Playing Hybrid Games with KeYmaera*

Jan-David Quesel[1] and André Platzer[2]

[1] University of Oldenburg, Department of Computing Science, Germany
quesel@informatik.uni-oldenburg.de
[2] Carnegie Mellon University, Computer Science Department, Pittsburgh, PA, USA
aplatzer@cs.cmu.edu

Abstract. We propose a new logic, called *differential dynamic game logic* (dDGℒ), that adds several game constructs on top of differential dynamic logic (dℒ) so that it can be used for hybrid games. The logic dDGℒ is a conservative extension of dℒ, which we exploit for our implementation of dDGℒ in the theorem prover KeYmaera. We provide rules for extending the dℒ sequent proof calculus to handle the dDGℒ constructs by identifying analogs to operators of dℒ. We have implemented dDGℒ in an extension of KeYmaera and verified a case study in which a robot satisfies a joint safety and liveness objective in a factory automation scenario, in which the factory may perform interfering actions independently.

Keywords: differential dynamic logic, hybrid games, sequent calculus, theorem proving, logics for hybrid systems, factory automation

1 Introduction

One relevant question when analyzing complex physical systems is whether one component is able to meet a given safety requirement no matter what its environment does. Consider an autonomous robot moving around in a robotic factory environment. Global decision planning is infeasible, so the robot has limited knowledge about what the other elements of the factory will decide to do. If there is any probabilistic information about the decisions of agents, stochastic system models can be used for verification [9]. Otherwise, the question can be considered as a game between the component and its environment. The mathematical model for interacting discrete control and continuous evolutions is called hybrid system [8]. The game theoretic extension is called hybrid games.

Hybrid games [12, 14, 4, 1, 15] have two types of actions: discrete jumps, which update the value of a variable instantaneously, and continuous evolutions along solutions of differential equations. Time only passes for the latter action. Hence, hybrid games are a natural extension of timed games [5], which only support clocks with differential equation $x' = 1$ and only allow variables to be reset to 0

* This material is based upon work supported by the German Research Council (DFG) in SFB/TR 14 AVACS, the National Science Foundation under NSF CAREER Award CNS-1054246 and NSF EXPEDITION CNS-0926181, grant no. CNS-0931985, and the Army Research Office under Award No. W911NF-09-1-0273.

B. Gramlich, D. Miller, and U. Sattler (Eds.): IJCAR 2012, LNAI 7364, pp. 439–453, 2012.

and not assigned arbitrarily. Fairly restricted classes of hybrid games have been shown to be decidable (see e.g. [4, 1, 15]), but the general case is undecidable. Tomlin et al. [14] study hybrid games for controller synthesis. They give a numerical algorithm for computing controllable predecessors and thus checking if there is a controller that drives the system into a safe state.

Our approach to hybrid games is based on logic and built on top of differential dynamic logic (d$\mathcal{L}$) [7, 8], which is a dynamic logic [3] for hybrid systems instead of the conventional discrete programs that dynamic logic has originally been invented for. The logic d$\mathcal{L}$ has modal formulas $[\alpha]\phi$ and $\langle\alpha\rangle\phi$ for each hybrid system α. The d$\mathcal{L}$ formula $[\alpha]\phi$ expresses that all states reachable by following hybrid system α satisfy ϕ and $\langle\alpha\rangle\phi$ expresses that at least one state reachable by α satisfies ϕ, where ϕ is an arbitrary d$\mathcal{L}$ formula. The logic d$\mathcal{L}$ is closed under all operators of first-order logic and nesting of modalities.

With these operators, we can express simple games in d$\mathcal{L}$ [8]. For example, when F is a hybrid system describing a factory and R a hybrid system describing a robot, then a formula of the form $[F]\langle R\rangle \mathit{safe}$ can be used to express that, for all behaviors of a factory F, the robot R can choose at least one behavior ensuring safety (represented by some d$\mathcal{L}$ formula *safe*). This is a simple game expressible in d$\mathcal{L}$, but it stops after one round of interactions by the factory player and the robot player. In order to say that the robot is still safe if it reacts appropriately after the factory changed its mind in response to the robot's first choice, we can use the formula $[F]\langle R\rangle(\mathit{safe} \wedge [F]\langle R\rangle \mathit{safe})$. We can do so for any given number of rounds of interactions of F and R, but we typically want to say that the system will be safe for *any* number of interactions of F and R, not just for 2.

In this paper, we propose a logic that can state those properties using several game constructs on top of d$\mathcal{L}$, including repetition operators $(G)^{[*]}$ and $(G)^{\langle*\rangle}$ to say that game G repeats. The difference between both operators is which player decides how often to repeat the game. They decide how often to repeat before the game starts. For example, the dDG$\mathcal{L}$ formula $([F]\langle R\rangle)^{[*]}\mathit{safe}$ expresses that, no matter how often the player responsible for $(\cdot)^{[*]}$ decides to repeat the game $[F]\langle R\rangle$, the state resulting from those alternating choices by F and R is safe.

In order to prove such properties, we lift the induction principles of d$\mathcal{L}$ to dDG$\mathcal{L}$. A dDG$\mathcal{L}$ formula $(G)^{[*]}\phi$ behaves in some ways like the d$\mathcal{L}$ or dDG$\mathcal{L}$ formula $[\alpha^*]\phi$ (where α^* is the hybrid system that repeats α). In both cases, we consider all possible numbers of iterations, because we do not know how often it will be repeated. The dDG$\mathcal{L}$ formula $(G)^{\langle*\rangle}\phi$ has similarities to the d$\mathcal{L}$ formula $\langle\alpha^*\rangle\phi$, since in both cases, we can choose some number of repetitions. Yet, G is a hybrid game, whereas α is a hybrid system. Nevertheless, we show that the induction principles of invariants and variants lift from d$\mathcal{L}$ to dDG$\mathcal{L}$.

We prove that dDG$\mathcal{L}$ is a conservative extension of d$\mathcal{L}$ and, thus, our theorem prover KeYmaera [11] for d$\mathcal{L}$ can be extended such that it can be used to prove hybrid games expressible in dDG$\mathcal{L}$. We develop a proof calculus for the specifics of dDG$\mathcal{L}$ and implement it in KeYmaera. We develop and verify a case study in which a mobile robot satisfies a joint safety and liveness objective in a factory automation scenario, in which the factory may perform interfering actions.

2 Hybrid Programs and dDG$\mathcal{L}$

Syntax. We use the *hybrid program* (HP) notation for hybrid systems. We sketch the syntax of these programs as defined in [7, 8]. The syntax of HPs is shown together with an informal semantics in Tab. 1. The basic terms (called θ in the table) are either rational numbers, real-valued variables or arithmetic expressions (with operators $+, -, \cdot, /$) built from those.

Discrete jumps are modeled by $x := \theta$. Their effect is to assign the value of the term θ to the variable x. Continuous evolutions, on the other hand, are modeled by $x' = \theta \,\&\, \chi$. Here, the variables evolve along the solution of the differential equation (x' denotes the derivative of x w.r.t. to time), without leaving the evolution domain characterized by the formula χ. If there is no evolution domain restriction, i.e., $\chi \equiv true$, we just write $x' = \theta$. Note that $x' = \theta$ can be a system of differential equations.

To test conditions on the program flow the test action $?\chi$ is used. If the formula χ holds in the current state, the action has no effect. Otherwise, it aborts the program execution and the execution is discarded. The nondeterministic choice $\alpha \cup \beta$ expresses alternatives in the behavior of the hybrid system. The sequential composition $\alpha; \beta$ expresses that β starts after α finishes. Nondeterministic repetition α^* says that HP α repeats an arbitrary number of times. These operations can be combined to form any other classical control structure [8].

The assignment $x := *$ nondeterministically assigns a real value to x, thereby expressing unbounded nondeterminism. This nondeterminism can be restricted by combining basic programs. For instance, the idiom $x := *; ?\phi$ assigns any value to x such that the formula ϕ holds.

Based on this program notation for the behavior of hybrid systems, we separately define *hybrid games*. The idea behind our notion of hybrid games is to use operators somewhat similar to those of hybrid programs, but for games on top of full hybrid systems. The particular hybrid games that we consider here are two-player games produced by the following grammar (α is a HP):

$$G ::= [\alpha] \mid \langle\alpha\rangle \mid (G_1 \cap G_2) \mid (G_1 \cup G_2) \mid (G_1 G_2) \mid (G)^{[*]} \mid (G)^{\langle *\rangle}$$

By $\mathcal{G}$, we denote the set of all such hybrid games. The intuition behind these games is as follows. The game is played by two players, which we call *Verifier*

Table 1. Statements of hybrid programs (χ is a first-order formula, α, β are HPs)

Statement	Effect
$\alpha;\ \beta$	sequential composition, performing first α and then β afterwards
$\alpha\ \cup\ \beta$	nondeterministic choice, following either α or β
α^*	nondeterministic repetition, repeating α some $n \geq 0$ times
$x_1 := \theta_1, .., x_n := \theta_n$	simultaneously assign θ_i to variables x_i by a discrete assignment
$x := *$	nondeterministic assignment of an arbitrary real number to x
$(x_1' = \theta_1, \ldots,$ $x_n' = \theta_n \,\&\, \chi)$	continuous evolution of x_i along the differential equation system $x_i' = \theta_i$, restricted to evolution domain χ
$?\chi$	test if formula χ holds at current state, abort otherwise

and *Falsifier* who play by the following rules: In the game $[\alpha]$ Falsifier resolves the nondeterminism whereas in the game $\langle\alpha\rangle$ Verifier is allowed to do so. Observe that our notion of hybrid games is built *on top of full hybrid systems*, that is, every hybrid program α is, by way of $[\alpha]$ or $\langle\alpha\rangle$, directly a hybrid game. The game $(G_1 G_2)$ is the sequential composition of games, where game G_2 is played right after game G_1 has finished. In a game $(G_1 \cap G_2)$ Falsifier may decide whether the game proceeds with G_1 or with G_2. In the game $(G_1 \cup G_2)$ this choice is made by Verifier. Repetitive game playing is possible using the iteration constructs $(G)^{[*]}$ and $(G)^{\langle*\rangle}$, where, for the first one, Falsifier decides how many iterations are played and, for the latter one, Verifier makes the choice. Note that the choice on the number of iterations has to be made by advance notice. That is, the player responsible for controlling the iteration decides how often G is repeated when the game starts and announces it to the other player.

Winning conditions for the games are formulated in dDG$\mathcal{L}$ as postconditions of games. A *strategy* for a player determines how to resolve the nondeterminism under his control based on the result of the game played so far. The nondeterminisms inside a hybrid system are resolved by choosing which real values to assign to x when executing $x := *$ statements, which branch to follow for choices $\cup$, the number of loop iterations, and how long to follow continuous flows.

The dDG$\mathcal{L}$*-formulas* are first-order formulas over the reals extended by hybrid games. They are defined by the following grammar (θ_i are terms, x is a variable, $\sim \in \{<, \leq, =, \geq, >, \neq\}$, ϕ and ψ are formulas, and G is a hybrid game):

$$\phi ::= \theta_1 \sim \theta_2 \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \rightarrow \psi \mid \phi \leftrightarrow \psi \mid \forall x\, \phi \mid \exists x\, \phi \mid G\, \phi$$

A dDG$\mathcal{L}$ formula $G\,\phi$ is valid if Verifier has a strategy to ensure that ϕ holds after playing the game G. Therefore, the goal of Verifier is to make ϕ true while that of Falsifier is complementary, i.e., to make ϕ false. Note that the formula ϕ itself might contain another game.

Consider the dDG$\mathcal{L}$ formula $([\alpha])^{\langle*\rangle}\phi$ which expresses that there is a number of repetitions n, such that the formula ϕ holds after n repetitions of α. Note that this dDG$\mathcal{L}$ formulas is not equivalent to $[\alpha^*]\,\phi$, which would demand that it holds for all possible numbers of executions of α. It is also not equivalent to $\langle\alpha^*\rangle\,\phi$ as this would give control to Verifier over the (possibly unbounded) nondeterminism during the executions of α. A similar observation can be made for $(\langle\alpha\rangle)^{[*]}\phi$ which says that the program α is always able to ensure ϕ by appropriate choices of the nondeterminisms in α. Combining the repetition operator and the choice operators we can express properties like $(\langle\beta\rangle\,[\alpha]\cup[\alpha]\,\langle\beta\rangle)^{[*]}\phi$. This formula means that ϕ holds after any number of iterations (as Falsifier has control over the number of iterations) while Verifier can control (by $\cup$) for each iteration if he wants to move first according to β or Falsifier has to move first according to α.

Semantics. Next, we define the semantics of dDG$\mathcal{L}$. For a set of variables V, denote by $Sta(V)$ the set of states, i.e., all mappings of type $V \rightarrow \mathbb{R}$. Let $val(\nu, \theta)$ denote the valuation of a term θ in a state ν.

Definition 1 (Transitions of hybrid programs). *The* transition relation, $\rho(\alpha)$, of a HP α, *specifies which state ω is reachable from a state ν by operations of the hybrid system α and is defined as follows*

1. $(\nu,\omega) \in \rho(x_1 := \theta_1, \ldots, x_n := \theta_n)$ *iff* $\nu[x_1 \mapsto val(\nu,\theta_1)] \ldots [x_n \mapsto val(\nu,\theta_n)]$ *equals state ω. Particularly, the value of other variables $z \notin \{x_1, \ldots, x_n\}$ remains constant, i.e., $val(\nu, z) = val(\omega, z)$.*
2. $(\nu,\omega) \in \rho(x := *)$ *iff state ω is identical to ν except for the value of x, which can be any real number (could be identical to the previous valuation of x).*
3. $(\nu,\omega) \in \rho(x_1' = \theta_1, \ldots, x_n' = \theta_n \,\&\, \chi)$ *iff there is a continuous function $f : [0,r] \to Sta(V)$ from $f(0) = \nu$ to $f(r) = \omega$, which solves the system of differential equations, i.e., for all $i \in [1,n]$, $val(f(\zeta), x_i)$ has a derivative of value $val(f(\zeta), \theta_i)$ at each time $\zeta \in (0,r)$. Other variables remain constant: $val(f(\zeta), y) = val(\nu, y)$ for $y \neq x_i$, for all $i \in [1,n]$ and $\zeta \in [0,r]$. And the evolution domain χ is respected: $val(f(\zeta), \chi) = true$ for each $\zeta \in [0,r]$.*
4. $\rho(?\chi) = \{(\nu,\nu) \;:\; \nu \models \chi\}$ *where $\nu \models \chi$ is defined as in first-order logic.*
5. $\rho(\alpha \cup \beta) = \rho(\alpha) \cup \rho(\beta)$
6. $\rho(\alpha;\beta) = \rho(\alpha) \circ \rho(\beta) = \{(\nu,\omega) : (\nu,z) \in \rho(\alpha), (z,\omega) \in \rho(\beta)$ *for some state* $z\}$
7. $(\nu,\omega) \in \rho(\alpha^*)$ *iff there are $n \in \mathbb{N}$ and $\nu{=}\nu_0, \ldots, \nu_n{=}\omega$ with $(\nu_i, \nu_{i+1}) \in \rho(\alpha)$ for all $0 \leq i < n$.*

The semantics of formulas of dDG$\mathcal{L}$ is defined as follows.

Definition 2 (Interpretation of dDG$\mathcal{L}$ formulas). *The interpretation $\models$ of a* dDG$\mathcal{L}$ *formula w.r.t. state ν uses the standard meaning of first-order logic:*

1. $\nu \models \theta_1 \sim \theta_2$ *iff* $val(\nu,\theta_1) \sim val(\nu,\theta_2)$ *for* $\sim \in \{=, \leq, <, \geq, >\}$
2. $\nu \models \phi \wedge \psi$ *iff $\nu \models \phi$ and $\nu \models \psi$, accordingly for $\neg, \vee, \rightarrow, \leftrightarrow$*
3. $\nu \models \forall x\, \phi$ *iff $\omega \models \phi$ for all ω that agree with ν except for the value of x*
4. $\nu \models \exists x\, \phi$ *iff $\omega \models \phi$ for some ω that agrees with ν except for the value of x*

Statements about hybrid games G and programs α have the following semantics

5. $\nu \models [\alpha]\phi$ *iff $\omega \models \phi$ for all ω with $(\nu,\omega) \in \rho(\alpha)$,*
6. $\nu \models \langle\alpha\rangle\phi$ *iff $\omega \models \phi$ for some ω with $(\nu,\omega) \in \rho(\alpha)$,*
7. $\nu \models (G_1 \cup G_2)\phi$ *iff $\nu \models G_1\phi$ or $\nu \models G_2\phi$,*
8. $\nu \models (G_1 \cap G_2)\phi$ *iff $\nu \models G_1\phi$ and $\nu \models G_2\phi$,*
9. $\nu \models (G_1 G_2)\phi$ *iff $\nu \models G_1(G_2\phi)$,*
10. $\nu \models (G)^{[*]}\phi$ *iff $\nu \models (G^n)\phi$ holds for all $n \in \mathbb{N}$,*
11. $\nu \models (G)^{\langle * \rangle}\phi$ *iff $\nu \models (G^n)\phi$ holds for some $n \in \mathbb{N}$,*

where G^n denotes the n-times sequential composition of G and $G^0\phi \equiv \phi$.
A formula ϕ is valid (denoted by $\models \phi$) iff $\nu \models \phi$ holds for all states $\nu \in Sta(V)$.

These definitions are abstract. They do not refer to how games are played. Therefore, we now provide a structural operational semantics for games, formally define the notions of play and strategy, and then prove that the existence of a winning strategy for Verifier coincides with the notion of satisfaction in Def. 2. For a game

$$\text{(F1)}\frac{(\nu,\omega)\in\rho(\alpha)}{[\alpha]@\nu\to *@\omega}\quad \text{(F2)}\frac{\rho(\alpha)=\emptyset}{[\alpha]@\nu\to \top@\nu}\quad \text{(F3)}\frac{G@\nu\to G'@\omega}{G\cap H@\nu\to G'@\omega}$$

$$\text{(F4)}\frac{G\cap H@\nu\to G'@\omega}{H\cap G@\nu\to G'@\omega}\quad \text{(F5)}\frac{n\in\mathbb{N}}{(G)^{[*]}@\nu\to G^n@\nu}$$

$$\text{(V1)}\frac{(\nu,\omega)\in\rho(\alpha)}{\langle\alpha\rangle@\nu\to *@\omega}\quad \text{(V2)}\frac{\rho(\alpha)=\emptyset}{\langle\alpha\rangle@\nu\to \bot@\nu}\quad \text{(V3)}\frac{G@\nu\to G'@\omega}{G\cup H@\nu\to G'@\omega}$$

$$\text{(V4)}\frac{G\cup H@\nu\to G'@\omega}{H\cup G@\nu\to G'@\omega}\quad \text{(V5)}\frac{n\in\mathbb{N}}{(G)^{\langle*\rangle}@\nu\to G^n@\nu}$$

$$\text{(S1)}\frac{G@\nu\to *@\omega}{(G\ H)@\nu\to H@\omega}\quad \text{(S2)}\frac{G@\nu\to \bot@\omega}{(G\ H)@\nu\to \bot@\omega}\quad \text{(S3)}\frac{G@\nu\to \top@\omega}{(G\ H)@\nu\to \top@\omega}$$

Fig. 1. Structural Operational Semantics of Hybrid Games (Verifier can only control V and S rules and Falsifier can only control F and S rules)

G and a state ν, we use $G@\nu$ to denote that the game is in the position where starting from state ν the game will follow the transitions of G.

The operational semantics for the games is structured into three types of actions: those controllable by Falsifier (prefixed with F), those controllable by Verifier (prefixed with V), and those for modelling sequential composition (prefixed with S). We add special games $*$, $\top$, and $\bot$ to denote the possible outcomes of a game. In the latter two cases either of the players was unable to make another move. The game terminates in $*$ after the players played all their actions.

Definition 3 (Structural Operational Semantics of Games). *For a game G its operational semantics $[\![G]\!]$ is given by the rules defined in Fig. 1. Here, G^n denotes the n-times sequential composition of G. The semantics provides a relation between game positions, i.e. $[\![G]\!]\subseteq\mathcal{G}\times Sta(V)\times\mathcal{G}\times Sta(V)$.*

Following the rules of the structural operational semantics defines a transition system that is possibly uncountably branching, due to the non-determinism in hybrid programs, e.g. in choosing evolution times. Observe that each path is of finite length, because the number of iterations is chosen non-deterministically but a priori. Note this semantics does not yet define who decides which options to follow. In particular, the structural operational semantics of $\cap$ and $\cup$ is still the same and that of $(\cdot)^{[*]}$ and $(\cdot)^{\langle*\rangle}$ is still the same, but they will differ as soon as we define which player gets to choose. The Verifier can choose V rules and the Falsifier can choose F rules. The S rules are determined anyway.

To determine whether a strategy is compatible with a game, we define the closure of games under a subgame relation. This closure gives the set of all game positions (ignoring the state of system variables) that can occur in a play.

Definition 4 (Closure under Subgame). *For a game G its closure under subgame, $cl(G)$, is defined inductively as:*

- $cl([\alpha])=\{[\alpha]\}$ *and* $cl(\langle\alpha\rangle)=\{\langle\alpha\rangle\}$
- $cl(G_1G_2)=\{G_1G_2\}\cup cl(G_1)\cup cl(G_2)$

- $cl(G_1 \cup G_2) = \{G_1 \cup G_2\} \cup cl(G_1) \cup cl(G_2)$
- $cl(G_1 \cap G_2) = \{G_1 \cap G_2\} \cup cl(G_1) \cup cl(G_2)$
- $cl((G)^{[*]}) = \{(G)^{[*]}\} \cup \bigcup_{n\in\mathbb{N}} cl(G^n)$ *and* $cl((G)^{\langle * \rangle}) = \{(G)^{\langle * \rangle}\} \cup \bigcup_{n\in\mathbb{N}} cl(G^n)$

Definition 5 (Strategy). *A strategy* $s : \mathcal{G} \times Sta(V) \to (\mathcal{G} \cup \{*, \top, \bot\}) \times Sta(V)$ *is a mapping between game positions. A strategy* s *is called compatible with a game* G *if its actions are allowed, i.e.,* $((g@\nu) \to s(g@\nu)) \in [\![G]\!]$ *for all* $g \in cl(G)$ *and for all* $\nu \in Sta(V)$.

Using this notion of strategy, we now formalize the rules of the game by determining which player gets to choose from the actions of the operational semantics.

Definition 6 (Play). *Given a game* $G \in \mathcal{G}$*, a state* $\nu \in Sta(V)$*, and two compatible strategies (one for Falsifier* f *and one for Verifier* v*), a play* $p_{f,v}(G@\nu)$ *is defined by the following algorithm:*

while $G \notin \{*, \bot, \top\}$ do
  Match the form of G:
    Case $[\alpha]$, $G_1 \cap G_2$, or $(G_1)^{[*]}$ then $G@\nu := f(G@\nu)$ *// Falsifier chooses*
    Case $\langle\alpha\rangle$, $G_1 \cup G_2$, or $(G_1)^{\langle * \rangle}$ then $G@\nu := v(G@\nu)$ *// Verifier chooses*
    Case G_1G_2 then do
      $G@\nu := p_{f,v}(G_1@\nu)$ *// play* G_1
      If $G = *$ then $G := G_2$ *// if* G_1 *terminated with* $*$ *move to* G_2
    od
od *// the result is* $G@\nu$ *with* $G \in \{*, \bot, \top\}$

Definition 7 (Winning). *Given a game* G *and a* dDGℒ *formula* ϕ *as winning condition. For an initial state* ν*, the game* G *is won by Verifier iff* G *ends in a position* $H@\omega$ *where either* $H = *$ *and* $\omega \models \phi$*, or* $H = \top$*. Otherwise, Falsifier wins (i.e. the game is zero-sum). For a* dDGℒ*-formula* ϕ *a strategy* s *is called winning in a game* G *if, by applying this strategy, Falsifier (resp. Verifier) wins every play of* G *regardless of which strategy Verifier (resp. Falsifier) follows.*

Lemma 1. *For a state* ν*,* $\nu \models G\phi$ *iff Verifier has a strategy in the game* G *with winning condition* ϕ *started in position* $G@\nu$ *such that he wins the game.*

The proof of Lemma 1 can be found in [13].

Corollary 1. *The formula* $G\phi$ *is valid iff Verifier has a winning strategy in the game* G *for the winning condition* ϕ.

A crucial point for the design of dDGℒ is that we want it to be conservative with respect to differential dynamic logic in the sense that all dℒ formulas are dDGℒ formulas and that any dℒ formula is valid in the semantics of dDGℒ if and only it was valid in the original semantics for dℒ. This allows us to transfer soundness results for proof calculus rules from dℒ to dDGℒ and extend our theorem prover KeYmaera with additional proof rules for handling the extra dDGℒ constructs.

Theorem 1 (Conservative Extension). *Differential dynamic game logic is a conservative extension of differential dynamic logic (A proof is in [13]).*

3 Proof Rules for dDG$\mathcal{L}$

In this section, we present a sound but incomplete proof calculus for dDG$\mathcal{L}$. It is incomplete, because hybrid systems are not semidecidable [8]. The calculus symbolically executes the hybrid games and hybrid programs. Thereby the dDG$\mathcal{L}$ calculus reduces properties of hybrid games to d$\mathcal{L}$ properties of hybrid programs, which it, in turn, reduces to validity questions of formulas in first-order logic over the reals like the d$\mathcal{L}$ proof calculus does [7].

Substitutions are defined only on first-order formulas. Therefore, state changes performed by assignments and continuous evolutions are kept as a simultaneous assignment $\mathcal{J}$ of the form $x_1 := \theta_1, \ldots, x_n := \theta_n$. When the formula was successfully transformed into a first-order one, these jump sets are applied as substitutions. See [7, 8] for more details on this matter. We denote by $\forall^G \phi$ the universal closure of the formula ϕ w.r.t. the variables changed within the game G. The sequent $\Gamma \vdash \Delta$ is an abbreviation for $\bigwedge_{\phi\in\Gamma} \phi \rightarrow \bigvee_{\psi\in\Delta} \psi$. Proof rules are applied from the desired conclusion (goal below bar) to the resulting premises (above bar) that need to be proved instead.

Definition 8 (Rules [8]). *Calculus rules are defined from the rule schemata presented in Fig. 2 using the following definitions:*

1. *If*
$$\frac{\phi_1 \vdash \psi_1 \quad \ldots \quad \phi_n \vdash \psi_n}{\phi_0 \vdash \psi_0}$$
is an instance of a rule schema in Fig. 2, then
$$\frac{\Gamma, \langle\mathcal{J}\rangle \phi_1 \vdash \langle\mathcal{J}\rangle \psi_1, \Delta \quad \ldots \quad \Gamma, \langle\mathcal{J}\rangle \phi_n \vdash \langle\mathcal{J}\rangle \psi_n, \Delta}{\Gamma, \langle\mathcal{J}\rangle \phi_0 \vdash \langle\mathcal{J}\rangle \psi_0, \Delta}$$
can be applied as a proof rule, where Γ, Δ are arbitrary (possible empty) finite sets of context formulas and $\mathcal{J}$ is a (possibly empty) discrete jump set.
2. *Symmetric schemata*
$$\frac{\phi}{\psi}$$
can be applied on either side of the sequent as
$$\frac{\Gamma, \langle\mathcal{J}\rangle \phi \vdash \Delta}{\Gamma, \langle\mathcal{J}\rangle \psi \vdash \Delta} \quad \textit{or as} \quad \frac{\Gamma \vdash \langle\mathcal{J}\rangle \phi, \Delta}{\Gamma \vdash \langle\mathcal{J}\rangle \psi, \Delta}$$
Again they do not alter the context. Additionally, we use the abbreviation $\langle\!\langle\alpha\rangle\!\rangle$ if the rule is independent of the player controlling the action α.
3. *The existential quantifier elimination rule applies to all goals containing variable X at once: If $\phi_1 \vdash \psi_1, \ldots, \phi_n \vdash \psi_n$ is the list of all open goals (i.e., goals that have not been proved yet) of the proof that contain the free variable X, then the following instance can be applied as a proof rule:*
$$\frac{\vdash QE(\exists X \bigwedge_i (\phi_i \vdash \psi_i))}{\phi_1 \vdash \psi_1 \quad \ldots \quad \phi_n \vdash \psi_n}$$

$$(\text{D1})\frac{\phi \wedge \psi}{\langle ?\phi \rangle \psi}\quad(\text{D3})\frac{\langle\!\langle \alpha \rangle\!\rangle \langle\!\langle \beta \rangle\!\rangle \phi}{\langle\!\langle \alpha;\beta \rangle\!\rangle \phi}\quad(\text{D5})\frac{[\alpha]\,\phi \wedge [\beta]\,\phi}{[\alpha \cup \beta]\,\phi}\quad(\text{D7})\frac{\phi \wedge [\alpha;\alpha^*]\,\phi}{[\alpha^*]\,\phi}$$

$$(\text{D2})\frac{\phi \rightarrow \psi}{[?\phi]\,\psi}\quad(\text{D4})\frac{\langle\alpha\rangle\,\phi \vee \langle\beta\rangle\,\phi}{\langle\alpha \cup \beta\rangle\,\phi}\quad(\text{D6})\frac{\phi \vee \langle\alpha;\alpha^*\rangle\,\phi}{\langle\alpha^*\rangle\,\phi}\quad(\text{D8})\frac{\phi^{\theta_1\ldots\theta_n}_{x_1\ldots x_n}}{\langle\!\langle x_1 := \theta_1, \ldots, x_n := \theta_n \rangle\!\rangle\,\phi}$$

$$(\text{D9})\frac{\forall t [x := t]\phi}{[x := *]\,\phi}\quad(\text{D11})\frac{\exists\, t \geq 0\, \forall\, 0 \leq \tilde{t} \leq t\, \langle x := y_v(\tilde{t})\rangle\, \chi \rightarrow \langle x := y_v(t)\rangle\, \phi}{\langle x' = \theta \& \chi\rangle\, \phi}$$

$$(\text{D10})\frac{\exists t [x := t]\phi}{\langle x := *\rangle\,\phi}\quad(\text{D12})\frac{\forall\, t \geq 0\, \forall\, 0 \leq \tilde{t} \leq t\, [x := y_v(\tilde{t})]\, \chi \rightarrow [x := y_v(t)]\, \phi}{[x' = \theta \& \chi]\, \phi}$$

$$(\text{D13})\frac{\vdash \forall^{[\alpha]}(\phi \rightarrow [\alpha]\,\phi)}{\phi \vdash [\alpha^*]\,\phi}\quad(\text{D14})\frac{\vdash \forall^{\langle\alpha\rangle}\forall n > 0(\varphi(n) \rightarrow \langle\alpha\rangle\,\varphi(n-1))}{\exists n \varphi(n) \vdash \langle\alpha^*\rangle\,\exists n(n \leq 0 \wedge \varphi(n))}$$

$$(\text{D15})\frac{\vdash \phi(s(X_1, \ldots, X_n))}{\vdash \forall x \phi(x)}\quad(\text{D17})\frac{\vdash \phi(X)}{\vdash \exists x \phi(x)}$$

$$(\text{D16})\frac{\phi(s(X_1, \ldots, X_n)) \vdash}{\exists x \phi(x) \vdash}\quad(\text{D18})\frac{\vdash \mathrm{QE}(\exists X \bigwedge_i (\phi_i \vdash \psi_i))}{\phi_1 \vdash \psi_1 \quad \ldots \quad \phi_n \vdash \psi_n}$$

$$(\text{D19})\frac{\phi(X) \vdash}{\forall x \phi(x) \vdash}\quad(\text{D20})\frac{\vdash \mathrm{QE}(\forall X \phi(X) \vdash \psi(X))}{\phi(s(X_1, \ldots, X_n)) \vdash \psi(s(X_1, \ldots, X_n))}$$

$$(\text{G1})\frac{G_1\phi \vee G_2\phi}{(G_1 \cup G_2)\phi}\quad(\text{G3})\frac{G_1(G_2\phi)}{(G_1G_2)\phi}\quad(\text{G5})\frac{\phi \vee G\ (G)^{\langle * \rangle}\phi}{(G)^{\langle * \rangle}\phi}$$

$$(\text{G2})\frac{G_1\phi \wedge G_2\phi}{(G_1 \cap G_2)\phi}\quad(\text{G4})\frac{\phi \wedge G\ (G)^{[*]}\phi}{(G)^{[*]}\phi}\quad(\text{G6})\frac{\vdash \forall^G(\phi \rightarrow \psi)}{G\phi \vdash G\psi}$$

$$(\text{G7})\frac{\vdash \forall^G(\phi \rightarrow G\phi)}{\phi \vdash (G)^{[*]}\phi}\quad(\text{G8})\frac{\vdash \forall^G \forall n > 0(\varphi(n) \rightarrow G\ (\varphi(n-1)))}{\exists n \varphi(n) \vdash (G)^{\langle * \rangle}\exists n(n \leq 0 \wedge \varphi(n))}$$

- t and $\tilde{t}$ are fresh logical variables, y_v is the solution of the symbolic initial value problem $(\dot{x} = \theta, x(0) = v)$.
- Logical variable n does neither occur in α nor G.
- $\phi^{\theta_1\ldots\theta_n}_{x_1\ldots x_n}$ denotes the formula where each x_i is substituted by θ_i simultaneously. The assignment in rule D8 must be admissible [8], otherwise it is added to the jump context $\langle \mathcal{J} \rangle$.
- X is a new logical variable.
- QE: quantifier elimination procedure (can only be applied to first-order formulas).
- For D18 $\phi_i \vdash \psi_i$ are the only branches where X occurs as a free logical variable.

Fig. 2. Free-variable proof calculus for dDG$\mathcal{L}$

Figure 2 shows a proof calculus for dDG$\mathcal{L}$. Together with rules for dealing with propositional logic (including a cut rule), the calculus rules D1-D20 form the original proof calculus for d$\mathcal{L}$ [7, 8]. To handle the new game constructs appearing in dDG$\mathcal{L}$ the rules G1-G8 are used. The rules D1-D12 are equivalences for transforming and decomposing hybrid programs. The rules D13 (resp. D14) allow reasoning about loops using induction (resp. proving convergence).

For handling first-order quantifiers, we use rules D15-D20 from d$\mathcal{L}$ [7]. They perform Skolemization [8] to allow for removing the modalities within the formulas using other rules. After the modalities are dealt with, the quantifiers are reintroduced and quantifier elimination (QE) is performed.

The rules G1 and G2 are equivalences to reason about the choice operations on games. They are the game-equivalents of D5 and D4. Rule G3 transforms sequential compositions such that they can be handled by the original rules of the $\mathsf{d}\mathcal{L}$ calculus. It takes the form of D3. The rules G4-G5 allow for unwinding of the game loops. Rule G7 follows the pattern of D13, but allows induction over game loops that are under Falsifier's control. If Verifier can establish that a formula ϕ holds after any run of game G that started in an arbitrary state satisfying ϕ, then, by induction, ϕ holds for an arbitrary number of plays. Rule G8 follows the pattern of D14 and can be used to show properties of game loops that are under Verifier's control. We can be sure that there is a number of iterations after which the postcondition $\varphi(n)$ holds for some $n \leq 0$ if G can be controlled by Verifier such that the state converges w.r.t. $\varphi(n)$. Here, the existence of some n such that $\varphi(n)$ holds serves as an induction anchor. As for each play started in an arbitrary state where $n > 0$ and $\varphi(n)$ holds, Verifier can assure that after playing the game G the formula $\varphi(n-1)$ holds, thus the game can be forced to eventually reach a state where $n \leq 0$ and $\varphi(n)$ holds. Note that n must not occur in the game G as otherwise it would be bound by the game instead of the quantifier prefix in the postcondition and thus falsify our induction. Additionally, the generalization rule G6 can be used to strengthen postconditions. This rule can, for example, be used to add induction anchors and use cases to the rules G7 and G8.

The purpose of the calculus is to provide a framework for deriving valid $\mathsf{dDG}\mathcal{L}$ formulas syntactically. A calculus is sound iff all formulas derived by applying the calculus rules are indeed valid.

Definition 9 (Soundness). *A calculus rule* $\frac{\phi_1,\ldots,\phi_n}{\psi_1,\ldots,\psi_n}$ *is sound iff validity of the premises* $\phi_1 \wedge \cdots \wedge \phi_n$ *implies the validity of the conclusions* $\psi_1 \wedge \cdots \wedge \psi_n$.

Theorem 2. *The* $\mathsf{dDG}\mathcal{L}$ *calculus rules presented in Fig. 2 are sound.*

The soundness proofs for the rules D1-D20 in [8] are valid for $\mathsf{dDG}\mathcal{L}$ as well, because $\mathsf{dDG}\mathcal{L}$ is a conservative extension of $\mathsf{d}\mathcal{L}$ (see Theorem 1).

The soundness of the rules G1 and G2 is obvious from the semantics of the operators $\cup$ and $\cap$ on games. For the rule G3 the soundness follows directly from the definition of the sequential composition. The soundness of the unwinding rules G4 (resp. G5) is a direct consequence of the semantics of $(G)^{[*]}$ (resp. $(G)^{\langle * \rangle}$). For proving soundness of rule G6 a similar pattern to that in [8] can be applied. The game G can only change the variables that occur in G. Therefore, if $\phi \rightarrow \psi$ and $G\phi$ holds independent of how the variables occurring in G are evaluated, ψ also holds after playing G.

Soundness of the induction rule G7 and the convergence rule G8 can be shown by induction over the number of executions of the loop, in analogy to the soundness proofs for D13 and D14. The proof of Theorem 2 can be found in [13].

4 Case Study: Robotic Factory Automation

To demonstrate the applicability of our approach we model a factory automation scenario in which an autonomous robot moves in an automatic factory. For

scalability reasons, central coordination and planning become infeasible, so the factory is set up as a collection of autonomous agents pursuing goals that may not be known globally. The robot has a secondary objective of reaching certain target positions, but its primary objective is to stay safe, i.e., neither leave the factory site nor bump into its surrounding wall, which could damage the robot.

Model. We model a robot with position (x, y), velocity $\boldsymbol{v} = (v_x, v_y)$, and acceleration $\boldsymbol{a} = (a_x, a_y)$ on a 2 dimensional rectangular factory ground (Fig. 3). There are two conveyor belts. One pointing in x-direction and one pointing in y-direction. The factory may independently decide to activate the conveyor belts, in which case they increase the velocity of the robot. The robot may decide to move in any direction. Therefore, it can decelerate to try to compensate for this increased speed. The goal of the robot is to avoid crashing into any wall and avoid other machines using the belt.

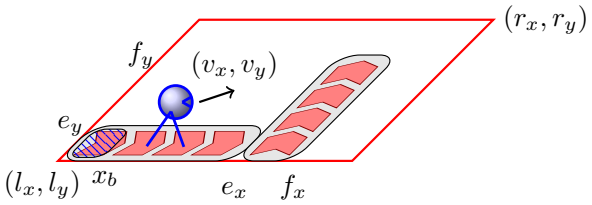

Fig. 3. Sketch of the robotic factory automation site

A sketch of the factory site is provided in Fig. 3. One conveyor belt is of y-width e_y between positions l_x and e_x and moves in x-direction if activated. Between e_x and f_x there is a belt of y-width f_y moving in y-direction. The shaded region in Fig. 3 indicates a region that has to be cleared within ε time units after the system was started, because other robotic elements of the factory may occupy this space then and not watch out for our robot. For simplicity, the robot is initially located at the lower left end (l_x, l_y) of the factory site. The conveyor belt in x-direction has a maximal velocity of c_x and that in y-direction of c_y. The conveyor belts accelerate very quickly, so we simply consider them to accelerate instantaneously. Thus, upon activation, their effect is to increase the velocity of the robot by a discrete assignment instantaneously if the robot is currently located on the conveyor belt that got activated. The robot itself can accelerate with any acceleration of absolute value at most $A = 2$ and that acceleration can be applied in x-direction (acceleration a_x) or y-direction (acceleration a_y) or both. The robot can activate a brake that will slow it down. The difference between braking and just accelerating in the opposite direction is that braking does not allow changes of the sign of the velocity but instead stops at velocity 0.

Specification. As the robot is a moving object and cannot come to a standstill instantaneously, certain conditions have to be satisfied to allow safe operation. Therefore, we assume the following conditions on the scenario. We require that the point x_b can be reached by accelerating for at most time ε, the x-belt moves to the right (if activated), and after passing the belts there is enough space to brake from the velocity we reach by accelerating for four cycles (each of duration $\leq \varepsilon$) and possible extra velocity gained when a conveyor belt activates:

$$x_b < \frac{1}{2}A\varepsilon^2 \land c_x > 0 \land (c_x + 4A\varepsilon)^2 \leq 2A(r_x - f_x) \tag{1}$$

For the y-direction we assume

$$c_y > 0 \land c_y^2 \leq 2A(r_y - l_y) \ , \tag{2}$$

i.e., the y-belt moves upwards (if activated) and there is enough space for the robot to compensate for the effects of the conveyor belt by braking long enough without having to leave the factory ground.

Even though these constraints limit the possible scenarios, we haven proven that a strategy for the robot exists such that it meets its objectives. Figure 4 shows the hybrid game describing the robotic factory scenario. The game is structured as follows. First the environment, i.e., Falsifier, chooses a number of iterations. In each iteration, the environment may choose to activate one of the conveyor belts if the robot is on it. Afterwards, the robot (i.e. Verifier) chooses his accelerations in x and y-direction. The clock t_s is reset to measure the cycle time (i.e. $t_s \leq \varepsilon$), then the robot chooses if it wants to brake or possibly to drive backwards (w.r.t. to its current direction). The time for the continuous evolution is then chosen by the environment within the cycle time constraint and possibly the zero crossing of one of the velocities. Thus, accelerating for ε time units can take many iterations of the loop as ε only provides an upper bound on the cycle time. Further, note that for the braking case if the velocity in a direction is 0 then that acceleration is set to 0 as well to avoid time deadlocks. Also the robot has to ensure that his choices for the acceleration are compatible with the current velocities: for a velocity v and an acceleration a, if the robot wants to brake, i.e. reduce the velocity to zero, then the product va has to be non-positive.

The winning condition for the robot is to stay safe, i.e.,:

$$l_x \leq x \leq r_x \land l_y \leq y \leq r_y \land (t \geq \varepsilon \rightarrow (x \geq x_b \lor y \geq e_y))$$

The robot must stay within the rectangle of the points (l_x, l_y) and (r_x, r_y) but has to leave the rectangle (l_x, l_y) and (x_b, e_y) after ε time units. The latter requirement models that uncooperative robotic elements might enter that region. Note that the number of iterations is chosen when the game starts not when specifying the system. Sensor and communication delays are not modelled explicitly here. Since they are beyond control for the robot, the number of iterations and the evolution durations are chosen by the factory environment. How long the robot needs to work in the factory is also decided by the factory, so the robot needs to guarantee safety for all times. However, whether the robot actually has a strategy is quite subtle. Simple strategies like always accelerating, or always braking are bound to fail and accelerating for exactly ε time units is not possible as the environment determines the actual cycle time and might not allow changing the acceleration at that exact point in time. The robot has to navigate the factory very carefully, react to changes in the conveyor belt activation as needed, and robustly adapt to the number of control loop repetitions and (possibly erratic) cycle durations chosen by the factory environment.

Verification. We consider an instance of the case study that is parametric w.r.t. ε, c_x, c_y, and x_b, but we fix $l_x = l_y = 0$, $r_x = r_y = 10$, $e_x = 2$, $e_y = 1$, $f_x = 3$,

$$\Big([\,?true \cup (?(x < e_x \wedge y < e_y \wedge \mathrm{eff}_1 = 1);\ v_x := v_x + c_x;\ \mathrm{eff}_1 := 0)$$
$$\cup\,(?(e_x \le x \wedge y \le f_y \wedge \mathrm{eff}_2 = 1);\ v_y := v_y + c_y;\ \mathrm{eff}_2 := 0)\,]$$
$$\langle\, a_x := *;\ ?(-A \le a_x \le A);\ a_y := *;\ ?(-A \le a_y \le A);\ t_s := 0\,\rangle$$
$$\big([x' = v_x, y' = v_y, v_x' = a_x, v_y' = a_y, t' = 1, t_s' = 1 \& t_s \le \varepsilon\,]$$
$$\cup(\langle ?a_x v_x \le 0 \wedge a_y v_y \le 0;\ \text{if } v_x = 0 \text{ then } a_x := 0 \text{ fi};\ \text{if } v_y = 0 \text{ then } a_y := 0 \text{ fi}\rangle$$
$$[x' = v_x, y' = v_y, v_x' = a_x, v_y' = a_y, t' = 1, t_s' = 1 \& t_s \le \varepsilon \wedge a_x v_x \le 0 \wedge a_y v_y \le 0])\big)\Big)^{[*]}$$

Fig. 4. Description of game for robotic factory automation scenario (RF)

$f_y = 10$. We have verified the following propositions using KeYmaera [11], to which we added dDGℒ proof rules. To establish the desired property, we first show that the robot can stay within the factory site whatever the factory does.

Proposition 1. *The following* dDGℒ *formula is valid, i.e., there is a strategy for Verifier in the game depicted in Fig. 4 that achieves the postcondition:*

$$(x = y = 0 \wedge (1) \wedge (2)) \to (RF)(l_x \le x \le r_x \wedge l_y \le y \le r_y)$$

When proving this property, we focus on the case where the robot is not driving towards the lower left corner; see Fig. 3.

Again allowing for arbitrary movement in x-direction, we analyze, for a projection to the x-axis, a more complex postcondition, where the robot has to leave the shaded region but stay inside the factory site.

Proposition 2. *The following* dDGℒ *formula is valid, i.e., there is a strategy for Verifier in the game in Fig. 4 projected to the x-axis (denoted $RF|_x$) that achieves the postcondition:*

$$(x = y = 0 \wedge (1)) \to (RF|_x)(l_x \le x \le r_x \wedge (t \ge \varepsilon \to (x \ge x_b)))$$

In the proof of Proposition 2 we prove the following inductive invariant:

$$\begin{aligned}
&\mathrm{eff}_1 \in \{0,1\} \wedge x \ge l_x \wedge v_x \ge 0 \wedge (t \ge \varepsilon \to x \ge x_b) \wedge (v_x + c_x \mathrm{eff}_1)^2 \le 2A(r_x - x)\\
&\wedge \Big(x < x_b \to t \le \varepsilon \wedge \big(x_b - x \le \frac{1}{2}A\varepsilon^2 - \frac{1}{2}At^2\\
&\wedge\ (\mathrm{eff}_1 = 1 \to v_x = At) \wedge (\mathrm{eff}_1 = 0 \to v_x = At + c_x)\\
&\wedge\ r_x - x \ge \frac{(v_x + \mathrm{eff}_1 c_x)^2}{2A} + A(2\varepsilon - t)^2 + 2(2\varepsilon - t)(v_x + \mathrm{eff}_1 c_x)\big)\Big)
\end{aligned} \tag{3}$$

The invariant says that enough space remains to brake before reaching the right end of the factory ground. Additionally, if the point x_b has not yet been passed then the time is not up and the distance to the right wall is bounded by the space the robot can cover by accelerating ε time units and the distance it could already have covered within the current runtime. Further, the distance

to the far right side is large enough to accelerate for another $2\varepsilon - t$ time units and brake afterwards without hitting the wall. The 2ε time units are necessary as Falsifier chooses how long to evolve and the robot may accelerate for ε time units before it can react again. Therefore, the robot may have to accelerate when clock t is almost ε and then may not react again within the next ε time units.

The KeYmaera proof [13] for Proposition 1 has 2471 proof steps on 742 branches, with 159 interactive steps. The proof for Proposition 2 has 375079 proof steps on 10641 branches (1673 interactive steps). The interactive steps provide the invariant and simplify the resulting arithmetic. Note that Proposition 1 is significantly simpler than Proposition 2, because there is a simpler strategy that ensures safety (Proposition 1), whereas the dDG$\mathcal{L}$ formula in Proposition 2 is only valid when the robot follows a subtle strategy to leave the shaded region quickly enough without picking up too much speed to get itself into trouble when conveyor belts decide to activate. Specifically, Proposition 2 needs the much more complicated invariant (3). Also, the a priori restriction (and thus strategy choice) to the case where the robot is driving in the direction towards larger x and larger y values reduces the proof for Proposition 1 significantly.

As every strategy witnessing Proposition 2 is compatible with some strategy witnessing Proposition 1, we claim that the robot meets its requirements.

5 Related Work

Our approach to hybrid games has some resemblance to Parikh's propositional game logic (GL) [6] for propositional games. But dDG$\mathcal{L}$ is a conservative extension of d$\mathcal{L}$ [7] with hybrid programs as hybrid system models. We refer to [10] for an identification of the fundamental commonalities and differences of GL versus d$\mathcal{L}$. Axiom K and Gödel's generalization rule stop to hold for games [10], but are used in KeYmaera, which had been designed for hybrid systems not games. It is, thus, crucial that K and Gödel's generalization are still sound for dDG$\mathcal{L}$. Unlike GL, dDG$\mathcal{L}$ has an advance notice semantics, i.e., the players announce the number of repetitions of a loop when it starts.

Vladimerou et. al. [15] extended o-minimal hybrid games [1] to STORMED hybrid games and proved decidability of optimal reachability. These hybrid games are based on STORMED hybrid systems, which require that all system actions point towards a common direction. Unfortunately, neither STORMED hybrid games nor their special case of o-minimal hybrid games are expressive enough for our needs. Our factory automation scenario is not STORMED, e.g., because some actions decrease the velocity (when the robot brakes) and some trajectories increase it (when the conveyor belt activates).

Tomlin et. al. [14] present an algorithm to compute maximal controlled invariants for hybrid games with continuous inputs. The class of games they consider is more general than ours as they allow inputs to differential equations to be controlled by both players, thereby added a differential game component. However, the general class of games they consider is so large, the algorithm presented is semi-decidable only for certain classes of systems, e.g., systems specified as timed

or linear hybrid automata, or o-minimal hybrid systems. They further present numerical techniques to compute approximations of their reach set computation operators. However, these sometimes give unsound results. Additionally, it only works for differentiable value functions. Extending these ideas, Gao et. al. [2] present a different technique for the same approach. The drawback is that the players can neither force discrete transitions to happen nor influence which location is reached by a discrete transition.

In contrast to these automata-based approaches to hybrid games we do not consider concurrent choices of actions. However, it is, for instance, possible to model voting for the next evolution time, provided the players can announce their choices in given orders. A precedence for player actions is often times encoded into the semantics of hybrid game automata, e.g. controller actions have precedence over environment actions in [15], whereas dDG$\mathcal{L}$ offers more flexibility in modeling these syntactically for the particular needs of the application.

References

1. Bouyer, P., Brihaye, T., Chevalier, F.: O-minimal hybrid reachability games. Logical Methods in Computer Science 6(1) (2009)
2. Gao, Y., Lygeros, J., Quincampoix, M.: On the Reachability Problem for Uncertain Hybrid Systems. IEEE Transactions on Automatic Control 52(9) (Sep 2007)
3. Harel, D.: First-Order Dynamic Logic, LNCS, vol. 68. Springer (1979)
4. Henzinger, T.A., Horowitz, B., Majumdar, R.: Rectangular hybrid games. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR. LNCS, vol. 1664. Springer (1999)
5. Maler, O., Pnueli, A., Sifakis, J.: On the synthesis of discrete controllers for timed systems (an extended abstract). In: STACS. pp. 229–242 (1995)
6. Parikh, R.: The logic of games and its applications. In: Annals of Discrete Mathematics. pp. 111–140. Elsevier (1985)
7. Platzer, A.: Differential dynamic logic for hybrid systems. J. Autom. Reas. 41(2), 143–189 (2008)
8. Platzer, A.: Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics. Springer, Heidelberg (2010)
9. Platzer, A.: Stochastic differential dynamic logic for stochastic hybrid programs. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE. LNCS, vol. 6803, pp. 431–445. Springer (2011)
10. Platzer, A.: Differential game logic for hybrid games. Tech. Rep. CMU-CS-12-105, School of Computer Science, Carnegie Mellon University, Pittsburgh (March 2012)
11. Platzer, A., Quesel, J.D.: KeYmaera: A hybrid theorem prover for hybrid systems (system description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR. LNCS, vol. 5195, pp. 171–178. Springer (2008)
12. Quesel, J.D., Fränzle, M., Damm, W.: Crossing the bridge between similar games. In: Tripakis, S., Fahrenberg, U. (eds.) FORMATS. LNCS, vol. 6919, pp. 160–176. Springer (2011)
13. Quesel, J.D., Platzer, A.: Playing Hybrid Games with KeYmaera. Tech. Rep. 84, SFB/TR 14 AVACS (April 2012), ISSN: 1860–9821, http://www.avacs.org.
14. Tomlin, C., Lygeros, J., Sastry, S.: A Game Theoretic Approach to Controller Design for Hybrid Systems. Proceedings of IEEE 88, 949–969 (Jul 2000)
15. Vladimerou, V., Prabhakar, P., Viswanathan, M., Dullerud, G.: Specifications for decidable hybrid games. Theoretical Computer Science 412(48), 6770 – 6785 (2011)