What's to Come is Still Unsure*

Synthesizing Controllers Resilient to Delayed Interaction

Mingshuai Chen[1(✉)], Martin Fränzle[2], Yangjia Li[3,1], Peter N. Mosaad[2], and Naijun Zhan[1]

[1] State Key Lab. of Computer Science, Institute of Software, CAS, Beijing, China &
University of Chinese Academy of Sciences, Beijing, China
{chenms,yangjia,znj}@ios.ac.cn
[2] Dpt. of Computing Science, Carl von Ossietzky Universität Oldenburg, Oldenburg, Germany
{fraenzle,peter.nazier.mosaad}@informatik.uni-oldenburg.de
[3] University of Tartu, Tartu, Estonia

Abstract. The possible interactions between a controller and its environment can naturally be modelled as the arena of a two-player game, and adding an appropriate winning condition permits to specify desirable behavior. The classical model here is the positional game, where both players can (fully or partially) observe the current position in the game graph, which in turn is indicative of their mutual current states. In practice, neither sensing or actuating the environment through physical devices nor data forwarding to and signal processing in the controller are instantaneous. The resultant delays force the controller to draw decisions before being aware of the recent history of a play. It is known that existence of a winning strategy for the controller in games with such delays is decidable over finite game graphs and with respect to ω-regular objectives. The underlying reduction, however, is impractical for non-trivial delays as it incurs a blow-up of the game graph which is exponential in the magnitude of the delay. For safety objectives, we propose a more practical incremental algorithm synthesizing a series of controllers handling increasing delays and reducing game-graph size in between. It is demonstrated using benchmark examples that even a simplistic explicit-state implementation of this algorithm outperforms state-of-the-art symbolic synthesis algorithms as soon as non-trivial delays have to be handled. We furthermore shed some light on the practically relevant case of non-order-preserving delays, as arising in actual networked control, thereby considerably extending the scope of regular game theory under delay pioneered by Klein and Zimmermann.

Keywords: Safety games · Control under delay · Efficient algorithmic synthesis

1 Introduction

Algorithmic game theory is an established approach to the synthesis of correct-by-construction reactive controllers [12,15]. A finite game graph is used to formalize the

* William Shakespeare, Twelfth Night/What You Will, Act 2, Scene 3.

The first and fifth authors are funded partly by NSFC under grant No. 61625206 and 61732001, by "973 Program" under grant No. 2014CB340701, and by the CAS/SAFEA International Partnership Program for Creative Research Teams. The second and fourth authors are supported partly by Deutsche Forschungsgemeinschaft under grant No. DFG RTG 1765 SCARE. The third author is funded partly by NSFC under grant No. 61502467 and ...

possible actions of the players; it is complemented by a winning condition specifying desirable properties of infinite paths by means of an acceptance condition or a specification in temporal logic. Frequently, the game is played on a finite graph alternating moves by two players; the first player is the controller (sometimes called "ego" player) and the second player is its environment ("alter"), which may be uncooperative, erratic, or even malicious. Correct controllers thus have to be able to counteract any environmental actions, i.e., they need a sure winning strategy in the game. Controller synthesis can thus be understood as search for a winning strategy for ego. In this paper, we are interested in the synthesis problem when the interaction of a controller and its environment is described by a safety game [12], i.e., an infinite two-player game on finite graphs comprising "unsafe" states that the controller should avoid visiting.

These safety games have traditionally been investigated in a setting where the current position in the game is either fully known ("perfect information") or known up to certain observability constraints ("imperfect/incomplete information"). In this article, we address the problem of control under delays in perception and action. This can be understood as a form of imperfect information, as decisions by the controller have to be drawn based on delayed state observation —i.e., with the recent game history being opaque to the controller— and in advance —i.e., well before the actual situation where the action takes effect is fully determined. Such games have numerous practical applications, especially in networked control settings like cooperative driving, where observation of and influence on other cars' states are delayed by communication protocols severely restricting frequency of certain message types in order to keep overall channel usage sustainable under the pertinent severe bandwidth constraints.

It is intuitively obvious that such delay renders control harder: the controller has to decide in advance and based on dated information, which may no longer be fully indicative of the current situation. The existence of a winning strategy for the controller under such delays is decidable over finite game graphs and with respect to ω-regular objectives [9,10]. The underlying reduction to delay-free games, however, is impractical for non-trivial delays as it incurs a blow-up of the game graph which is strictly exponential in the magnitude of the delay, as also observed by Tripakis [19].

In this article, we follow Tripakis' quest for more efficient algorithms. For safety objectives, we propose a more practical incremental algorithm synthesizing a series of controllers handling increasing delays and reducing game-graph size in between. We demonstrate on benchmark examples that even a simplistic explicit-state implementation of this algorithm outperforms state-of-the-art symbolic synthesis algorithms as soon as non-trivial delays have to be handled. We furthermore shed some light on the practically relevant case of non-order-preserving delays, as arising in actual networked control, thereby considerably extending the scope of regular game theory under delay/lookahead pioneered by Klein and Zimmermann in [9,10,21] and explained below.

Related work. In the literature on games, constraints on observation and interaction are reflected by corresponding restrictions on the information frames available to the players. The majority of the results about two-player games played on graphs adopt the hypothesis of *perfect information*, where fixed-point algorithms for the computation of winning strategies exist [6,5,15]. In this case, the controller is aware of the exact current (and past) state of its environment when selecting its next control action. Reif [16]

has studied games of *incomplete information* and Kupferman and Vardi in [11] have extended the work of Pnueli and Rosner [14] about the synthesis of reactive modules to consider incomplete information. Similarly [20] and [15] study two-player games on graphs with ω-regular objectives subject to partial observability of the current (and past) game state. Recent state information is available, however; no restriction concerning the minimum age of observable state information is imposed. As the latter is an increasingly relevant problem in, e.g., networked control with its non-trivial end-to-end communication latencies, we here address the problem of two-player safety games subject to *delayed observation* and *delayed action* of the controlled process, obtaining a specific (and practically extremely relevant) case of imperfect information amenable to optimized synthesis algorithms.

The notion of control under delayed information exchange between the controller and the environment, where both the ego and the alter player suffer from having to operate under dated information about their mutual adversary's state, is complementary to the notion of delayed ω-regular games investigated by Zimmermann et al. [10,9]. In their setting, a delayed output player lags behind the input player in that the output player has to produce the i-th letter of the output string only when $i+\sum_{j=0}^{i} f(j)$ letters of the input string are available, with $\forall j : f(j) \geq 0$. Thus, delay essentially comes as an advantage, as the input player grants the output player a lookahead — the burden for the output player is "just" that she may have to memorize (a finite abstraction of) infinite lookahead if delay is unbounded in that $\sum_{j=0}^{i} f(j)$ diverges. In Zimmermann's terminology, our setting can be understood as asking for a strategy of the input player —whose strategic strength suffers from having to grant a lookahead— rather than for the output player and under the condition that delay is constant, i.e., $f(0) > 0$ and $\forall i > 0 : f(i) = 0$. We exploit a similar reduction to games of perfect information as the oblivious-delay construction of Zimmermann [21], which in the case of constant delay exploits a product construction on the game graph essentially representing a synchronous concurrent composition of the graph with a shift register implementing the delays. In contrast to Zimmermann et al., we do not grant introspection into the shift register, i.e., lookahead into an adversary's future actions. We do instead adopt the perspective of their input player, who has to submit her actions without knowledge of the recent history, as is frequently the case in practice. For this setting, the above reduction by means of a shift register also provides a consistent semantics of playing under delay.

It is worth noting that the notion of delay employed in this paper and by Klein and Zimmermann in [10] is different from that in timed games and their synthesis algorithms, like UPPAAL-TIGA [2], as well as from that used in the discrete-event system community, e.g. [13,1]. In timed games, delay refers to the possibility to deliberately delay the next control action, i.e., a single event. Up-to-date positional information, however, is always fully transparent to both players in timed games. In our setting, delay refers to a time lag imposed when obtaining positional information, modelling the end-to-end latency of information distribution in a communication network. Up-to-date positional information thus is opaque to the players as long as it resides in a queue modelling the network, where state information as well as control events of multiple different ages coexist and pipeline towards delivery. Such pipelining of control actions is lacking in the model of delay from [13], where only one controllable event can be latent at any time

and just the time of its actual execution is determined by the environment. Meanwhile, the model of delay in [1] is different from ours as it leads to non-regular languages.

2 Safety Games under Delayed Information

Notation. Given a set A, we denote its powerset by 2^A, the set of finite sequences over A by A^*, and the set of infinite sequences over A by A^ω. The relative complement of a set B in A is denoted $A \setminus B = \{x \in A \mid x \notin B\}$. An empty sequence is denoted by ε.

2.1 Games with perfect information

The plays we consider are played on finite bipartite game graphs as known from ω-regular games, see e.g. [18]:

Definition 1 (Two-player game graph). *A* finite game graph *is of the form* $G = \langle S, s_0, S_0, S_1, \Sigma, \to \rangle$*, where* S *is a finite (non-empty) set of states,* S_0*,* S_1 *define a partition of* S *(*S_i *containing the states where it is the turn of player* i *to perform an action),* $s_0 \in S_0$ *is the* initial *state,* Σ *is a finite alphabet of* actions *for player 0 (while any action for player 1 is abstracted as* $u \notin \Sigma$*), and* $\to \subseteq S \times (\Sigma \cup \{u\}) \times S$ *is a set of labeled transitions satisfying the following four conditions:*

Bipartition: *if* $s \in S_i$ *and* $s \xrightarrow{\sigma} s'$ *for some* $\sigma \in \Sigma \cup \{u\}$ *then* $s' \in S_{1-i}$*;*
Absence of deadlock: *for each* $s \in S$ *there exist* $\sigma \in \Sigma \cup \{u\}$ *and* $s' \in S$ *s.t.* $s \xrightarrow{\sigma} s'$*;*
Alphabet restriction on actions: *if* $s \xrightarrow{\sigma} s'$ *for some* $\sigma \in \Sigma \cup \{u\}$ *then* $\sigma \in \Sigma$ *iff* $s \in S_0$ *(and consequently,* $\sigma = u$ *iff* $s \in S_1$*);*
Determinacy of Σ **moves:** *if* $s \in S_0$ *and* $s \xrightarrow{\sigma} s_1$ *and* $s \xrightarrow{\sigma} s_2$ *then* $s_1 = s_2$*.*

The state space is required to be deadlock-free and bipartite with respect to the transitions, which thus alternate between S_0 and S_1 states. Furthermore, the actions of player 0 are from Σ and deterministic, while all actions of player 1 are lumped together into a non-deterministic u action, since we are interested in synthesizing a winning strategy merely for player 0 who models the controller.

The game is played by a controller (player 0, ego) against an environment (player 1, alter) in turns. Starting from $s = s_0$ and in each second turn, the controller chooses an action $\sigma \in \Sigma$ that is enabled in the current state s. By $s \xrightarrow{\sigma} s'$, this leads the game to a unique successor state $s' \in S_1$. From s', it now is the environment's turn to select an action, which it does by selecting a successor state $s'' \in S_0$ with $s' \xrightarrow{u} s''$. As s'' again is a position controlled by player 0, the game alternates between moves of player 0 (the controller) and player 1 (the environment) forever, leading to the following definition.

Definition 2 (Infinite play). *A* play *on game graph* $G = \langle S, s_0, S_0, S_1, \Sigma, \to \rangle$ *is an infinite sequence* $\pi = \pi_0 \sigma_0 \pi_1 \ldots \sigma_{n-1} \pi_n \sigma_n \ldots$ *s.t.* $\pi_0 = s_0$*, and* $\forall i \in \mathbb{N} : \pi_i \xrightarrow{\sigma_i} \pi_{i+1}$*.*

The game graph is accompanied by a *winning condition*. In a *safety game*, this is a set of *unsafe positions* $\mathcal{U} \subseteq S$ and the controller loses (and thus the environment wins) as soon as the play reaches an unsafe state $s_i \in \mathcal{U}$. Conversely, the controller wins (and the environment loses) iff the game goes on forever without ever visiting $\mathcal{U}$.

Definition 3 (Two-player safety game). *A* two-player safety game *is of the form* $G = \langle S, s_0, S_0, S_1, \Sigma, \mathcal{U}, \rightarrow \rangle$*, where* $G' = \langle S, s_0, S_0, S_1, \Sigma, \rightarrow \rangle$ *is a finite game graph and* $\mathcal{U} \subseteq S$ *is a set of* unsafe positions.
$\Pi(G)$ *denotes the set of plays over the underlying game graph* G'*. Play* $\pi_0\sigma_0\pi_1 \ldots \in \Pi(G)$ *is* won by player 0 *iff* $\forall i \in \mathbb{N} : \pi_i \notin \mathcal{U}$ *and* won by player 1 *otherwise.*

The objective of the controller in a safety game thus is to always select actions avoiding unsafe states, while the hostile or just erratic environment would try to drive the game to a visit of an unsafe state by picking adequate successor states on u actions.

For a given play $\pi \in \Pi(G)$, its *prefix* up to position π_n is denoted $\pi(n)$. This prefix thus is the finite sequence $\pi(n) = \pi_0\sigma_0\pi_1 \ldots \sigma_{n-1}\pi_n$, whose *length* is $|\pi(n)| = n + 1$ and whose *last* element is $\texttt{Tail}(\pi(n)) = \pi_n$. The set of prefixes of all plays in $\Pi(G)$ is denoted by $\texttt{Pref}(G)$, in which we denote those ending in a controller state by $\texttt{Pref}_c(G) = \{\rho \in \texttt{Pref}(G) \mid \texttt{Tail}(\rho) \in S_0\}$. Likewise, $\texttt{Pref}_e(G) = \{\rho \in \texttt{Pref}(G) \mid \texttt{Tail}(\rho) \in S_1\}$ marks prefixes of plays ending in environmental positions.

For a game $G = \langle S, s_0, S_0, S_1, \Sigma, \mathcal{U}, \rightarrow \rangle$, a *strategy* for the controller is a mapping $\xi : \texttt{Pref}_c(G) \mapsto 2^\Sigma$ s.t. all $\sigma \in \xi(\rho)$ are enabled in $\texttt{Tail}(\rho)$ and $\xi(\rho) \neq \emptyset$ for any $\rho \in \texttt{Pref}_c(G)$. The *outcome* of the strategy ξ in G is defined as $O(G, \xi) = \{\pi = \pi_0\sigma_0\pi_1 \ldots \in \Pi(G) \mid \forall i \in \mathbb{N} : \sigma_{2i} \in \xi(\pi(2i))\}$ and denotes all plays possible when player 0 respects strategy ξ while player 1 plays arbitrarily.

Definition 4 (Winning strategy for the controller). *A strategy* ξ *for the controller in a safety game* $G = \langle S, s_0, S_0, S_1, \Sigma, \mathcal{U}, \rightarrow \rangle$ *is* winning for the controller *(or just* winning *for short) iff* $\forall \pi = \pi_0\sigma_0\pi_1 \ldots \in O(G, \xi). \forall k \in \mathbb{N} : \pi_k \notin \mathcal{U}$.

A winning strategy for the environment can be defined similarly as being a mapping $\tilde{\xi} : \texttt{Pref}_e(G) \mapsto 2^{S_0}$ with equivalent well-defined conditions as above. It is a classical result of game theory that such safety games under perfect observation are determined: one of the two players has a sure winning strategy enforcing a win irrespective of the opponent's choice of actions.

Theorem 1 (Determinacy [8]). *Safety games are determined, i.e., in each safety game* $G = \langle S, s_0, S_0, S_1, \Sigma, \mathcal{U}, \rightarrow \rangle$ *exactly one of the two players has a winning strategy.*

We call a (controller) strategy $\xi : \texttt{Pref}_c(G) \mapsto 2^\Sigma$ *positional* (or *memoryless*) if for any ρ and $\rho' \in \texttt{Pref}_c(G)$, $\texttt{Tail}(\rho) = \texttt{Tail}(\rho')$ implies $\xi(\rho) = \xi(\rho')$. Being positional implies that at any position in a play, the next decision of a controller which follows the strategy only depends on the current position in the game graph and not on the history of the play. As a consequence, such a positional strategy can also be described by a function $\xi' : S_0 \mapsto 2^\Sigma$ that maps every state of the controller in the game to a set of actions to be performed whenever the state is visited. The reduction to positional strategies is motivated by the fact that in delay-free safety games, whenever there exists a winning strategy for the controller, then there also exists a positional strategy for it.

Theorem 2 (Computing positional strategies [7,18]). *Given a two-player safety game* G*, the set of states from which player 0 (player 1, resp.) can enforce a win is computable, and memoryless strategies are sufficient for the winning party.*

The construction of a positional strategy builds on backward fixed-point iteration computing the set of states from which a visit in $\mathcal{U}$ can be enforced by player 1 [18].

2.2 Games under delayed control

As immediately obvious from the fact that memoryless strategies suffice in the above setting, being able to fully observe the *current* state and to react on it immediately is an essential feature of the above games. In practice, this is often impossible due to delays between sensing the environmental state, computing the control action, submitting it, and it taking effect. The strategy, if existent, thus cannot resort to the full state history, but only to a proper prefix thereof due to the remainder becoming visible too late.

If the delay is constant and equates to $\delta \in \mathbb{N}$ steps, then the controller would have to decide about the action to be taken after some finite play $\pi_0\sigma_0\pi_1 \dots \pi_{2n}$ already after just seeing its proper prefix $\pi_0\sigma_0\pi_1 \dots \pi_{2n-\delta}$. Furthermore, a constant strategy not dependent on any historic observations would have to be played by the controller initially for the first δ steps. That motivates the following definition:

Definition 5 (Playing under delay). *Given a delay $\delta \in \mathbb{N}$, a* strategy for the controller under delay δ *is a map $\xi : \mathtt{Pref}_x(G) \mapsto 2^{\Sigma}$, where $x = c$ if δ is even and $x = e$ else, together with a non-empty set $\alpha \subseteq \Sigma^{\lceil \frac{\delta}{2} \rceil}$ of initial action sequences. The* outcome of playing strategy (α, ξ) in G under delay δ *is* $O(G, \alpha, \xi, \delta) =$

$$\left\{ \pi = \pi_0\sigma_0\pi_1 \dots \in \Pi(G) \;\middle|\; \begin{array}{l} \exists a = a_0 \dots a_{\lceil \frac{\delta}{2} \rceil - 1} \in \alpha . \forall i \in \mathbb{N} : \\ \left(\begin{array}{c} 2i < \delta \Rightarrow \sigma_{2i} = a_i \\ \wedge\ 2i \geq \delta \Rightarrow \sigma_{2i} \in \xi(\pi(2i - \delta)) \end{array} \right) \end{array} \right\}.$$

We call the strategy (α, ξ) playable *by the controller iff it always assigns permitted moves, i.e., iff for each prefix $\pi_0\sigma_0\pi_1 \dots \sigma_{2n-1}\pi_{2n-1}$ of a play in $O(G, \alpha, \xi, \delta)$, we have that the set of next actions*

$$\Sigma_n = \begin{cases} \{a_n \mid \langle \sigma_0, \sigma_2, \sigma_4, \dots, \sigma_{2n-2}, a_n \rangle \textit{ is a prefix of a word in } \alpha\} & \textit{iff } 2n < \delta, \\ \xi(\pi(2n - \delta)) & \textit{iff } 2n \geq \delta \end{cases}$$

suggested by the strategy is non-empty and contains only actions enabled in π_{2n-1}. Strategy (α, ξ) is winning (for the controller) under delay δ *iff it is playable and for each $\pi = \pi_0\sigma_0\pi_1 \dots \in O(G, \alpha, \xi, \delta)$, the condition $\forall k \in \mathbb{N} : \pi_k \notin \mathcal{U}$ holds, i.e., no unsafe state is ever visited when playing the strategy.*

Playing under a delay of δ thus means that for a play $\pi = \pi_0\sigma_0\pi_1 \dots$, the choice of actions suggested by the winning strategy at state π_{2i} has to be pre-decided at state $\pi_{2i-\delta}$ for any $i \geq \lceil \frac{\delta}{2} \rceil$ and decided without recourse to positional information for the first $\delta - 1$ steps. Playing under delay 0 is identical to playing under complete information.

From Def. 5 it is obvious that existence of a (delay-free) winning strategy in the complete information game G is a necessary, yet not sufficient condition for existence of a strategy that is winning under a delay of $\delta > 0$. Likewise, existence of a strategy winning under some relatively small delay δ is a necessary, yet not sufficient condition for existence of a strategy that is winning under a delay of $\delta' > \delta$: the strategy for δ' can be played for δ by simply waiting $\delta' - \delta$ steps before implementing the control action.

Remark 1. The reader may wonder why Def. 5 assumes strictly sequential delay, i.e., in-order delivery of the delayed information, which cannot be guaranteed in many practical applications of networked control. The reason is that random out-of-order delivery

with a maximum delay of δ has in-order delivery with an exact delay of δ as its worst-case instance: whenever a data item is delivered out-of-order then it is delivered before δ, implying earlier availability of more recent state information and thus enhanced controllability. In a qualitative setting, as addressed in this article, solving the control problem for out-of-order delivery with a maximum delay of δ is consequently —up to delaying data items arriving early— identical to solving the control problem under in-order delivery with an exact delay of δ, as the latter is the former's worst case.

Issues are, however, different in a stochastic setting, where out-of-order delivery with a maximum delay of δ induces a reduced expected message delay strictly smaller than δ, i.e., it even truly enhances controllability. Dealing with this basic quantitative case and furthermore exploiting constructive means of control on message delay, like setting a network's QoS parameters, for control will be subject of future research.

2.3 Insufficiency of memoryless strategies

Recall that in safety games with complete information, the existence of a winning strategy for the controller implies existence of a memoryless strategy for player 0. For games with delayed information, however, memoryless strategies are not powerful enough:

Example 1. Consider the safety game $G = \langle S, s_0, S_0, S_1, \Sigma, \mathcal{U}, \rightarrow \rangle$, shown in Fig. 2, where $S = S_0 \cup S_1$, $S_0 = \{c_1, c_2, c_3\}$, $S_1 = \{e_1, e_2, e_3, e_4, e_5\}$, $s_0 = c_1$, $\Sigma = \{a, b\}$, and $\mathcal{U} = \{e_3\}$. Player 0 can obviously win this safety game if no delay is involved. Now consider a memoryless strategy $\xi' : S_0 \mapsto 2^{\Sigma}$ for the controller under delay 2. We obviously need $\xi'(c_2) = \{b\}$, indicating that the controller executes b two steps later at either c_1 or c_3, as a at c_3 would yield the unsafe state e_3. Analogously, we have $\xi'(c_3) = \{a\}$. It is a different matter when arriving at c_1, where the controller has to draw a pre-decision for both c_2 and c_3. If the controller picks a (or b) at c_1, then two steps later at c_3 (c_2, resp.) it executes the unsafe action a (b, resp.). For a win, extra memory keeping track of the historic sequence of actions is necessary such that the controller can determine whether it will visit c_2 or c_3 from c_1.

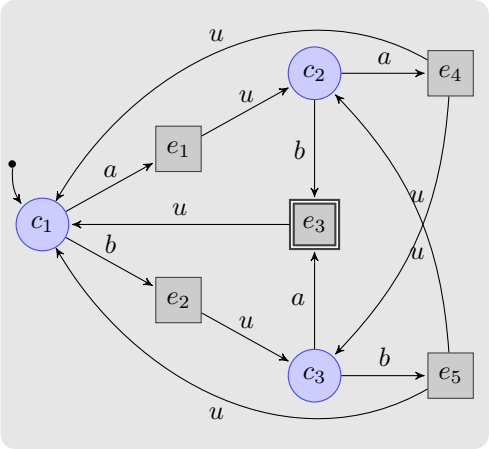

Fig. 2: A safety game winnable with memoryless strategies for delay $\delta \leq 1$, yet not beyond.

The above example shows that memoryless strategies are generally insufficient for winning a safety game under delays. A straightforward generalization of the situation shown in Fig. 2, namely deeply nesting triangles of the shape spanned by c_1, c_2, and c_3, demonstrates that the amount of memory needed will in worst case be exponential in the delay. Any reduction to safety games under complete information will have to introduce a corresponding blow-up of the game graph.

2.4 Reduction to delay-free games

As playing a game under delay δ amounts to pre-deciding actions δ steps in advance, the problem of finding a winning strategy for the controller in $G = \langle S, s_0, S_0, S_1, \Sigma, \mathcal{U}, \to \rangle$ that wins under delay δ can be reduced to the problem of finding an undelayed winning strategy for the controller in a related safety game:

Lemma 1. *Let $G = \langle S, s_0, S_0, S_1, \Sigma, \mathcal{U}, \to \rangle$ be a safety game and $\delta \in \mathbb{N}$ a delay. Then the controller has strategy that wins G under a delay δ iff the controller has a winning strategy in the game $\widehat{G} = \langle S', s_0', S_0', S_1', \Sigma \cup \Sigma^{\lceil \frac{\delta}{2} \rceil}, \mathcal{U}', \to' \rangle$ given by*

1. $S' = \left(S \times \Sigma^{\lceil \frac{\delta}{2} \rceil}\right) \uplus \{s_0'\} \uplus \left(\{s_0'\} \times \Sigma^{\lceil \frac{\delta}{2} \rceil}\right)$, *where $\uplus$ denotes disjoint union,* $S_0' = \left(S_0 \times \Sigma^{\lceil \frac{\delta}{2} \rceil}\right) \cup \{s_0'\}$, *and* $S_1' = \left(S_1 \times \Sigma^{\lceil \frac{\delta}{2} \rceil}\right) \cup \left(\{s_0'\} \times \Sigma^{\lceil \frac{\delta}{2} \rceil}\right)$,
2. $s \overset{\sigma}{\to}' s'$ *iff*

$$\begin{aligned} & s = s_0' \wedge \sigma = a_1 \dots a_n \in \Sigma^n \wedge s' = (s_0', a_1 \dots a_n) \\ \vee\ & s = (s_0', \alpha) \wedge \sigma = u \wedge s' = (s_0, \alpha) \\ \vee\ & s = (\hat{s}, a_1 \dots a_n) \wedge \hat{s} \in S_0 \wedge \sigma \in \Sigma \wedge \hat{s} \xrightarrow{a_1} \hat{s}' \wedge s' = (\hat{s}', a_2 \dots a_n \sigma) \\ \vee\ & s = (\hat{s}, \alpha) \wedge \hat{s} \in S_1 \wedge \sigma = u \wedge \hat{s} \xrightarrow{u} \hat{s}' \wedge s' = (\hat{s}', \alpha), \end{aligned}$$

   where $n = \frac{\delta}{2}$ if δ is even and $n = \frac{\delta+1}{2}$ if δ is odd.
3. $\mathcal{U}' = \mathcal{U} \times \Sigma^{\lceil \frac{\delta}{2} \rceil}$.

The essential idea of the above reduction is to extend the game graph by a synchronous product with a shift register appropriately delaying the implementation of the control action decided by the controller. The blow-up in graph size incurred is by a factor $|\Sigma|^{\lceil \frac{\delta}{2} \rceil}$ and thus exponential in the delay. It is obvious that due to this, a winning strategy for the controller in the delayed game can, if existent, be synthesized with $|\Sigma|^{\lceil \frac{\delta}{2} \rceil}$ memory.

Note that the above reduction to delay-free safety games does not imply that games under delay are determined, as the claim in Lemma 1 is not symmetric for the environment. A simple guessing game, where player 1 guesses in each step either a 0 or a 1 and player 0 has to repeat the exact guess, losing as soon as she fails to properly repeat, reveals that player 0 has a sure winning strategy under delay 0, but none of the two players has one under any positive delay.[1] Determinacy is only obtained if one of the players is granted a lookahead equivalent to the other's delay, as in Klein and Zimmermann's setting [10]. Such lookahead does not, however, correspond to any physical reality in distributed control, where both players are subject to the same end-to-end latency (i.e., delay) in their mutual feedback loop.

3 Synthesizing Controllers

As stated above, controller synthesis for games under delay can be obtained using a reduction to a delay-free safety game involving the introduction of a shift register. The

[1] While player 1 could enforce a win with probability 1 in a probabilistic setting by just playing a random sequence, she cannot enforce a win in the qualitative setting where player 0 may just be lucky to draw the right guesses throughout.

exponential blow-up incurred by this reduction, however, seems impractical for any non-trivial delay. We therefore present a novel incremental synthesis algorithm, which starts from synthesizing a winning strategy for the underlying delay-free safety game and then incrementally hardens the strategy against larger and larger delays, thus avoiding explicit reductions. We further optimize the algorithm by pruning the otherwise exponentially sized game graph after each such hardening step: as controllability (i.e., the controller wins) under delay k is a necessary condition for controllability under delay $k' > k$, each state uncontrollable under delay k can be removed before proceeding to the next larger delay. The algorithm thus alternates between steps extending memory, as necessary for winning under delay, and steps compressing the game graph.

The key idea of the synthesis procedure (Algorithm 1) is to compute a series of finite-memory winning strategies $\hat{\xi}_k$ while increasing delays from $k = 0$ to the final delay of interest $k = \delta$. The algorithm takes as input a delayed safety game G_δ and returns either `WINNING` paired with a winning strategy $(\alpha, \hat{\xi}_\delta)$ for the controller if G_δ is controllable, or `LOSING` otherwise with an integer m indicating that the winning strategy vanishes when lifting delay to m. Line 2 invokes the classical fixed-point iteration (cf. Appendix C) to generate the *maximally permissive strategy* for the controller in G under no delay. The procedure **FPIteration** first conducts a backward fixed-point iteration computing the set L of states from which a visit to $\mathcal{U}$ can be enforced by the alter player 1 [18]. The maximally permissive strategy for the controller is then obtained by admitting in each state from $S_0 \setminus L$ exactly those actions leading to a succesor in $S_1 \setminus L$. Then the delays are lifted from $k = 0$ to δ by a `while` loop in line 3, and within each step of the loop the strategy $\hat{\xi}_{k+1}$ is computed based on $\hat{\xi}_k$ as follows:

1. If $k + 1$ is an odd delay, the controller needs to make pre-decisions at safe states of the environment, namely at each $s \in S_1 \setminus \mathcal{U}$. The controller needs to pre-decide at s a set of actions that are safe to perform at any successor $s' \in \texttt{Succ}(s)$, for which the winning actions have already been encoded in the strategy $\hat{\xi}_k(s', \cdot)$. This is achieved, in line 7, by taking an intersection of $\hat{\xi}_k(s', \rho)$ for all $s' \in \texttt{Succ}(s)$ with the same history sequence of actions ρ. The derived strategy can be spurious however, inasmuch as the intersection involves only immediate successors of s, yet without observing the entire strategy space. At line 9 we therefore remove all uncontrollable predecessors of freshly unwinnable states by a **Shrink** procedure depicted in Algorithm 2, which will be explained below.
2. In case of an even delay $k + 1$, the controller needs to make pre-decisions at safe states of its own, i.e. at each $s \in S_0 \setminus \mathcal{U}$. In contrast to an intersection in the odd case, the controller can inherit the winning strategy $\hat{\xi}_k(s', \rho)$ directly from each successor s' of s. However, we have to prepend, if $s \xrightarrow{\sigma_0} s'$, the action σ_0 to the history sequence ρ to record the choice in the shift register (line 19).

The synthesis algorithm may abort at line 14 if the controller does not have available actions to pick anymore at the initial state s_0, declaring `LOSING` at $k + 1$ where the winning strategy vanishes. Otherwise, the algorithm continues and eventually produces a winning strategy $\hat{\xi}_\delta$ for the controller in G.

Only when a fresh unwinnable state s for the controller is detected (line 8), the **Shrink** function (Algorithm 2) will be launched to carry out two tasks in a recursive

Algorithm 1: Synthesizing winning finite-memory strategy

input : $G = \langle S, s_0, S_0, S_1, \Sigma, \mathcal{U}, \rightarrow \rangle$, a safety game played under delay δ.

/* initialization */

$k \leftarrow 0$; $\alpha \leftarrow \{\varepsilon\}$;

/* computing maximally permissive strategy under no delay */

$\hat{\xi}_0 \leftarrow$ **FPIteration**(G);

/* lifting delays from 0 to δ */

while $k < \delta$ **do**

  /* with an odd delay $k+1$ */

  if $k \equiv 0 \pmod 2$ **then**

    for $s \in S$, $\sigma_1 \dots \sigma_{\frac{k}{2}} \in \alpha$ **do**

      if $s \in S_1 \setminus \mathcal{U}$ **then**

        $\hat{\xi}_{k+1}(s, \sigma_1 \dots \sigma_{\frac{k}{2}}) \leftarrow \bigcap_{s' : s \xrightarrow{u} s'} \hat{\xi}_k(s', \sigma_1 \dots \sigma_{\frac{k}{2}})$;

        /* shrinking the possibly-spurious strategy */

        if $\hat{\xi}_{k+1}(s, \sigma_1 \dots \sigma_{\frac{k}{2}}) = \emptyset$ and $\bigwedge_{s' : s \xrightarrow{u} s'} \hat{\xi}_k(s', \sigma_1 \dots \sigma_{\frac{k}{2}}) \neq \emptyset$ **then**

          Shrink$(\hat{\xi}_{k+1}, \hat{\xi}_k, G, (s, \sigma_1 \dots \sigma_{\frac{k}{2}}))$;

      else

        $\hat{\xi}_{k+1}(s, \sigma_1 \dots \sigma_{\frac{k}{2}}) \leftarrow \emptyset$;

    $\alpha \leftarrow \{\sigma_0 \sigma_1 \cdots \sigma_{\frac{k}{2}} \mid s_0 \xrightarrow{\sigma_0} s', \sigma_1 \cdots \sigma_{\frac{k}{2}} \in \alpha, \hat{\xi}_{k+1}(s', \sigma_1 \cdots \sigma_{\frac{k}{2}}) \neq \emptyset\}$;

    if $\alpha = \emptyset$ **then**

      return $(\texttt{LOSING}, k+1)$;

  /* with an even delay $k+1$ */

  else

    for $s \in S$, $\sigma_1 \dots \sigma_{\frac{k-1}{2}} \in \alpha$ **do**

      if $s \in S_0 \setminus \mathcal{U}$ **then**

        for $\sigma_0, s' : s \xrightarrow{\sigma_0} s'$ **do**

          $\hat{\xi}_{k+1}(s, \sigma_0 \sigma_1 \dots \sigma_{\frac{k-1}{2}}) \leftarrow \hat{\xi}_k(s', \sigma_1 \dots \sigma_{\frac{k-1}{2}})$;

      else

        $\hat{\xi}_{k+1}(s, \sigma_0 \sigma_1 \dots \sigma_{\frac{k-1}{2}}) \leftarrow \emptyset$;

  $k \leftarrow k+1$;

return $(\texttt{WINNING}, (\alpha, \hat{\xi}_k))$;

manner: (1) it traverses the graph backward and removes from the current strategy all the actions that may lead the play to this unwinnable state, and consequently (2) it gives a state-space pruning that removes all states no longer controllable under the given delay before proceeding to the next larger delay. The latter accelerates synthesis, while the former is a key ingredient to the correctness of Algorithm 1, as can be seen from the proof of Theorem 3: it avoids "blind alleys" where locally controllable actions run towards subsequently deadlocked states.

Algorithm 2: Shrink: Shrinking the possibly-spurious strategy

input : $\hat{\xi}_{2n+1}$, the strategy under an odd delay $2n+1$;
$\hat{\xi}_{2n}$, the strategy under an even delay $2n$;
$G = \langle S, s_0, S_0, S_1, \Sigma, \mathcal{U}, \rightarrow \rangle$, a safety game played under delay δ;
$(s, \sigma_1 \dots \sigma_n)$, a fresh unwinnable state with the sequence of actions.

for $s' : s' \xrightarrow{\sigma} s$ **do**
  if $\sigma_n \in \hat{\xi}_{2n}(s', \sigma\sigma_1 \dots \sigma_{n-1})$ **then**
    $\hat{\xi}_{2n}(s', \sigma\sigma_1 \dots \sigma_{n-1}) \leftarrow \hat{\xi}_{2n}(s', \sigma\sigma_1 \dots \sigma_{n-1}) \setminus \{\sigma_n\}$;
    `/* ` $\tilde{s} < s$ `indicates the existence of` $\hat{\xi}_{2n+1}(\tilde{s}, \cdot)$`, i.e., we visit merely states that have already been attached with (possibly deadlocking) actions by Alg. 1 */`
    for $\tilde{s} : \tilde{s} \xrightarrow{u} s'$ and $\tilde{s} \notin \mathcal{U}$ and $\tilde{s} < s$ **do**
      if $\sigma_n \in \hat{\xi}_{2n+1}(\tilde{s}, \sigma\sigma_1 \dots \sigma_{n-1})$ **then**
        $\hat{\xi}_{2n+1}(\tilde{s}, \sigma\sigma_1 \dots \sigma_{n-1}) \leftarrow \hat{\xi}_{2n+1}(\tilde{s}, \sigma\sigma_1 \dots \sigma_{n-1}) \setminus \{\sigma_n\}$;
        if $\hat{\xi}_{2n+1}(\tilde{s}, \sigma\sigma_1 \dots \sigma_{n-1}) = \emptyset$ **then**
          Shrink$(\hat{\xi}_{2n+1}, \hat{\xi}_{2n}, G, (\tilde{s}, \sigma\sigma_1 \dots \sigma_{n-1}))$;

The worst-case complexity of Alg. 1 follows straightforwardly as $O(\delta \cdot |S_0| \cdot |S_1| \cdot |\Sigma|^{\lfloor \frac{\delta}{2} \rfloor})$, as is the case for the reduction to a delay-free safety games. In practice, the advantage however is that we avoid explicit construction of the graph of the corresponding delay-free game, which yields an exponential blow-up, and interleave the expansion by yet another shift-register stage with state-set shrinking removing uncontrollable states.

Theorem 3 (Correctness and Completeness). *Algorithm 1 always terminates. If its output is* $(\texttt{WINNING}, (\alpha, \hat{\xi}))$ *then* $(\alpha, \hat{\xi})$ *is a winning strategy of* G_δ*; otherwise, with output* $(\texttt{LOSING}, k+1)$ *of the algorithm,* G_δ *has no winning strategy.*

Proof. Elaborated in Appendix A.

Example 2. Consider the safety game G under delayed information in Fig. 2. The series of finite-memory winning strategies produced by Algorithm 1 is:

$$\hat{\xi}_0(c_1, \varepsilon) = \{a, b\}, \qquad \hat{\xi}_0(c_2, \varepsilon) = \{a\}, \qquad \hat{\xi}_0(c_3, \varepsilon) = \{b\}.$$

$$\hat{\xi}_1(e_1, \varepsilon) = \{a\}, \quad \hat{\xi}_1(e_2, \varepsilon) = \{b\}, \quad \hat{\xi}_1(e_3, \varepsilon) = \emptyset, \quad \hat{\xi}_1(e_4, \varepsilon) = \{b\}, \quad \hat{\xi}_1(e_5, \varepsilon) = \{a\}.$$

$$\hat{\xi}_2(c_1, a) = \{a\}, \qquad \hat{\xi}_2(c_2, a) = \{b\}, \qquad \hat{\xi}_2(c_3, a) = \emptyset,$$
$$\hat{\xi}_2(c_1, b) = \{b\}, \qquad \hat{\xi}_2(c_2, b) = \emptyset, \qquad \hat{\xi}_2(c_3, b) = \{a\}.$$

Winning strategies for the controller vanish when the delay reaches 3.

4 Case Study and Experimental Evaluation

Avoiding collisions is a central issue in transportation systems as well as in many other applications. The task of a collision avoidance (CA) system is to track objects of potential collision risk and determine any action to avoid or mitigate a collision. One of the

challenges in designing a CA system is determining the correct action in presence of the end-to-end latency of the overall control system.

In the context of avoiding collisions, we present an escape game as an artificial scenario to illustrate our approach. The game is a two-player game between a robot (i.e., the controller) and a kid (i.e., the dynamical part of its environment), which are moving in a closed room with some fixed obstacles as shown in Fig. 3. In this scenario, the robot has to make decisions (*actions*) under δ*-delayed information*.

Definition 6 (Two-player escape game in a $p \times q$ room under delay). *A* two-player escape game under delay δ *is of the form* $\widehat{G} = \langle S, s_0, S_0, S_1, \mathcal{O}, \Sigma, \mathcal{U}, \rightarrow \rangle$*, where*

- $S = X \times Y \times X \times Y \times \mathbb{B}$ *is a non-empty set of states providing* $x \in X = \{0, \ldots, p-1\}$ *and* $y \in Y = \{0, \ldots, q-1\}$ *coordinates for the robot as well as for the kid, together with a flag denoting whose move is next. Concretely, a state* (x_0, y_0, x_1, y_1, b) *encodes that the robot currently is at position* (x_0, y_0)*, while the kid is at* (x_1, y_1)*, and that the next move is the robot's iff* b *holds. Here* $p, q \in \mathbb{N}_{\geq 1}$ *denote the width and length of the room.*
- $\mathcal{O} \subseteq X \times Y$ *is a finite set of positions occupied by fixed obstacles.*
- Σ *is a finite alphabet of actions for player 0 (i.e., the robot), which consists of kinematically constrained moves explained below.*
- $\mathcal{U} \subseteq S$ *is the finite set of undesirable states, which are characterized by featuring collisions with the obstacles or the kid.*
- $\rightarrow \subseteq S \times (\Sigma \cup \{u\}) \times S$ *is a set of labelled transitions, and*
- δ *is the delay in information retrieval s.t. the robot has to react on* δ *old information.*

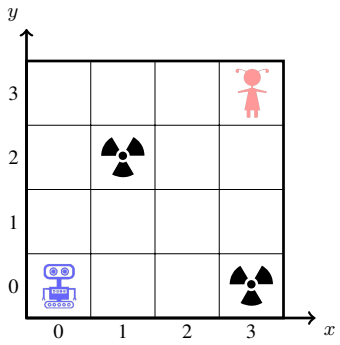

Fig. 3: The robot escape game

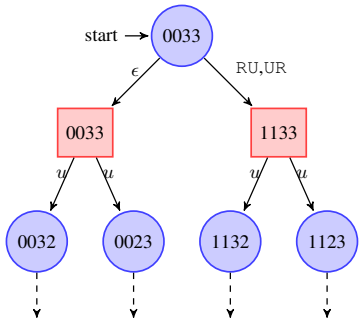

Fig. 4: A snippet of the game graph

In our scenario, we first consider a room of extent 4×4, as shown in Fig. 3. The fixed obstacles are located at $o_1 = (1, 2)$ and $o_2 = (3, 0)$ and the initial state s_0 where the robot and the kid are located in the room is $s_0 = (0, 0, 3, 3, \text{true}) \in S_0$. The kid can move in the room and her possible moves (i.e., the *uncontrollable actions*) are unilaterally denoted u for unpredictable, yet amount to moves either one step to the right `R`, left `L`, up `U`, or down `D`. The robot has a finite set of moves (i.e., *controllable actions*), which are kinematically constrained as being a combination of two moves, e.g., up then right `UR`, denoted as $\Sigma = \{\texttt{RU}, \texttt{UR}, \texttt{LU}, \texttt{UL}, \texttt{RD}, \texttt{DR}, \texttt{LD}, \texttt{DL}, \epsilon\}$, and ϵ means doing nothing. We assume that the two players respect the geometry of the room

and consequently never take any action leaving the inside area of the room or running through an obstacle, which can be achieved by specifying two groups of constraints $\mathcal{C}$ and $\mathcal{E}$ (exemplified in Appendix D) respectively for the robot and the kid, defining their legal actions. Representing a state (x_0, y_0, x_1, y_1, b) as $x_0y_0x_1y_1$ inside a blue circular node if b (robot's turn) and inside a red square node if $\neg b$ (kid's turn), the game graph spanned by the legal actions looks as shown in Fig. 4.

The safety objective for the robot is to move inside the working room while avoiding to ever be collocated with the kid or the fixed obstacles. We consequently define the set of unsafe states as $\mathcal{U} = \{(x_0, y_0, x_1, y_1, b) \mid (x_0, y_0) \in \mathcal{O} \vee (x_0, y_0) = (x_1, y_1)\}$.

There obviously exists a winning strategy for the robot in a delay-free setting, namely to cycle around the obstacle at o_1 to avoid being caught by the kid. To investigate the controllability resilient to delays, we first construct the graph structure from the symbolic description by a C++ program. It consists of 224 states, 16 unsafe states, and 738 legal transitions satisfying the respective conditions $\mathcal{C}$ and $\mathcal{E}$. The obtained game graph is then used as input to a prototypical implementation in Mathematica[2] of Algorithm 1, which declares `WINNING` paired with a finite-memory winning strategy (i.e., a safe controller) $\hat{\xi}_\delta$ under delays $0 \leq \delta \leq 2$ (see Appendix E), while `LOSING` when the delay is 3. The latter indicates that the problem is uncontrollable under any delay $\delta' \geq 3$.

To further investigate the scalability and efficiency of our method, we have evaluated the implementation on two additional examples (cf. Appendix B) as well as evasion games instantiated to rooms of different sizes (marked with prefix `Escp.`). A slightly adapted scenario (denoted by prefix `Stub.`) was also investigated, where the kid plays in a rather stubborn way, namely she always moves either one step to the left or down, yet never goes right nor up, which yields potentially larger affordable delays for the robot. In particular, a comparison of the performance of our incremental algorithm was done with respect to two points of reference: to the same Mathematica-based algorithm using $\delta = 0$ (the underlying explicit-state delay-free safety synthesis) employed after reducing the games to delay-free ones by shift registers (cf. Lemma 1), and to the state-of-the-art synthesizer SafetySynth[3] for solving safety games applied to an appropriate symbolic form of that shift-register reduction. All experiments were pursued on a 2.5GHz Intel Core-i7 processor with 8GB RAM running 64-bit Ubuntu 17.04.

From the upper part of Table 1, it can be seen that our incremental algorithm significantly outperforms the use of the shift-register reduction. On all cases involving delay, Algorithm 1 is faster than the same underlying explicit-state implementation of safety synthesis employed to the reduction of Lemma 1. The benefits from not resorting to an explicit reduction, instead taking advantage of incrementally generated strategies and on-the-fly pruning of already-uncontrollable branches, are thus obvious. In contrast, the reduction-based approach suffers inevitably from the state-explosion problem: for e.g. `Escp.4×5` under $\delta = 3$, the reduction yields a game graph comprising 29242 states and 107568 transitions.

[2] Both the prototype implementation and the evaluation examples used in this section can be found at `http://lcs.ios.ac.cn/~chenms/tools/DGame.tar.bz2`. We opted for an implementation in Mathematica due to its built-in primitives for visualization.

[3] Available at `https://www.react.uni-saarland.de/tools/safetysynth/`

Benchmark				Reduction + Explicit-State Synthesis						Algorithm 1						
name	$\|S\|$	$\|\rightarrow\|$	$\|\mathcal{U}\|$	$\delta_{\max}$	$\delta=0$	$\delta=1$	$\delta=2$	$\delta=3$	$\delta=4$	$\delta_{\max}$	$\delta=0$	$\delta=1$	$\delta=2$	$\delta=3$	$\delta=4$	%
Exmp. 3	14	20	4	≥ 22	0.00	0.00	0.01	0.02	0.02	$\geq$ **30**	0.00	0.00	**0.00**	**0.01**	**0.01**	
Exmp. 4	14	22	4	$=2$	0.00	0.01	0.01	0.02	–	$=2$	0.00	**0.00**	**0.00**	**0.01**	–	81.97
Escp.4×4	224	738	16	$=2$	0.08	11.66	11.73	1059.23	–	$=2$	0.08	**0.13**	**0.22**	**0.25**	–	99.02
Escp.4×5	360	1326	20	$=2$	0.18	34.09	33.80	3084.58	–	$=2$	0.18	**0.27**	**0.46**	**0.63**	–	99.02
Escp.5×5	598	2301	26	≥ 2	0.46	96.24	97.10	?	?	$=$ **2**	0.46	**0.68**	**1.16**	**1.71**	–	98.98
Escp.5×6	840	3516	30	≥ 2	1.01	217.63	216.83	?	?	$=$ **2**	**1.00**	**1.42**	**2.40**	**4.30**	–	99.00
Escp.6×6	1224	5424	36	≥ 2	2.13	516.92	511.41	?	?	$=$ **2**	**2.06**	**2.90**	**5.12**	**10.30**	–	98.97
Escp.7×7	2350	11097	50	≥ 2	7.81	2167.86	2183.01	?	?	$=$ **2**	**7.71**	**10.67**	**19.04**	**52.47**	–	98.99
Escp.7×8	3024	14820	56	≥ 0	**13.07**	?	?	?	?	$=$ **2**	13.44	**18.25**	**32.69**	**108.60**	–	99.01

Benchmark		Reduction + Yosys + SafetySynth[4] (symbolic)							Algorithm 1 (simple explicit-state implementation)							
name	$\delta_{\max}$	$\delta=0$	$\delta=1$	$\delta=2$	$\delta=3$	$\delta=4$	$\delta=5$	$\delta=6$	$\delta=0$	$\delta=1$	$\delta=2$	$\delta=3$	$\delta=4$	$\delta=5$	$\delta=6$	%
Stub.4×4	$=2$	1.07	1.24	1.24	1.80	–	–	–	**0.04**	**0.07**	**0.12**	**0.18**	–	–	–	98.98
Stub.4×5	$=2$	1.16	1.49	1.49	2.83	–	–	–	**0.08**	**0.14**	**0.25**	**0.44**	–	–	–	98.97
Stub.5×5	$=2$	1.19	2.61	2.50	13.67	–	–	–	**0.21**	**0.37**	**0.63**	**1.17**	–	–	–	98.97
Stub.5×6	$=2$	1.18	2.60	2.59	23.30	–	–	–	**0.42**	**0.69**	**1.20**	**2.49**	–	–	–	98.96
Stub.6×6	$=4$	1.17	2.76	2.74	19.96	19.69	655.24	–	**0.93**	**1.47**	**2.60**	**5.79**	**7.54**	**7.60**	–	99.89
Stub.7×7	$=4$	**1.23**	**2.50**	**2.48**	24.57	**23.01**	2224.62	–	3.60	5.52	10.08	**22.75**	31.18	**32.98**	–	99.88

$\delta_{\max}$: the maximum delay under which G_δ is controllable.
$\delta_{\max}=\delta'$: G_δ is verified controllable under delays $0\leq\delta\leq\delta'$ while uncontrollable under any delay $\delta>\delta'$.
$\delta_{\max}\geq\delta'$: G_δ is verified controllable under delays $0\leq\delta\leq\delta'$ within 1 hour CPU time bound, yet unknown under $\delta>\delta'$ due to the limitation of computing capability.
–: already for smaller δ the controller has no winning strategy.
?: algorithm fails to answer the control/synthesis problem within 1 hour of CPU time.
%: percentage of savings in state space compared to the reduction-based methods, as obtained on $\delta_{\max}+1$.

Table 1: Benchmark results in relation to reduction-based approaches (time in seconds)

Within the lower part of Table 1, the performance of the current explicit-state implementation of Algorithm 1 is compared with that of SafetySynth, the winner in the sequential safety synthesis track of the 3rd and 4th Reactive Synthesis Competition[4] (SYNTCOMP 2016 and 2017). In order to be able to examine the efficiency of our incremental algorithm under larger delays, we used a slight modification of the escape game forbidding the kid to take moves to the right or up, thus increasing the controllability for the robot. Note that Algorithm 1 completes synthesis faster in these "stubborn" scenarios due to the reduced action set. SafetySynth implements a symbolic backward fixed-point algorithm for solving delay-free safety games using the CUDD package. Its input is an extension of the AIGER[5] format known from hardware model-checking and synthesis. We therefore provided symbolic models of the escape games in Verilog[6] and compiled them to AIGER format using Yosys[7]. Verilog supports compact symbolic modelling of the coordinates other than an explicit representation of the game graph as in Fig. 4, and further admits direct use of shift registers for memorizing actions of the robot under delays. Therefore, as visible in Table 1, SafetySynth outperforms our explicit-state safety synthesis for some large room sizes under small delays. For larger delays it is, however, evident that our incremental algorithm always wins, despite its use of non-symbolic encodings.

Remark 2. It would be desirable to pursue a comparison on standard benchmarks like the synthesis track of SYNTCOMP. As these are conveyed in AIGER format only and

[4] http://www.syntcomp.org/ [5] http://fmv.jku.at/aiger/
[6] http://www.verilog.com/ [7] http://www.clifford.at/yosys/

not designed for modifiability, like the introduction of shift registers, this unfortunately is not yet possible. Likewise, other state-of-the-art synthesizers from the SYNTCOMP community, like AbsSynthe [4], could not be used for comparison as they do not support the state initializations appearing in the AIGER translations of the escape game.

5 Conclusions

Designing controllers that work safely and reliably when exposed to delays is a crucial challenge in many application domains, like transportation systems or industrial robots. In this paper, we have used a straightforward, yet exponential reduction to show that the existence of a finite-memory winning strategy for the controller in games with delays is decidable with respect to safety objectives. As such a reduction being exponential in the magnitude of the delay would rapidly become unwieldy, we proposed an algorithm that incrementally synthesizes a series of controllers withstanding increasingly larger delays, thereby interleaving the unavoidable introduction of memory with state-space pruning removing all states no longer controllable under the given delay before proceeding to the next larger delay. To the best of our knowledge, we also provided the first implementation of such a state-space pruning within an algorithm for solving games with delays, and we demonstrated the beneficial effects of this incremental approach on a number of benchmarks.

The benchmarks used were robot escape games indicative of collision avoidance scenarios in, e.g., traffic maneuvers. Control under delay here involves selecting appropriate safe actions or movements without yet knowing the most recent positions of the other traffic participants. Experimental results on such escape games demonstrate that our incremental algorithm outperforms reduction-based safety synthesis, irrespective of whether this safety synthesis employs naïve explicit-state or state-of-the-art symbolic synthesis methods, as available in Saarbrücken's SafetySynth tool.

An extension to hybrid control, dealing with infinite-state game graphs described by hybrid safety games, is currently under development and will be exposed in future work. We are also moving forward to a more efficient implementation of Algorithm 1 based on symbolic encodings, like BDDs [17] or SAT [3]. A further subject of future investigation is stochastic models of out-of-order delivery of messages. As these result in a high likelihood of state information being available before the maximum transportation delay, such models can quantitatively guarantee better controllability than the worst-case scenario of always delivering messages with maximum delay addressed in this paper. We will therefore attack synthesis towards quantitative safety targets in such stochastic settings and may also exploit constructive means of manipulating probability distributions of message delays, like QoS control, within the synthesis.

Acknowledgements. The authors would like to thank Bernd Finkbeiner and Ralf Wimmer for insightful discussions on the AIGER format for synthesis and Leander Tentrup for extending his tool SafetySynth by state initialization, thus facilitating a comparison.

References

1. S. Balemi. Communication delays in connections of input/output discrete event processes. In *CDC 1992*, pages 3374–3379, 1992.

2. G. Behrmann, A. Cougnard, A. David, E. Fleury, K. G. Larsen, and D. Lime. UPPAAL-Tiga: time for playing games! In *CAV 2007*, volume 4590 of *Lecture Notes in Computer Science*, pages 121–125, 2007.
3. R. Bloem, R. Könighofer, and M. Seidl. SAT-based synthesis methods for safety specs. In *VMCAI 2014*, volume 8318 of *Lecture Notes in Computer Science*, pages 1–20, 2014.
4. R. Brenguier, G. A. Pérez, J. Raskin, and O. Sankur. AbsSynthe: abstract synthesis from succinct safety specifications. In *SYNT 2014*, volume 157 of *EPTCS*, pages 100–116, 2014.
5. R. Brenguier, G. A. Pérez, J. Raskin, and O. Sankur. Compositional algorithms for succinct safety games. In *SYNT 2015*, pages 98–111, 2015.
6. J. Büchi and L. Landweber. Solving sequential conditions by finite-state strategies. *Trans. Amer. Math. Soc.*, 138:295–311, 1969.
7. J. R. Büchi and L. H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138(1):295–311, 1969.
8. D. Gale and F. M. Stewart. Infinite games with perfect information. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the theory of games II*, Annals of Mathematics Studies 28, pages 245–266. Princeton University Press, 1953.
9. F. Klein and M. Zimmermann. How much lookahead is needed to win infinite games? In *ICALP 2015*, volume 9135 of *Lecture Notes in Computer Science*, pages 452–463. 2015.
10. F. Klein and M. Zimmermann. What are strategies in delay games? Borel determinacy for games with lookahead. In *CSL 2015*, volume 41 of *Leibniz International Proceedings in Informatics*, pages 519–533, 2015.
11. O. Kupferman and M. Y. Vardi. Synthesis with incomplete information. In *Advances in Temporal Logic*, pages 109–127. Springer, 2000.
12. R. McNaughton. Infinite games played on finite graphs. *Ann. Pure Appl. Logic*, 65(2):149–184, 1993.
13. S. Park and K. Cho. Delay-robust supervisory control of discrete-event systems with bounded communication delays. *IEEE Trans. on Automatic Control*, 51(5):911–915, 2006.
14. A. Pnueli and R. Rosner. On the synthesis of an asynchronous reactive module. In *ICALP 1989*, volume 327 of *Lecture Notes in Computer Science*, pages 652–671, 1989.
15. J. Raskin, K. Chatterjee, L. Doyen, and T. A. Henzinger. Algorithms for omega-regular games with imperfect information. *Logical Methods in Computer Science*, 3(3), 2007.
16. J. H. Reif. The complexity of two-player games of incomplete information. *J. Comput. Syst. Sci.*, 29(2):274–301, 1984.
17. F. Somenzi. Binary decision diagrams. In *Calculational System Design, volume 173 of NATO Science Series F: Computer and Systems Sciences*, pages 303–366. IOS Press, 1999.
18. W. Thomas. On the synthesis of strategies in infinite games. In *STACS 95*, volume 900 of *Lecture Notes in Computer Science*, pages 1–13. 1995.
19. S. Tripakis. Decentralized control of discrete-event systems with bounded or unbounded delay communication. *IEEE Trans. on Automatic Control*, 49(9):1489–1501, 2004.
20. M. D. Wulf, L. Doyen, and J. Raskin. A lattice theory for solving games of imperfect information. In *HSCC 2006*, volume 3927 of *Lecture Notes in Computer Science*, pages 153–168, 2006.
21. M. Zimmermann. Finite-state strategies in delay games. In P. Boyer, A. Orlandini, and P. San Pietro, editors, *8th Symposium on Games, Automata and Formal Verification (GandALF '17)*, volume 256 of *EPTCS*, pages 151–165, 2017.

Appendix A Proofs of Lemma 1 and Theorem 3

Proof (of Lemma 1). We first concentrate on the case of even delay δ. For an even delay δ, game $\widehat{G}$ simulates playing G under delay δ by first forcing the controller to guess an initial action sequence $\alpha \in \Sigma^{\frac{\delta}{2}}$ and then maintains a shift register of player-0 actions. It subsequently determines actions of player 0 by the head of the shift register while appending fresh actions to its tail, thus delaying the effect of player-0 actions by δ steps. As each action that thus comes to effect has been decided δ time units ago, this is equivalent to deciding actions at step i based on the play prefix $\pi(i-\delta)$, as a strategy under delay would have to. Consequently, a winning strategy for the controller in this safety game $\widehat{G}$ exists iff a strategy winning for the controller in G under delay δ exists.

For the case of odd delay $\delta = 2k+1$, we observe that the move from a state at $\pi_{i-\delta-1}$ to $\pi_{i-\delta}$ is under control of the controller if i itself is an even position, i.e., under control of the controller. If playing a deterministic strategy, which obviously is as powerful as playing potentially non-deterministic strategies, the controller consequently cannot gain any additional information from being able to observe the play prefix $\pi(i-\delta)$ rather than just the shorter prefix $\pi(i-\delta-1)$. The problem of finding a strategy under odd delay δ thus is equivalent to that of finding a strategy for even delay $\delta+1$, which warrants using reduction to the same safety game $\widehat{G}$ in both cases. □

Proof (of Theorem 3). Termination is trivially guaranteed by the strictly increasing index k bounded by the final delay of interest δ. For convenience, we define the union of two maps $\hat{\xi}_1, \hat{\xi}_2 : S \times \Sigma^{\lfloor \frac{\delta}{2} \rfloor} \mapsto 2^{\Sigma}$ as $\hat{\xi}_1 \cup \hat{\xi}_2 : S \times \Sigma^{\lfloor \frac{\delta}{2} \rfloor} \mapsto 2^{\Sigma}$ by $(\hat{\xi}_1 \cup \hat{\xi}_2)(s,\alpha) = \hat{\xi}_1(s,\alpha) \cup \hat{\xi}_2(s,\alpha)\ \forall s \in S, \alpha \in \Sigma^{\lfloor \frac{\delta}{2} \rfloor}$. It then follows that if $(\alpha, \hat{\xi}_1)$ and $(\alpha, \hat{\xi}_2)$ are both winning strategies of a game with delay δ, then $(\alpha, \hat{\xi}_1 \cup \hat{\xi}_2)$ is also a winning strategy. This fact allows us to define for any $\alpha \in \Sigma^{\lfloor \frac{\delta}{2} \rfloor}$ the maximally permissive winning strategy as $(\alpha, \cup\hat{\xi})$, where the union is over all such $\hat{\xi}$'s that $(\alpha, \hat{\xi})$ is a winning strategy with delay δ.

Now, we prove that with output $(\texttt{WINNING}, \alpha, \hat{\xi}_\delta)$ the strategy $(\alpha, \hat{\xi}_\delta)$ is actually a maximally permissive winning strategy of game G_δ. We prove by induction on k that during execution of the algorithm, $(\alpha, \hat{\xi}_k)$ is always a maximally permissive winning strategy of the game with delay k. The initial case of $k = 0$ is guaranteed by **FPIteration** in line 2 of Algorithm 1, and the induction from k to $k+1$ is achieved by two steps. First, we prove that $(\alpha, \hat{\xi}_{k+1})$ is a winning strategy. It suffices to prove the fact $\emptyset \neq O(G, \alpha, \hat{\xi}_{k+1}, k+1) \subseteq O(G, \alpha, \hat{\xi}_k, k)$, which is demonstrated in the following two cases:

1. For an even k, the strategy $(\alpha, \hat{\xi}_{k+1})$ is playable, since for any path $\pi = \pi_0\sigma_0\pi_1\sigma_1\ldots$ obtained under $(\alpha, \hat{\xi}_{k+1})$, $\hat{\xi}_{k+1}(\pi_{2i+1}, \sigma_{2i+2}\sigma_{2i+4}\ldots\sigma_{2i+k}) \neq \emptyset$; otherwise, the configuration $(\pi_{2i+1}, \sigma_{2i+2}\ldots\sigma_{2i+k})$ will be removed by the **Shrink** procedure in line 9 and thus cannot be reached under the strategy. Furthermore, the assignment in line 7 implies that for any play $\pi_0\sigma_0\pi_1\sigma_1\ldots \in \Pi(G)$,

   $$\alpha_{2i} \in \hat{\xi}_{k+1}(\pi_{2i-k-1}, \sigma_{2i-k}\sigma_{2i-k+2}\ldots\sigma_{2i-2}) \Rightarrow \alpha_{2i} \in \hat{\xi}_k(\pi_{2i-k}, \sigma_{2i-k}\ldots\sigma_{2i-2}),$$

   thus we have $O(G, \alpha, \hat{\xi}_{k+1}, k+1) \subseteq O(G, \alpha, \hat{\xi}_k, k)$. So, all the outcomes are safe from the induction.

2. For an odd k, playing under $(\alpha, \hat{\xi}_{k+1})$ is as the same as playing under $(\alpha, \hat{\xi}_k)$, since $O(G, \alpha, \xi_{k+1}, k+1) = O(G, \alpha, \xi_k, k)$ can be verified from the assignment in line 19. So, $(\alpha, \hat{\xi}_{k+1})$ should be a winning strategy, as the same as $(\alpha, \hat{\xi}_k)$.

Second, the maximality of $(\alpha, \hat{\xi}_{k+1})$ can be argued by the maximality of $(\alpha, \hat{\xi}_k)$, together with the fact: for any winning strategy $(\alpha, \hat{\xi}'_{k+1})$ for an odd delay $k+1$, a winning strategy $(\alpha', \hat{\xi}'_k)$ for delay k can be constructed by

$$\alpha' = \{\alpha_0 \dots \alpha_{\frac{k}{2}-1} \mid \alpha_0 \dots \alpha_{\frac{k}{2}} \in \alpha\} \text{ and } \hat{\xi}'_k(s', \sigma_1 \dots \sigma_{\frac{k}{2}}) = \hat{\xi}'_{k+1}(s, \sigma_1 \dots \sigma_{\frac{k}{2}}), \quad (1)$$

where $s \xrightarrow{u} s'$, and the maximality of the winning strategy is preserved.

To prove that with output $(\texttt{LOSING}, k)$ the game has no winning strategy even for delay k, we note that k should be an odd number in this case. Suppose that there is a winning strategy $(\alpha, \hat{\xi}'_k)$ for delay k, we can construct a winning strategy $(\alpha', \hat{\xi}'_{k-1})$ for delay $k-1$ as by Eq. (1). It is easy to check that $\hat{\xi}_{k-1} \cup \hat{\xi}'_{k-1} \neq \hat{\xi}_{k-1}$, which contradicts to the maximality of $\hat{\xi}_{k-1}$. □

Appendix B Additional Examples

Example 3. Consider the safety game $G = \langle S, s_0, S_0, S_1, \Sigma, \mathcal{U}, \rightarrow \rangle$, illustrated in Fig. 5, where $S = S_0 \cup S_1$, with $S_0 = \{c_1, c_2, c_3, c_4, c_5, c_6\}$, and $S_1 = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\}$, while $s_0 = c_1$, $\Sigma = \{a, b\}$, and $\mathcal{U} = \{e_1, e_3, e_5, e_7\}$.

Example 4. The game in this example shares the same graph structure as that in Fig. 5, except that we empower the environment there a bit by introducing two fresh transitions $e_4 \xrightarrow{u} c_2$ and $e_4 \xrightarrow{u} c_6$ in G. The winning strategy then vanishes when the delay is lifted to 3.

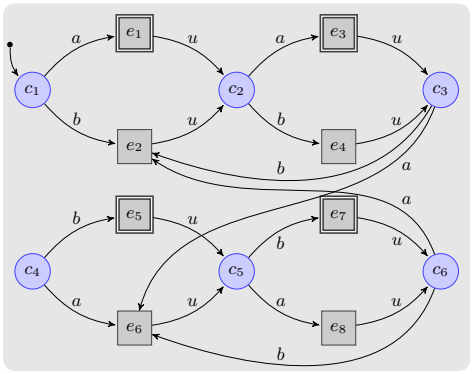

Fig. 5: A safety game winnable for the controller with finite-memory strategies with $\delta_{\max} \geq 30$.

To show the essential feature of a game, the choices of the environment in these examples can be presented distinguishably as e.g. $u, v, w, \dots$, which though are abstracted away —as we do in this paper— when one is interested in synthesizing a winning strategy merely for the controller.

Appendix C Sub-algorithm: FPIteration

Algorithm 3: FPIteration: Generating permissive strategy for delay-free games

input : $G = \langle S, s_0, S_0, S_1, \Sigma, \mathcal{U}, \rightarrow \rangle$, a safety game played under no delay.

`/* initialization */`

$U \leftarrow \mathcal{U}$;
for $s \in S_0$ **do**
  if $s \in \mathcal{U}$ **then**
    $\hat{\xi}_0(s, \varepsilon) \leftarrow \emptyset$;
  else
    $\hat{\xi}_0(s, \varepsilon) \leftarrow \{\sigma \mid \exists s' : s \xrightarrow{\sigma} s'\}$;

`/* computing fixed-point on set of unwinnable states` U `*/`

while $U \neq \emptyset$ **do**
  $s' \leftarrow$ **Pop**(U) ;
  if $s' \in S_0$ **then**
    Push$(U, \{s \mid s \xrightarrow{u} s'\} \setminus U)$;
  else
    for $s : s \xrightarrow{\sigma} s'$ **do**
      $\hat{\xi}_0(s, \varepsilon) \leftarrow \hat{\xi}_0(s, \varepsilon) \setminus \{\sigma\}$;
      if $\hat{\xi}_0(s, \varepsilon) = \emptyset$ and $s \notin U$ **then**
        Push(U, s) ;

return $\hat{\xi}_0$;

Appendix D Constraint Systems $\mathcal{C}$ and $\mathcal{E}$ in `Escp.4×4`

For instance, when the robot takes UR action that leads a play from $(x_0, y_0, x_1, y_1, \text{true})$ to $(x_0', y_0', x_1', y_1', \text{false})$, this action should satisfy the constraint $\mathcal{C}$, which consists of the following four conditions:

- $\mathcal{C}_1 := (x_0' - x_0 = 1) \wedge (y_0' - y_0 = 1) \wedge (x_1' = x_1) \wedge (y_1' = y_1)$, which describes how the robot moves.
- $\mathcal{C}_2 := \neg(y_1 - y_0 = 1 \wedge x_0 = x_1)$, which prohibits the robot from running into the kid, namely the collision with the kid occurs in the first direction of its move.
- $\mathcal{C}_3 := (x_0, y_0 + 1) \notin \mathcal{O}$, which prohibits the robot from running through an obstacle during the first direction of its move (that the endpoint of the move is outside obstacles is taken care off by the safety condition).
- $\mathcal{C}_4 := \forall x_0, y_0, x_0', y_0' : 0 \leq x_0, y_0, x_0', y_0' \leq 3$, which restricts the robot to move inside the room area and avoid running into the walls.

The four conditions are instantiated for each available action and the robot must during the game only choose from legal actions satisfying the corresponding constraint $\mathcal{C}$. An analogous constraint system $\mathcal{E}$ defines the possible actions of the kid.

Appendix E Finite-Memory Winning Strategies in $\texttt{Escp}\,.\,4\times4$

Relabelling states: r for robot, k for kid.
Relabelling actions: $\{\mathtt{a}:\mathtt{UL},\mathtt{b}:\mathtt{LU},\mathtt{c}:\mathtt{UR},\mathtt{d}:\mathtt{RU},\mathtt{e}:\mathtt{DL},\mathtt{f}:\mathtt{LD},\mathtt{g}:\mathtt{DR},\mathtt{h}:\mathtt{RD},\mathtt{i}:\epsilon\}$.

Under $\delta = 0$:

$\hat{\xi}_0 = \{$ $(\text{r0033},\varepsilon)\mapsto\{\mathtt{c},\mathtt{d},\mathtt{i}\}$, $(\text{r1123},\varepsilon)\mapsto\{\mathtt{b},\mathtt{e},\mathtt{f},\mathtt{g},\mathtt{h},\mathtt{i}\}$, $(\text{r1132},\varepsilon)\mapsto\{\mathtt{b},\mathtt{e},\mathtt{f},\mathtt{g},\mathtt{h},\mathtt{i}\}$,
$(\text{r0023},\varepsilon)\mapsto\{\mathtt{i},\mathtt{c},\mathtt{d}\}$, $(\text{r0032},\varepsilon)\mapsto\{\mathtt{i},\mathtt{c},\mathtt{d}\}$, $(\text{r0213},\varepsilon)\mapsto\{\mathtt{g},\mathtt{i}\}$,
$(\text{r0233},\varepsilon)\mapsto\{\mathtt{g},\mathtt{c},\mathtt{i}\}$, $(\text{r0222},\varepsilon)\mapsto\{\mathtt{c},\mathtt{g},\mathtt{i}\}$, $(\text{r2213},\varepsilon)\mapsto\{\mathtt{e},\mathtt{g},\mathtt{h},\mathtt{i}\}$,
$(\text{r2233},\varepsilon)\mapsto\{\mathtt{a},\mathtt{e},\mathtt{g},\mathtt{h},\mathtt{i}\}$, $(\text{r2222},\varepsilon)\mapsto\{\}$, $(\text{r0013},\varepsilon)\mapsto\{\mathtt{c},\mathtt{d},\mathtt{i}\}$,
$(\text{r0022},\varepsilon)\mapsto\{\mathtt{c},\mathtt{d}\}$, $(\text{r2013},\varepsilon)\mapsto\{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{i}\}$, $(\text{r2033},\varepsilon)\mapsto\{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{i}\}$,
$(\text{r2022},\varepsilon)\mapsto\{\mathtt{a},\mathtt{b}\}$, $(\text{r1113},\varepsilon)\mapsto\{\mathtt{i},\mathtt{b},\mathtt{d},\mathtt{e},\mathtt{f},\mathtt{g},\mathtt{h}\}$, $(\text{r1133},\varepsilon)\mapsto\{\mathtt{i},\mathtt{e},\mathtt{f},\mathtt{b},\mathtt{d},\mathtt{g},\mathtt{h}\}$,
$(\text{r1122},\varepsilon)\mapsto\{\mathtt{b},\mathtt{i}\}$, $(\text{r0231},\varepsilon)\mapsto\{\mathtt{c},\mathtt{g},\mathtt{i}\}$, $(\text{r2231},\varepsilon)\mapsto\{\mathtt{a},\mathtt{e},\mathtt{i}\}$, $(\text{r0031},\varepsilon)\mapsto\{\mathtt{c},\mathtt{d}\}$,
$(\text{r2031},\varepsilon)\mapsto\{\mathtt{a},\mathtt{b}\}$, $(\text{r1131},\varepsilon)\mapsto\{\mathtt{b},\mathtt{d},\mathtt{i}\}$, $(\text{r1303},\varepsilon)\mapsto\{\mathtt{h}\}$, $(\text{r1323},\varepsilon)\mapsto\{\mathtt{f}\}$,
$(\text{r1103},\varepsilon)\mapsto\{\mathtt{d},\mathtt{e},\mathtt{f},\mathtt{g},\mathtt{h},\mathtt{i}\}$, $(\text{r0203},\varepsilon)\mapsto\{\mathtt{g}\}$, $(\text{r0223},\varepsilon)\mapsto\{\mathtt{g},\mathtt{i}\}$,
$(\text{r1332},\varepsilon)\mapsto\{\mathtt{f},\mathtt{i}\}$, $(\text{r0232},\varepsilon)\mapsto\{\mathtt{i},\mathtt{g},\mathtt{c}\}$, $(\text{r1321},\varepsilon)\mapsto\{\mathtt{f},\mathtt{i}\}$, $(\text{r1121},\varepsilon)\mapsto\{\mathtt{b}\}$,
$(\text{r0221},\varepsilon)\mapsto\{\mathtt{c},\mathtt{i}\}$, $(\text{r3303},\varepsilon)\mapsto\{\mathtt{e},\mathtt{f},\mathtt{i}\}$, $(\text{r3323},\varepsilon)\mapsto\{\}$, $(\text{r3103},\varepsilon)\mapsto\{\mathtt{a},\mathtt{b},\mathtt{f},\mathtt{i}\}$,
$(\text{r3123},\varepsilon)\mapsto\{\mathtt{f}\}$, $(\text{r2203},\varepsilon)\mapsto\{\mathtt{c},\mathtt{d},\mathtt{e},\mathtt{g},\mathtt{h},\mathtt{i}\}$, $(\text{r2223},\varepsilon)\mapsto\{\mathtt{e}\}$, $(\text{r3332},\varepsilon)\mapsto\{\}$,
$(\text{r3132},\varepsilon)\mapsto\{\mathtt{f}\}$, $(\text{r2232},\varepsilon)\mapsto\{\mathtt{a},\mathtt{e}\}$, $(\text{r3321},\varepsilon)\mapsto\{\}$, $(\text{r3121},\varepsilon)\mapsto\{\}$,
$(\text{r2221},\varepsilon)\mapsto\{\mathtt{a}\}$, $(\text{r0003},\varepsilon)\mapsto\{\mathtt{i},\mathtt{c},\mathtt{d}\}$, $(\text{r0021},\varepsilon)\mapsto\{\}$, $(\text{r2003},\varepsilon)\mapsto\{\mathtt{i},\mathtt{a},\mathtt{b},\mathtt{c}\}$,
$(\text{r2023},\varepsilon)\mapsto\{\mathtt{a},\mathtt{b},\mathtt{i}\}$, $(\text{r2032},\varepsilon)\mapsto\{\mathtt{a},\mathtt{b},\mathtt{i}\}$, $(\text{r2021},\varepsilon)\mapsto\{\}$,
$(\text{r2202},\varepsilon)\mapsto\{\mathtt{a},\mathtt{c},\mathtt{d},\mathtt{e},\mathtt{g},\mathtt{h},\mathtt{i}\}$, $(\text{r1313},\varepsilon)\mapsto\{\}$, $(\text{r1302},\varepsilon)\mapsto\{\mathtt{h},\mathtt{i}\}$,
$(\text{r1333},\varepsilon)\mapsto\{\mathtt{i},\mathtt{f},\mathtt{h}\}$, $(\text{r1322},\varepsilon)\mapsto\{\mathtt{f},\mathtt{i}\}$, $(\text{r0202},\varepsilon)\mapsto\{\}$, $(\text{r0002},\varepsilon)\mapsto\{\mathtt{c},\mathtt{d}\}$,
$(\text{r2002},\varepsilon)\mapsto\{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{i}\}$, $(\text{r1102},\varepsilon)\mapsto\{\mathtt{d},\mathtt{g},\mathtt{h},\mathtt{i}\}$, $(\text{r1331},\varepsilon)\mapsto\{\mathtt{i},\mathtt{f},\mathtt{h}\}$,
$(\text{r0211},\varepsilon)\mapsto\{\mathtt{c},\mathtt{i}\}$, $(\text{r0220},\varepsilon)\mapsto\{\mathtt{c},\mathtt{g},\mathtt{i}\}$, $(\text{r2211},\varepsilon)\mapsto\{\mathtt{a},\mathtt{i}\}$, $(\text{r2220},\varepsilon)\mapsto\{\mathtt{a},\mathtt{e},\mathtt{i}\}$,
$(\text{r1311},\varepsilon)\mapsto\{\mathtt{i},\mathtt{f},\mathtt{h}\}$, $(\text{r1320},\varepsilon)\mapsto\{\mathtt{i},\mathtt{f},\mathtt{h}\}$, $(\text{r0011},\varepsilon)\mapsto\{\}$, $(\text{r0020},\varepsilon)\mapsto\{\mathtt{c},\mathtt{d}\}$,
$(\text{r2011},\varepsilon)\mapsto\{\}$, $(\text{r2020},\varepsilon)\mapsto\{\}$, $(\text{r1111},\varepsilon)\mapsto\{\}$, $(\text{r1120},\varepsilon)\mapsto\{\mathtt{b},\mathtt{d},\mathtt{i}\}$,
$(\text{r3313},\varepsilon)\mapsto\{\mathtt{e},\mathtt{f}\}$, $(\text{r3302},\varepsilon)\mapsto\{\mathtt{i},\mathtt{e},\mathtt{f}\}$, $(\text{r3333},\varepsilon)\mapsto\{\}$, $(\text{r3322},\varepsilon)\mapsto\{\}$,
$(\text{r3113},\varepsilon)\mapsto\{\mathtt{i},\mathtt{a},\mathtt{b},\mathtt{f}\}$, $(\text{r3102},\varepsilon)\mapsto\{\mathtt{i},\mathtt{a},\mathtt{b},\mathtt{f}\}$, $(\text{r3133},\varepsilon)\mapsto\{\mathtt{i},\mathtt{a},\mathtt{b},\mathtt{f}\}$,
$(\text{r3122},\varepsilon)\mapsto\{\}$, $(\text{r3331},\varepsilon)\mapsto\{\mathtt{e},\mathtt{f}\}$, $(\text{r3131},\varepsilon)\mapsto\{\}$, $(\text{r3311},\varepsilon)\mapsto\{\mathtt{e},\mathtt{f}\}$,
$(\text{r3320},\varepsilon)\mapsto\{\mathtt{e},\mathtt{f}\}$, $(\text{r3111},\varepsilon)\mapsto\{\mathtt{a},\mathtt{b}\}$, $(\text{r3120},\varepsilon)\mapsto\{\mathtt{a},\mathtt{b}\}$, $(\text{r1301},\varepsilon)\mapsto\{\mathtt{h},\mathtt{i}\}$,
$(\text{r3301},\varepsilon)\mapsto\{\mathtt{e},\mathtt{f},\mathtt{i}\}$, $(\text{r1101},\varepsilon)\mapsto\{\mathtt{d}\}$, $(\text{r3101},\varepsilon)\mapsto\{\mathtt{a},\mathtt{b},\mathtt{i}\}$,
$(\text{r2201},\varepsilon)\mapsto\{\mathtt{a},\mathtt{c},\mathtt{d},\mathtt{g},\mathtt{h},\mathtt{i}\}$, $(\text{r0201},\varepsilon)\mapsto\{\mathtt{c}\}$, $(\text{r0001},\varepsilon)\mapsto\{\}$, $(\text{r2001},\varepsilon)\mapsto\{\mathtt{c}\}$,
$(\text{r1310},\varepsilon)\mapsto\{\mathtt{f},\mathtt{h},\mathtt{i}\}$, $(\text{r1110},\varepsilon)\mapsto\{\mathtt{b},\mathtt{d}\}$, $(\text{r0210},\varepsilon)\mapsto\{\mathtt{c},\mathtt{i}\}$, $(\text{r3310},\varepsilon)\mapsto\{\mathtt{e},\mathtt{f},\mathtt{i}\}$,
$(\text{r3110},\varepsilon)\mapsto\{\mathtt{a},\mathtt{b},\mathtt{i}\}$, $(\text{r2210},\varepsilon)\mapsto\{\mathtt{a},\mathtt{c},\mathtt{d},\mathtt{g},\mathtt{h},\mathtt{i}\}$, $(\text{r0010},\varepsilon)\mapsto\{\}$, $(\text{r2010},\varepsilon)\mapsto\{\mathtt{c}\}$,
$(\text{r0200},\varepsilon)\mapsto\{\mathtt{c},\mathtt{g},\mathtt{i}\}$, $(\text{r2200},\varepsilon)\mapsto\{\mathtt{a},\mathtt{e},\mathtt{c},\mathtt{d},\mathtt{g},\mathtt{h},\mathtt{i}\}$, $(\text{r1300},\varepsilon)\mapsto\{\mathtt{i},\mathtt{f},\mathtt{h}\}$,
$(\text{r3300},\varepsilon)\mapsto\{\mathtt{i},\mathtt{e},\mathtt{f}\}$, $(\text{r0000},\varepsilon)\mapsto\{\}$, $(\text{r2000},\varepsilon)\mapsto\{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{i}\}$,
$(\text{r1100},\varepsilon)\mapsto\{\mathtt{b},\mathtt{d},\mathtt{g},\mathtt{h},\mathtt{i}\}$, $(\text{r3100},\varepsilon)\mapsto\{\mathtt{i},\mathtt{a},\mathtt{b},\mathtt{f}\}$ $\}$.

Under $\delta = 1$:

$$\hat{\xi}_1 = \{\ (\text{k1133}, \varepsilon) \mapsto \{\texttt{b}, \texttt{e}, \texttt{f}, \texttt{g}, \texttt{h}, \texttt{i}\}, (\text{k0033}, \varepsilon) \mapsto \{\texttt{c}, \texttt{d}, \texttt{i}\}, (\text{k0223}, \varepsilon) \mapsto \{\texttt{g}, \texttt{i}\},$$
$$(\text{k2223}, \varepsilon) \mapsto \{\}, (\text{k0023}, \varepsilon) \mapsto \{\texttt{c}, \texttt{d}\}, (\text{k2023}, \varepsilon) \mapsto \{\texttt{a}, \texttt{b}\}, (\text{k1123}, \varepsilon) \mapsto \{\texttt{b}, \texttt{i}\},$$
$$(\text{k0232}, \varepsilon) \mapsto \{\texttt{c}, \texttt{g}, \texttt{i}\}, (\text{k2232}, \varepsilon) \mapsto \{\}, (\text{k0032}, \varepsilon) \mapsto \{\texttt{c}, \texttt{d}\}, (\text{k2032}, \varepsilon) \mapsto \{\texttt{a}, \texttt{b}\},$$
$$(\text{k1132}, \varepsilon) \mapsto \{\texttt{b}, \texttt{i}\}, (\text{k1313}, \varepsilon) \mapsto \{\}, (\text{k1113}, \varepsilon) \mapsto \{\texttt{e}, \texttt{f}, \texttt{g}, \texttt{h}, \texttt{i}\}, (\text{k0213}, \varepsilon) \mapsto \{\texttt{g}\},$$
$$(\text{k1333}, \varepsilon) \mapsto \{\texttt{f}\}, (\text{k0233}, \varepsilon) \mapsto \{\texttt{g}, \texttt{i}\}, (\text{k1322}, \varepsilon) \mapsto \{\texttt{f}\}, (\text{k1122}, \varepsilon) \mapsto \{\texttt{b}\},$$
$$(\text{k0222}, \varepsilon) \mapsto \{\texttt{i}\}, (\text{k3313}, \varepsilon) \mapsto \{\}, (\text{k3113}, \varepsilon) \mapsto \{\texttt{f}\}, (\text{k2213}, \varepsilon) \mapsto \{\texttt{e}\}, (\text{k3333}, \varepsilon) \mapsto \{\},$$
$$(\text{k3133}, \varepsilon) \mapsto \{\texttt{f}\}, (\text{k2233}, \varepsilon) \mapsto \{\texttt{e}\}, (\text{k3322}, \varepsilon) \mapsto \{\}, (\text{k3122}, \varepsilon) \mapsto \{\}, (\text{k2222}, \varepsilon) \mapsto \{\},$$
$$(\text{k0013}, \varepsilon) \mapsto \{\texttt{c}, \texttt{d}, \texttt{i}\}, (\text{k0022}, \varepsilon) \mapsto \{\}, (\text{k2013}, \varepsilon) \mapsto \{\texttt{a}, \texttt{b}, \texttt{i}\}, (\text{k2033}, \varepsilon) \mapsto \{\texttt{a}, \texttt{b}, \texttt{i}\},$$
$$(\text{k2022}, \varepsilon) \mapsto \{\}, (\text{k1331}, \varepsilon) \mapsto \{\texttt{f}, \texttt{i}\}, (\text{k1131}, \varepsilon) \mapsto \{\texttt{b}\}, (\text{k0231}, \varepsilon) \mapsto \{\texttt{c}, \texttt{i}\},$$
$$(\text{k3331}, \varepsilon) \mapsto \{\}, (\text{k3131}, \varepsilon) \mapsto \{\}, (\text{k2231}, \varepsilon) \mapsto \{\texttt{a}\}, (\text{k0031}, \varepsilon) \mapsto \{\}, (\text{k2031}, \varepsilon) \mapsto \{\},$$
$$(\text{k2203}, \varepsilon) \mapsto \{\texttt{e}, \texttt{g}, \texttt{h}, \texttt{i}\}, (\text{k1303}, \varepsilon) \mapsto \{\}, (\text{k1323}, \varepsilon) \mapsto \{\}, (\text{k0203}, \varepsilon) \mapsto \{\},$$
$$(\text{k0003}, \varepsilon) \mapsto \{\texttt{c}, \texttt{d}\}, (\text{k2003}, \varepsilon) \mapsto \{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{i}\}, (\text{k1103}, \varepsilon) \mapsto \{\texttt{d}, \texttt{g}, \texttt{h}, \texttt{i}\},$$
$$(\text{k1332}, \varepsilon) \mapsto \{\texttt{f}, \texttt{i}\}, (\text{k0221}, \varepsilon) \mapsto \{\texttt{c}, \texttt{i}\}, (\text{k2221}, \varepsilon) \mapsto \{\}, (\text{k1321}, \varepsilon) \mapsto \{\texttt{f}, \texttt{i}\},$$
$$(\text{k0021}, \varepsilon) \mapsto \{\}, (\text{k2021}, \varepsilon) \mapsto \{\}, (\text{k1121}, \varepsilon) \mapsto \{\}, (\text{k3303}, \varepsilon) \mapsto \{\texttt{e}, \texttt{f}\}, (\text{k3323}, \varepsilon) \mapsto \{\},$$
$$(\text{k3103}, \varepsilon) \mapsto \{\texttt{a}, \texttt{b}, \texttt{f}, \texttt{i}\}, (\text{k3123}, \varepsilon) \mapsto \{\}, (\text{k3332}, \varepsilon) \mapsto \{\}, (\text{k3132}, \varepsilon) \mapsto \{\},$$
$$(\text{k3321}, \varepsilon) \mapsto \{\}, (\text{k3121}, \varepsilon) \mapsto \{\}, (\text{k1302}, \varepsilon) \mapsto \{\texttt{h}\}, (\text{k3302}, \varepsilon) \mapsto \{\texttt{e}, \texttt{f}, \texttt{i}\},$$
$$(\text{k1102}, \varepsilon) \mapsto \{\texttt{d}\}, (\text{k3102}, \varepsilon) \mapsto \{\texttt{a}, \texttt{b}, \texttt{i}\}, (\text{k2202}, \varepsilon) \mapsto \{\texttt{c}, \texttt{d}, \texttt{g}, \texttt{h}, \texttt{i}\}, (\text{k0202}, \varepsilon) \mapsto \{\},$$
$$(\text{k0002}, \varepsilon) \mapsto \{\}, (\text{k2002}, \varepsilon) \mapsto \{\texttt{c}\}, (\text{k1311}, \varepsilon) \mapsto \{\texttt{i}\}, (\text{k1111}, \varepsilon) \mapsto \{\}, (\text{k0211}, \varepsilon) \mapsto \{\texttt{c}\},$$
$$(\text{k1320}, \varepsilon) \mapsto \{\texttt{f}, \texttt{i}\}, (\text{k1120}, \varepsilon) \mapsto \{\texttt{b}\}, (\text{k0220}, \varepsilon) \mapsto \{\texttt{c}, \texttt{i}\}, (\text{k3311}, \varepsilon) \mapsto \{\},$$
$$(\text{k3111}, \varepsilon) \mapsto \{\}, (\text{k2211}, \varepsilon) \mapsto \{\texttt{a}\}, (\text{k3320}, \varepsilon) \mapsto \{\}, (\text{k3120}, \varepsilon) \mapsto \{\}, (\text{k2220}, \varepsilon) \mapsto \{\texttt{a}\},$$
$$(\text{k0011}, \varepsilon) \mapsto \{\}, (\text{k0020}, \varepsilon) \mapsto \{\}, (\text{k2011}, \varepsilon) \mapsto \{\}, (\text{k2020}, \varepsilon) \mapsto \{\}, (\text{k0201}, \varepsilon) \mapsto \{\},$$
$$(\text{k2201}, \varepsilon) \mapsto \{\texttt{a}, \texttt{i}\}, (\text{k1301}, \varepsilon) \mapsto \{\texttt{h}, \texttt{i}\}, (\text{k3301}, \varepsilon) \mapsto \{\texttt{e}, \texttt{f}\}, (\text{k0001}, \varepsilon) \mapsto \{\},$$
$$(\text{k2001}, \varepsilon) \mapsto \{\}, (\text{k1101}, \varepsilon) \mapsto \{\}, (\text{k3101}, \varepsilon) \mapsto \{\texttt{a}, \texttt{b}\}, (\text{k0210}, \varepsilon) \mapsto \{\texttt{c}, \texttt{i}\},$$
$$(\text{k2210}, \varepsilon) \mapsto \{\texttt{a}, \texttt{i}\}, (\text{k1310}, \varepsilon) \mapsto \{\texttt{f}, \texttt{h}, \texttt{i}\}, (\text{k0010}, \varepsilon) \mapsto \{\}, (\text{k2010}, \varepsilon) \mapsto \{\},$$
$$(\text{k1110}, \varepsilon) \mapsto \{\}, (\text{k3310}, \varepsilon) \mapsto \{\texttt{e}, \texttt{f}\}, (\text{k3110}, \varepsilon) \mapsto \{\texttt{a}, \texttt{b}\}, (\text{k1300}, \varepsilon) \mapsto \{\texttt{h}, \texttt{i}\},$$
$$(\text{k1100}, \varepsilon) \mapsto \{\texttt{d}\}, (\text{k0200}, \varepsilon) \mapsto \{\texttt{c}\}, (\text{k3300}, \varepsilon) \mapsto \{\texttt{e}, \texttt{f}, \texttt{i}\}, (\text{k3100}, \varepsilon) \mapsto \{\texttt{a}, \texttt{b}, \texttt{i}\},$$
$$(\text{k2200}, \varepsilon) \mapsto \{\texttt{a}, \texttt{c}, \texttt{d}, \texttt{g}, \texttt{h}, \texttt{i}\}, (\text{k0000}, \varepsilon) \mapsto \{\}, (\text{k2000}, \varepsilon) \mapsto \{\texttt{c}\}\ \}.$$

The strategy under $\delta = 2$ is analogous to that under $\delta = 1$, except that an action is prepended in the history sequence (which is previously ε). Thus we omit the detailed strategy here for the sake of space.