S = (X, C)$ over boolean variables is consistent with a tree $\mathcal{T}$ if and only if $v_{\mathcal{T}} \vDash \bigwedge_{c \in C} c$.

In summary it can be said, therefore, that we can get a valuation over boolean variables over a Game Sample, transforming it into a Sample with the same constraints over boolean variables. We will need this observation to show how we can transform such a Game Sample into a Horn Sample.

3 Transformation from Game Sample to Horn Sample

We will begin by applying our observation from the previous section and look at a Sample which is based on a Game Sample, but uses boolean variables instead.

Let C be a set of formulas over a set of variables X. These formulas are of the following forms:

1. $true \to x$
2. $x \to false$
3. $x \to (x_1 \vee ... \vee x_k)$
4. $x \to (x_1 \wedge ... \wedge x_k)$

Define for a valuation v over X a new valuation $v' : X \to \mathbb{B}; x \to 1 - v(x)$. Define a set of horn constraints C'. Add a new formula to C' for every formula in C:

1. $\forall(true \to x \in C)$ add $(x \to false)$ to C'
2. $\forall(x \to false \in C)$ add $(true \to x)$ to C'
3. $\forall(x \to (x_1 \vee ... \vee x_k) \in C)$ add $((x_1 \wedge ... \wedge x_k) \to x)$ to C'
4. $\forall(x \to (x_1 \wedge ... \wedge x_k) \in C)$ add a set of formulas $\{x_i \to x | i \in \{1, ..., k\}\}$ to C'.

We can see that C' is a set of Horn constraints.

Lemma 3.1. *Let v be a valuation for a set of formulas C like in the construction above. Let v' be the valuation and C' a set of horn constraints we get by applying the construction. Then:*

$$v \vDash C \iff v' \vDash C'$$

Proof.
Let v be a valuation for C. If $v \vDash C$ then $v \vDash \bigwedge_{c \in C} c$ and $\forall c \in C : v \vDash c$. Therefore, it is enough to perform a case analysis for each formula $c \in C$. We do this by applying the described construction above to v and C:

1. We apply the construction to formulas of the form $c := true \to x$ and get $c' := x \to false$:

$$v \vDash c \iff v \vDash x \iff v' \nvDash x \iff v' \vDash c'$$

2. We apply the construction to formulas of the form $c := x \to false$ and get $c' := true \to x$:

$$v \vDash c \iff v \nvDash x \iff v' \vDash x \iff v' \vDash c'$$

3. We apply the construction to formulas of the form $c := (x \to (x_1 \vee ... \vee x_k))$ and get $c' := (x_1 \wedge ... \wedge x_k) \to x)$:

$$v \vDash c \iff (\exists i \in \{1..k\} : v \vDash x_i) \vee (v \nvDash x)$$
$$\iff (v' \vDash x) \vee ((\exists i \in \{1..k\} : v' \nvDash x_i)) \iff v' \vDash c'$$

4. We apply the construction to formulas of the form $c := x \to (x_1 \wedge ... \wedge x_k)$ and get $c' := \{x_i \to x | i \in \{1, ..., k\}\}$:

$$\begin{aligned} v \vDash c &\iff (\forall i \in \{1..k\} : v \vDash x_i) \vee (v \nvDash x) \\ &\iff (v' \vDash x) \vee ((\forall i \in \{1..k\} : v' \nvDash x_i)) \\ &\iff (\forall i \in \{1..k\} : v' \vDash x_i \to x) \iff v' \vDash c' \end{aligned}$$

Summarizing the case analysis we can see that if we take any formula in $c \in C$ and transform it into a formula $c' \in C'$ we get that $v \vDash c \iff v' \vDash c'$ and therefore $v \vDash C \iff v' \vDash C'$ for any valuation v.

□

Corollary 3.2. *A formula ψ is consistent with a Game Sample if and only if it is consistent with the corresponding Horn Sample.*

Proof. Let ψ be a formula that is consistent with the Game Sample S_{Game}. Now we have to perform a case analysis again for each formula $c \in C_{Game}$.

1. $$\begin{aligned} &x \to x_1 \vee \cdots \vee x_n \textit{ is consistent with } \psi \\ &\iff (\exists i \in \{1, \ldots, n\} : x_i \vDash \psi) \vee (x \nvDash \psi) \\ &\iff (x' \vDash \psi) \vee ((\exists i \in \{1..n\} : x'_i \nvDash \psi)) \\ &\iff x'_1 \wedge \cdots \wedge x'_n \to x' \textit{ is consistent with } \psi \end{aligned}$$

2. The rest of the cases are similarly proven.

□

4 Bounded Decision Tree Learner for Horn Samples

In this section, we show that we can build a decision tree learner that is guaranteed to terminate. We already know that the Horn Learner terminates on a sample[3], but what is left to show is that the teacher learner interaction eventually terminates. This is not obvious since the learner is learning over a set of formulas that involves numerical attributes, making it infinitely large.

We want to bound the maximum threshold that can occur in the inequalities the learner uses in the hypothesis. Thus, the amount of non equivalent boolean formulas that respect the restriction is finite. We iteratively bound the maximum threshold that occur in the inequalities and only increase the maximum threshold if there is no tree that respects the restriction and is consistent with the sample. The learner never proposes the same tree twice since the teacher

will respond with an appropriate counterexample if the tree is not a winning set. Furthermore, we won't increase the maximum threshold if there is a tree that respects the threshold and is a winning set. Thus, if there is a tree that is a winning set within a maximum threshold, the learner will find this tree in an finite amount of time.

To proof our claim, we want to use theorem 4.1. For this, we will define when a set of points is separable.

Theorem 4.1. *If the input set of points X is separable and the input Horn Constraints are satisfiable, then the Learner always terminates with a decision tree consistent with the Horn sample (X,C).[3]*

Definition 4.2. *Given an Horn sample $S = (X, C)$. Define the equivalence class $\equiv_m$ m on X:*

$$s \equiv_m s' \iff \text{there is no predicate of the form } a_i \leq c \text{ with } |c| \leq m \text{ that separates } s \text{ and } s'.$$

Definition 4.3. *We can augment a sample $S = (X, C)$ to obtain a* m-augmented Horn Sample of S*, denoted by $S \oplus m$. To obtain $S \oplus m$, we first construct the set $E = \{(s, s') \mid s \equiv_m s'\}$ and add it to a our Horn Constraints, $(X, C \cup E)$.*

Next, we want to show that if we have a Horn Sample that respect a maximum threshold m, our Horn Learner returns with a tree that is consistent with the sample. Our goal is to show that if we have a Game Sample, then our Horn Learner returns with a tree that is consistent with the Game Sample. But first we have to take detour to Horn Samples. We will use later our transformation to show that the same can be done for the Game Sample.

Theorem 4.4. *Let S be an Horn Sample. Then the following holds:*

1. *If $S \oplus m$ is not valid, then there is* no Boolean formula with absolute maximum threshold m *that is consistent with S. In the overall learning loop, we would increment m in this case and restart learning from S.*
2. *If $S \oplus m$ is valid, then calling our Decision Tree Learner for Horn Samples on $S \oplus m$ while restricting it to predicates that use thresholds with absolute values at most m is* guaranteed to terminate and return a tree that is consistent with S.

Proof.

1. Assume $S \oplus m$ is not valid. This means the Horn Solver returns with an unsat. Now for the sake of contradiction assume that there is a Boolean formula f with absolute maximum threshold m that is consistent with S. Thus, f satisfies all the new horn constraints added and consequently $S \oplus m$. Thus calling our Horn Solver on $S \oplus m$ should have returned with a boolean function f' which assigns the values in X according to f which is a contradiction to our Horn Solver returning unsat.

2. Assume now that $S \oplus m$ is valid. We need to show that the input set of points is separable. For this assume that our bounded Decision Tree Learner for Horn Samples processes a node with the sample of $S \oplus m$. Moreover, assume that we can't find a split with threshold $|c| \leq m$. We know that the data points we want to split can't be pure. Otherwise, we have a split which assigns them all to one side. Thus, there are three different cases that can happen, two of those are analogous.

   (a) We have at least one positively classified data point p and at least one negatively classified data point n. Since we can't separate those two points, we know that they are in the same m-equivalence class. Thus, (p, n) is a horn constraint in $S \oplus m$. This means our Horn Solver should have returned with unsat, since (p, n) is an unsatisfiable horn constraint.

   (b) We have no negatively classified data points. Thus, we only have positively classified data points and unclassified data points. Since those points are inseparable, they are all in the same m-equivalence class. This means that they all have to have the same classification which is positive, making a split possible. This is a contradiction to our assumption that we can't separate those data points.

   (c) We have no positively classified data points. This is analogous to case (b).

   Furthermore, if $S \oplus m$ is valid, then there is a valuation that satisfies C.

   Now, we have that the input set of points is separable and that $S \oplus m$ is valid and can use Theorem 4.1 to get that the Learner always terminates with a decision tree consistent with the Horn sample (X, C).

□

Now, that we have shown that our Horn learner works with a Horn Sample, we show that it also works if we start with a Game Sample.

Theorem 4.5. *Let S_{Game} be a Game Sample. Then the following holds:*

1. *If $S \oplus m$ is not valid, then there is* no Boolean formula with absolute maximum threshold m *that is consistent with S. In the overall learning loop, we would increment m in this case and restart learning from S.*

2. *If $S \oplus m$ is valid, then calling our Decision Tree Learner for Horn Samples on $S \oplus m$ while restricting it to predicates that use thresholds with absolute values at most m is* guaranteed to terminate and return a tree that is consistent with S.

Proof.

1. $S_{Game} \oplus m$ not valid $\Rightarrow$ there is no valuation v such that $v \vDash C \overset{Lemma\ 3.1}{\Rightarrow}$ there is no valuation v' for the Horn Sample $S_{Horn} \oplus m$. $\Rightarrow S_{Horn} \oplus m$ is not valid $\overset{Theorem\ 4.4}{\Rightarrow}$ there is no Boolean formula with maximum threshold m that is consistent with S_{Horn}

Assume there exists a Boolean formula f' with maximum threshold m such that f' is consistent with S_{Game}. With Corollary 3.2 we know that f' is also consistent with S_{Horn}. Thus, we found a Boolean formula with maximum threshold m that is consistent with S_{Horn} which is a contradiction to $S_{Game} \oplus m$ being not valid.

2. $S_{Game} \oplus m$ is valid $\overset{Lemma\ 3.1}{\Rightarrow}$ $S_{Horn} \oplus m$ is valid $\overset{Theorem\ 4.4}{\Rightarrow}$ our Learner for Horn Samples returns with a tree that is consistent with S_{Horn}, this tree encodes a Boolean formula that is consistent with S_{Horn} $\overset{Corollary\ 3.2}{\Rightarrow}$ this tree is also consistent with S_{Game}

□

5 Experiments

In order to compare how effective the Horn learner is, we implemented a teacher using Microsofts Z3 [2] constraint solver and used the transformation mentioned in section 3 such that the Horn learner can learn using Horn constraints. The samples our teacher produces are Game samples. Furthermore, we will use the proposed Horn learner in "Horn-ICE Learning for Synthesizing Invariants and Contracts" [3]. We will compare our results with the Sat learner and the RPNI learner mentioned in "An Automaton Learning Approach to Solving Safety Games over Infinite Graphs" [4].

Again we will restrict our experiment in a way that each vertex has a finite number of outgoing edges. In our case this number also needs to be bounded. We need the limitation since our teacher is required to return a counterexample which lists all outgoing edges if the counterexample is an existential or universal implication.

We conducted the experiments on a intel core i3-4005U with Ubuntu 18.04 as operating system. The memory limit is at 3.8 GiB.

We considered following examples, for an detailed explanation we refer to the appendix:

1. *Limited Diagonal game:* A variation of the diagonal game, used in "An Automaton Learning Approach to Solving Safety Games over Infinite Graphs" [4]. Player 0 can only control the robot's vertical movement and Player 1 the horizontal.

2. *Box game:* A variation of the diagonal game. Player 0 wins if the robot stays within a horizontal stripe of width tree.

3. *Limited Box game:* A variation of the box game, used in "An Automaton Learning Approach to Solving Safety Games over Infinite Graphs" [4]. Player 0 can only control the robot's vertical movement and Player 1 the horizontal.

4. *Solitary box game:* Box game with only Player 0 controlling the robot.

5. *Evasion game:* Two robots are moving in an infinite, discrete two-dimensional grid world. The robots take turns moving at most one cell in any direction. Both players control one robot. Player 0's objective is to avoid getting caught by Player 1's robot.

6. *Follow game:* A version of the evasion game where Player 0's objective is to keep the robot within a distance of two cells. (Manhattan distance)

7. *Program repair game:* The program-repair game described by Beyene et al. [1]

8. *Square game:* A variation of the box game, where the space Player 0 is allowed to move is limited to a square.

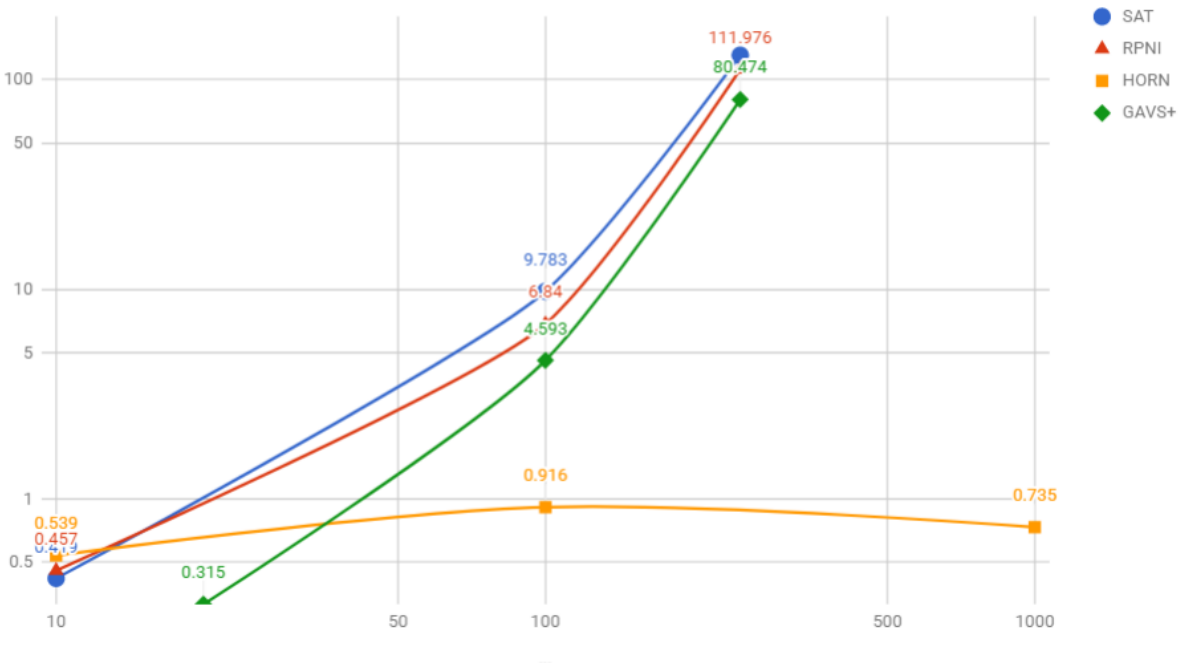
	Horn Learner							Sat Learner							RPNI Learner					
Game	T	Iter.	Size	*Pos*	*Neg*	*Ex*	*Uni*	T	Iter.	Size	*Pos*	*Neg*	*Ex*	*Uni*	T	Iter.	Size	*Pos*	*Neg*	*Ex*	*Uni*
Diagonal	2.929	24	6	11	6	4	2	2.891	64	4	1	56	3	3	1.789	64	4	1	49	8	5
Limited Box	0.641	9	4	2	3	0	3	2.024	45	5	1	41	0	2	0.906	16	6	1	11	0	3
Box	0.202	4	3	1	2	0	0	1.212	36	4	1	33	0	1	0.599	15	5	1	10	1	2
Limited Solitary Box	0.697	4	3	1	2	0	0	5.812	76	6	2	70	3	0	0.423	16	6	1	13	1	0
Evasion	5.206	16	4	1	4	7	3	751.919	237	7	2	219	7	8	2.606	82	11	1	62	9	9
Follow	44.097	181	14	1	129	32	18	307.693	311	7	2	287	9	12	18.922	352	16	1	273	33	44
Program-repair	1.129	14	11	1	2	10	0	3.249	71	3	2	64	4	0	0.239	7	3	1	2	3	0
Quadrat 3x3	6.213	61	13	1	18	19	22			timeout					1.267	35	12	1	30	0	3
Quadrat 5x5	4.195	41	16	1	13	16	10			timeout					1.398	39	14	1	31	1	5

Figure 1: Summary of results on games over infinite arenas

	Horn Learner							Sat Learner							RPNI Learner					
Game	T	Iter.	Size	*Pos*	*Neg*	*Ex*	*Uni*	T	Iter.	Size	*Pos*	*Neg*	*Ex*	*Uni*	T	Iter.	Size	*Pos*	*Neg*	*Ex*	*Uni*
Diagonal	2.929	24	6	11	6	4	2	2.891	64	4	1	56	3	3	1.789	64	4	1	49	8	5
Limited Box	0.641	9	4	2	3	0	3	2.024	45	5	1	41	0	2	0.906	16	6	1	11	0	3
Box Limited	0.202	4	3	1	2	0	0	1.212	36	4	1	33	0	1	0.599	15	5	1	10	1	2
Solitary Box	0.697	4	3	1	2	0	0	5.812	76	6	2	70	3	0	0.423	16	6	1	13	1	0
Evasion	5.206	16	4	1	4	7	3	751.919	237	7	2	219	7	8	2.606	82	11	1	62	9	9
Follow	44.097	181	14	1	129	32	18	307.693	311	7	2	287	9	12	18.922	352	16	1	273	33	44
Program-repair	1.129	14	11	1	2	10	0	3.249	71	3	2	64	4	0	0.239	7	3	1	2	3	0
Quadrat 3x3	6.213	61	13	1	18	19	22			timeout					1.267	35	12	1	30	0	3
Quadrat 5x5	4.195	41	16	1	13	16	10			timeout					1.398	39	14	1	31	1	5

Scalability. To compare the scalability of our tool we model a problem on an one dimensional grid-world which consists of m cells. Only the right most cell is considered unsafe, while every other cell is considered safe. The system and the environment take turns controlling a robot. The system can move the robot one cell to the left or stay, the environment can move the robot one to the right or stay. But the system is more restricted, because if the robot reaches one cell that is right of cell $\lfloor \frac{m}{2} \rfloor$, the system is not able to move left anymore. This makes the winning set a subset of the cells left of cell $\lfloor \frac{m}{2} \rfloor$.

Figure 2: Results of Scalability benchmark



6 Conclusion

References

[1] Tewodros Beyene, Swarat Chaudhuri, Corneliu Popeea, and Andrey Rybalchenko. A constraint-based approach to solving games on infinite graphs. In *ACM SIGPLAN Notices*, volume 49, pages 221–233. ACM, 2014.

[2] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[3] Deepak D'Souza, P Ezudheen, Pranav Garg, P Madhusudan, and Daniel Neider. Horn-ice learning for synthesizing invariants and contracts. *arXiv preprint arXiv:1712.09418*, 2017.

[4] Daniel Neider and Ufuk Topcu. An automaton learning approach to solving safety games over infinite graphs. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 204–221. Springer, 2016.

A Appendix

A.1 Game Examples

Figure 3: In this example we have a robot moving on an infinite, discrete one-dimensional grid. Two players can control the robot, Player 0 and Player 1. The goal of Player 0 is to let the robot stay in the safe zone, the goal of Player 1 is to move the robot out of it. Both player take turns moving the robot, moving it at most one field in one direction. .

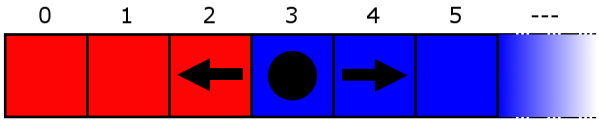


Figure 4: Here we have a water tank and two players. This example works similiar to figure above, but this time the safe zone is limited by two sides since a water tank has only a finite capacity and we don't want the tank to run out of water.

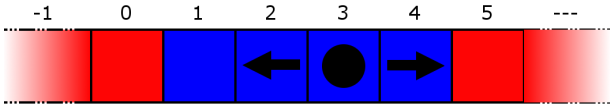


Figure 5: Consider a robot on an infinite two-dimensional grid with two players playing against each other. Both player can move the robot in an arbitrary direction by at most one field. Player 0 tries to keep the robot in a safe square, while Player 1 tries to move the robot out of the square.

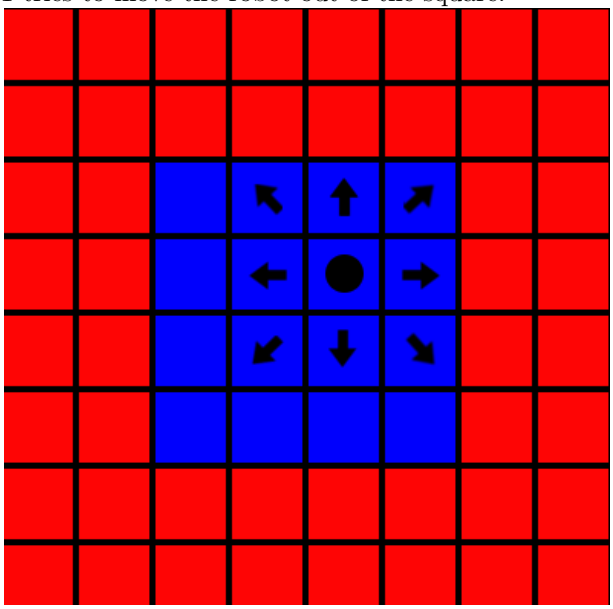


Figure 6: This example works like the water tank example, played with multiple tanks. Each player can control the tank in its turn and can decide to either fill the tank or empty the tank by one unit.

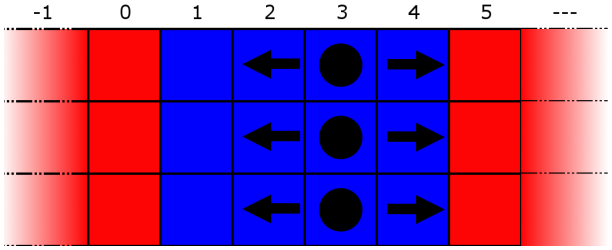


Figure 7: In this example we have an infinite, discrete two-dimensional grid that is limited by two straight lines. Both player can move the robot in an arbitrary direction. Player 0 wins if the robot stays within the area limited by the straight lines. Player 1 wins if the robot moves out of it.

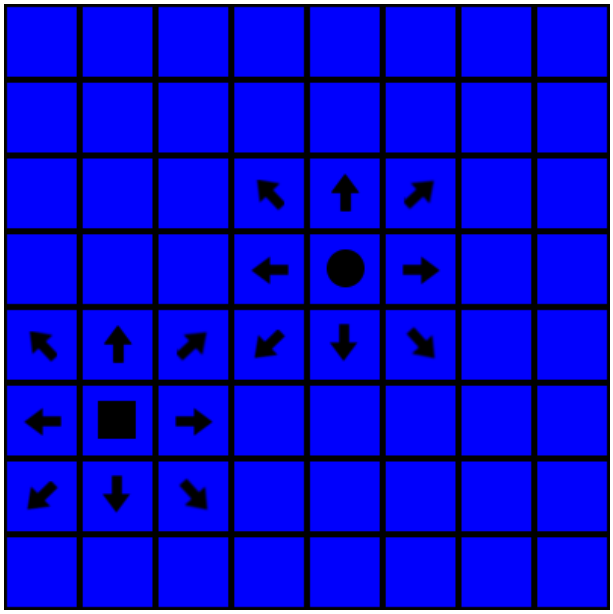


Figure 8: We have two robots moving in an infinite, discrete two-dimensional grid. Player 0's objective is to avoid being caught by Player 1.