Chapter 1
An Introduction to Automatic Synthesis of Discrete and Timed Controllers

Franck Cassez, Kim Larsen, Jean-François Raskin, and Pierre-Alain Reynier

1.1 Introduction

In this chapter, we introduce models and algorithms for the automatic synthesis of controllers for discrete and timed (infinite state) systems. The techniques that we expose here are based on the game metaphor [5, 4]: when designing an embedded controller, you can see the controller as interacting with its environment. As the actions taken by the environment are uncontrollable, those actions should be considered as *adversarial*. Indeed, a controller should be correct no matter how the environment in which it operates behaves. The models, algorithms and tools presented here are applied to an industrial case study in the next chapter. This case study was provided to us by HYDAC ELECTRONIC GMBH within the Quasimodo project.

The objective of the chapter is to allow the reader to understand timed game automata [3] as a model for solving timed control problems. With this objective in mind, we define the notions of game graphs, controllable and uncontrollable actions, strategies, and winning objectives. We also give a gentle introduction to the main algorithmic ideas that are used to solve games played on graphs. Those techniques are used in the tool UPPAAL-TIGA [1]. A good understanding of the techniques used in UPPAAL-TIGA should help the users when modeling control problems and formulating queries about their models.

The chapter is organized as follows. In section 1.2, we introduce an example of a timed control problem called the‘Chinese juggler control problem”. This example allows us to illustrate the game metaphor for formalizing the timed control problem. In Section 1.3, we introduce the basic definitions underlying the game approach to controller synthesis. In Section 1.4, we outline two algorithms that are used to solve (untimed) reachability and safety games respectively. In Section 1.5, we show how the concepts developed in Sections 1.3 and 1.4 can be extended to timed systems. In Section 1.6, we summarize the main ideas underlying the algorithms for solving timed games. In Section 1.7, we give an introduction to the tool UPPAAL-TIGA and

show how to model and automatically solve the Chinese juggler control problem with timed game automata.

1.2 The Chinese juggler control problem

In this section, we introduce a running example that we use later in this chapter to illustrate how timed controllers can be automatically synthesized using the tool UPPAAL-TIGA. The example also allows us to illustrate the game metaphor for controller synthesis that underlies the development of the theory in Sections 1.3 and 1.4.

A Chinese juggler has to spin plates that are on sticks to prevent them from falling, see Fig. 1.1 for an illustration. Assume, for our example, that the juggler wants to handle two plates, called `Plate 1` and `Plate 2`. Plates crash after a while if they are not spun. Initially, each plate is spinning on its stick and the spin is fast enough so that they will stay stable for at least 5 seconds. The juggler has to maintain them stable for as long as possible (forever if possible). For that, the juggler can spin each plate but he can spin only one of the plates at a time. When he decides to spin `Plate` $i \in \{1,2\}$, he should do it for at least 1 time unit. If he decides to do it for t time units then `Plate` i stays stable for 3 time units if $1 \leq t \leq 2$, and for 5 time units if $t > 2$.

Now, assume that there is also a mosquito in the room. When the mosquito touches one of the two plates, it reduces the spinning of the plate, and as a result its remaining stability time is decreased by 1 time unit. When the mosquito touches the plate, it gets afraid and this guarantees that it will not touch any plate again before D time units have elapsed (after that amount of time the mosquito has forgotten and he is not afraid any more).

We want to answer the following question (CP):

> Given a value for D, does the Chinese juggler have a way to spin the plates so that none of the two plates ever falls down no matter what the behavior of the mosquito is ?

Let us first try to understand how this timed control problem can be seen as a two player game. In the system underlying our example, we have several components: the Chinese juggler, the plates, and the mosquito. Clearly, only the behavior of the Chinese juggler is under control. The plates and the mosquito are part of the environment: when a plate has not been spun enough, it can fall at any time, and the behavior of the mosquito is out of control of the juggler, i.e. the mosquito decides when it touches plates. As a consequence, we can see the control problem as *opposing* two players: on one hand the Chinese juggler (Player 1), and on the other hand the plates and the mosquito (Player 2).

During this game, at any point in time, the Chinese juggler may decide to spin one of the two plates. If he decides to do so, he will do it for at least one time unit. Then either he decides to continue to spin the plate, or to stop and remain idle for a while, or to start spinning the other plate. The alternatives that are offered to

Fig. 1.1 A Chinese juggler (cartoon courtesy of Jean Cardinal.)

the juggler along time can be understood as *moves* in the underlying game. The mosquito, if it has not touched a plate in the preceding D time units, may decide to touch one of the two plates whenever it wishes to do so. Again, those alternatives can be seen as moves in the underlying game. Similarly, when a plate does not spin fast enough then it may crash at any moment. To summarize, the only moves that we *control* are the moves of the Chinese juggler, they are the moves of Player 1, all the other moves are *uncontrollable*, they are the moves of Player 2. We must be prepared to face all the moves available to the mosquito and to the plates.

Second, we need to understand what the *objectives* of the two players are. The objective of the Chinese juggler is to avoid the plates to crash. For the objective of the plates and the mosquito (Player 2), it may not be as clear. The mosquito flies randomly in the room and touches one of the plates on occasion. But clearly, we want to devise a strategy for the Chinese juggler such that, *whatever the behavior of the mosquito is* (within the hypothesis that it does not touch twice the plates within less than D time units), the plates never crash. So, even if we do not know exactly the intention (or the exact specification) of the mosquito, it is safe to be prepared for the worst case scenario. So the kind of game that we consider are *zero sum games*: a set of behaviors (of the system) is identified as good for Player 1, and the complement of this set (all other behaviors) are considered as good for Player 2.

1.3 Control as a two-player games

Now that we agree that control problems can be seen as two-player games, we introduce the precise definitions underlying the theory of two-player games played on graphs. Later we extend those notions with dense time. After presenting timed games, we show how to model the Chinese juggler problem with timed game au-

tomata and how we can answer question (*CP*); moreover if the answer is yes we also show how to synthesize automatically a winning strategy for the Chinese juggler using the tool UPPAAL-TIGA.

1.3.1 Game structures

A *game structure* is a tuple $G = (L, \ell_{\mathsf{init}}, \mathsf{Act}_1, \mathsf{Act}_2, E)$ where L is a finite (non-empty) set of locations, $\ell_{\mathsf{init}} \in L$ is the *initial location* of the game, Act_1 and Act_2 are the two disjoint *sets of actions* for Player 1 (the controller) and Player 2 (the environment) respectively, and $E \subseteq L \times \mathsf{Act}_1 \cup \mathsf{Act}_2 \times L$ is a *set of edges* between the locations of the game labelled by actions that belong either to Player 1 or to Player 2. Intuitively, edges labeled with elements from Act_1 belong to Player 1 and are controllable (represented by plain arrows) while edges labeled by elements from Act_2 belong to Player 2 and are uncontrollable (represented by dashed arrows). We let $\mathsf{Enabled}_i(\ell)$ be the set of actions of Player $i \in \{1,2\}$ available at location ℓ i.e. $\mathsf{Enabled}_i(\ell) = \{\alpha \in \mathsf{Act}_i \mid \exists(\ell, \alpha, \ell') \in E\}$.

We require that for all $\ell \in L$, $\mathsf{Enabled}_1(\ell) \neq \varnothing$, so that Player 1 is always able to propose an action to play in any location of the game.

Example 1. Fig. 1.2 depicts a game structure. The set of locations is $L = \{\mathtt{L0}, \mathtt{L1}, \mathtt{L2}, \mathtt{L3}\}$. In location L0, Player 1 can choose between action a or action b, while Player 2 can choose between action $u1$ and action $u2$. L0 (inner circle) is the initial location of the game. The edge $(\mathtt{L0}, a, \mathtt{L1})$ belongs to Player 1 and the edge $(\mathtt{L0}, u1, \mathtt{L1})$ belongs to Player 2. □

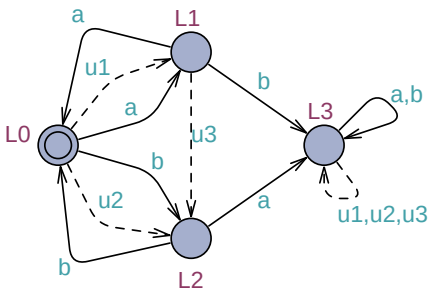

Fig. 1.2 An example of a game structure

The way players are playing on a game structure $G = (L, \ell_{\mathsf{init}}, \mathsf{Act}_1, \mathsf{Act}_2, E)$ is defined as follows. Initially, a pebble lies on ℓ_{init}, the initial location of game. Then the game is played in rounds. Let us assume that, for the current round, the pebble lies on location $\ell \in L$. Then, first, Player 1 chooses some action $\alpha \in \mathsf{Enabled}_1(\ell)$.

Then Player 2 decides where to move the pebble onto the successor locations of ℓ while respecting the following rule: for moving the pebble she uses either an edge labeled with an action of Act_2 or an edge[1] labeled with the action α chosen by Player 1. By interacting in such a way for an infinite number of rounds, Player 1 and Player 2 are constructing a *play*. Formally a play $\rho = \ell_0\ell_1\ldots\ell_n\ldots$ of the game structure G is an infinite sequence of locations. We let $\rho[i] = \ell_i, i \geq 0$ and denote $\mathsf{Play}(G)$ for the set of plays of G.

Example 2. Let us illustrate the notion of play using the example of Fig. 1.2. As L0 is the initial location of the game structure G, the pebble initially lies on L0. Then Player 1 is asked to make a choice among the actions that are available for her in location L0. This set is $\{a,b\}$. Assume that she chooses a. In this case, there are two possibilities. Either, Player 2 chooses to let Player 1 play and the pebble is moved using an edge labeled with the letter a. In our example, there is only one such edge, and so the pebble is moved on location L1. By this interaction, a finite prefix L0L1 of play is built. Or, Player 2 chooses to overtake Player 1 and to play $u2$; in that case, the finite prefix L0 L2 of a play is built. Assume that the second situation applies. Then a new round starts in L2. In that location, there is no uncontrollable transition, so if Player 1 chooses a then the pebble is moved to L3 and if she chooses b, it is moved to L0, etc. □

1.3.2 Winning objectives and strategies

We have seen that the interaction between Player 1 (the controller) and Player 2 (the environment) on a game structure $G = (L, \ell_{\mathsf{init}}, \mathsf{Act}_1, \mathsf{Act}_2, E)$ generates a play which is an infinite sequence $\ell_0\ell_1\ldots\ell_n\ldots$ of locations in the game graph, i.e. the sequence of locations traversed by the pebble during the course of the game. Such a sequence models one behavior of the system under control, and this behavior could be considered as a *good* behavior or as a *bad* behavior depending on what we expect from our system[2]. In the game terminology, such a classification of good and bad behaviors leads to the notion of winning objective. A *winning objective* for a game structure G is a set of infinite sequences of locations, the intention being that such sequences represent the good behaviors of the system.

Example 3. Assume that in our running example, Player 1 has the objective to reach the set of locations $\{\mathtt{L3},\mathtt{L4}\}$. In this case the winning objective will contain all the plays $\ell_0\ell_1\ell_2\ldots\ell_n\ldots$ such that $\ell_i = \mathtt{L3} \vee \ell_i = \mathtt{L4}$ for some $i \geq 0$. □

In the example above, the winning objective is a so-called *reachability objective* as it specifies a set of locations that we want to visit. In this chapter, we concentrate on two classes of objectives: *reachability* and *safety*. Given a set of *target* locations

[1] There might be more than one α-successor of ℓ. In this case, Player 2 resolves the non-deterministic choice of the α-successor.

[2] As stated earlier, we play zero-sum games and in this case a play is either good or bad.

$T \subseteq L$, we define the set of winning plays of the *reachability objective defined by* T as the set of plays $\mathsf{Reach}_G(T) = \{\rho \in \mathsf{Play}(G) \text{ s.t. } \exists i \geq 0 \cdot \rho[i] \in T\}$. Given a set of *safe* locations $S \subseteq L$, we define the set of winning plays of the *safety objective defined by* S as the set $\mathsf{Safe}_G(S) = \{\rho \in \mathsf{Play}(G) \text{ s.t. } \forall i \geq 0 \cdot \rho[i] \in S\}$. In the sequel, we often use Obj to represent a set of winning plays.

The winning objective specifies what the good plays for Player 1 are. Those good behaviors can be enforced by the controller (Player 1) if she has a strategy to force the play to be within the winning objective no matter what the strategy played by Player 2 is (so without the help of Player 2). For the later definition to be completely clear, we need to define more precisely what a *strategy* is. In our games, a strategy for Player 1 determines what actions from Act_1 to pick during the course of the game. In general, a strategy may depend on the history of the game for deciding what the good action to play is. Nevertheless, for reachability and safety objectives, the situation is simpler and it can be shown that strategies that only depend on the current position of the pebble are sufficient: those strategies are called "memoryless" strategies. So, in this chapter we concentrate on such simple strategies. We now define them formally. A *(memoryless) strategy* for Player 1 is a function $\lambda_1 : L \to \mathsf{Act}_1$, i.e. it is a function that given the current location $\ell \in L$ chooses an action $\lambda_1(\ell) \in \mathsf{Enabled}_1(\ell)$ for Player 1.

Let us now define the possible behaviors in the game structure $G = (L, \ell_{\mathsf{init}}, \mathsf{Act}_1, \mathsf{Act}_2, E)$ when Player 1 plays according to the strategy λ_1. Remember that Player 1, in the game above, chooses an action at each round. Then Player 2 chooses between the edges labeled by this action or labeled with one of her own actions. The set of behaviors in this case is thus the set of paths that start in ℓ_{init} and use only edges that are either labeled with actions of Player 2 or labeled with actions that are prescribed by the strategy λ_1. We can also see a strategy for Player 1 as cutting out edges of Player 1 that are not chosen by the strategy. Let us define that formally. We call the *outcome of the strategy* λ_1 in the game $G = (L, \ell_{\mathsf{init}}, \mathsf{Act}_1, \mathsf{Act}_2, E)$ the set of plays

$$\mathsf{Outcome}(G, \lambda_1) = \{\rho \mid \forall i \geq 0, \exists a \in \lambda_1(\rho[i]) \cup \mathsf{Act}_2 \cdot (\rho[i], a, \rho[i+1]) \in E\}.$$

A strategy λ_1 is *winning* for the objective Obj in G if $\mathsf{Outcome}(G, \lambda_1) \subseteq \mathsf{Obj}$.

Example 4. Let us consider again the example of Fig. 1.2. Assume that the winning objective for Player 1 is to reach location L3, i.e. $\mathsf{Obj} = \mathsf{Reach}_G(\{\mathtt{L3}\})$. Let us consider the strategy λ_1 defined as follows: $\mathtt{L0} \mapsto b$, $\mathtt{L1} \mapsto b$, $\mathtt{L2} \mapsto a$, and $\mathtt{L3} \mapsto a$. It should be clear that no matter how Player 2 plays when the play starts in L0, the result of the interaction with this strategy λ_1 is a play that reaches L3. For example, let us consider the following scenario: in L0, instead of playing b as chosen by Player 1, Player 2 moves the pebble to location L1. From there, instead of playing b as asked by Player 1 (this would lead directly to L3), Player 2 moves the pebble to L2. From L2, Player 1 chooses a and Player 2 has no other choices than to move the pebble to L3. So, under any adversarial behavior of Player 2, Player 1 can force the pebble to reach location L3. As a consequence, λ_1 is a winning strategy for Player 1 to win the reachability game defined by the objective $\mathsf{Obj} = \mathsf{Reach}_G(\{\mathtt{L3}\})$. □

1.4 Solving two-player games

In the previous section, we have defined two-player game structures, reachability and safety winning objectives, strategies for Player 1, and we have explained when a strategy for Player 1 is winning. In this section, we introduce the basic ideas that are underlying algorithms for solving games with safety and reachability objectives.

To understand the basic ideas behind the algorithms for solving reachability and safety games, we must first concentrate on what happens in one round, i.e. we need to consider one-step objectives. A *one step objective* is defined by a set of locations $T \subseteq L$. In a location $\ell \in L$ of the game $G = (L, \ell_{\mathsf{init}}, \mathsf{Act}_1, \mathsf{Act}_2, E)$, Player 1 wins the one step objective T if there exists an action $\alpha \in \mathsf{Act}_1$ such that all edges labeled by α and all edges labeled with Act_2 actions lead to a location in T, i.e. ℓ is such that

$$\exists \alpha \in \mathsf{Act}_1 \cdot \forall \beta \in \mathsf{Act}_2 \cup \{\alpha\} \cdot \forall (\ell, \beta, \ell') \in E : \ell' \in T.$$

In that case, we say that ℓ is a *controllable predecessor* of T, and we denote by $\mathsf{CPre}(T)$ the set of locations that are controllable predecessors of T.

Example 5. To illustrate the definition of *controllable predecessors*, we use Fig. 1.3. First, let us consider the set of locations $T_1 = \{\mathtt{L1}, \mathtt{L3}\}$. The location L0 is a controllable predecessor of T_1. Indeed, in L0 if Player 1 chooses a, no matter what is the choice of Player 2 (to move the pebble using an edge labeled with a or to play an edge labeled with her own actions) the pebble will be either in L1 or L3 after the round, so it will lie in T_1. Second, let us consider the set of locations $T_2 = \{\mathtt{L1}, \mathtt{L2}\}$. The location L0 is not a controllable predecessor of T_2. Indeed, neither a nor b ensures that the pebble will lie in T_2 as Player 2 can choose to go to L3 using a or $u2$ in the first case, and Player 2 can decide to go to L1 using $u1$ in the second case. □

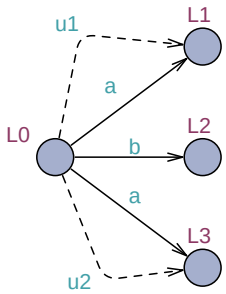

Fig. 1.3 Controllable and uncontrollable predecessors

Now that we understand what it means for a location ℓ to be a controllable predecessor of a set of locations T, we provide algorithms to solve reachability and safety games. Let us start with reachability games. Let $G = (L, \ell_{\mathsf{init}}, \mathsf{Act}_1, \mathsf{Act}_2, E)$

be a two-player game structure and $\mathsf{Obj} = \mathsf{Reach}_G(T)$ be the reachability objective for Player 1.

The algorithm that computes the set of winning locations for the reachability objective $\mathsf{Reach}_G(T)$ works by induction on the number of rounds needed for Player 1 to win. Clearly, all the locations in T are winning in 0 rounds, let us denote this set of locations by W_0. Now, it should be clear that the set of controllable predecessors of T are locations that are winning in 1 step. By taking the union of this set with W_0, we obtain the set of locations from which Player 1 can force a visit to T in 0 or 1 rounds, i.e. $W_1 = W_0 \cup \mathsf{CPre}(W_0)$. Generalizing this reasoning, we get that $W_i = W_{i-1} \cup \mathsf{CPre}(W_{i-1}), i \geq 1$ is the set of locations from which Player 1 can force a visit to the set T in less than i rounds. Clearly, we have that $W_0 \subseteq W_1 \subseteq \cdots \subseteq W_i \subseteq L$. As L is a finite set, the monotonic sequence of W_i reaches a fixed point W for some $k \leq |L|$ and $W = W_k = W_{k-1}$. The set W is the set of locations from which Player 1 has a strategy to force a visit to T in a finite number of steps. If $\ell_{\mathsf{init}} \in W$ then Player 1 has a winning strategy from the initial location of the game. From the computation of this sequence, we can extract a winning strategy for all locations in W as follows. Let $\ell \in W$ be such that $\ell \in W_i, i \geq 1$ and $\ell \notin W_{i-1}$. Define $\lambda_1(\ell)$ to be any action a such $a \in \mathsf{Act}_1$ and all the edges labeled with a that leave the location ℓ go to a location that belongs to the set W_{i-1}; because of the definition of W_i and CPre, such an action a is guaranteed to exist.

Example 6. Let us consider again the example of Fig. 1.2 with the reachability objective $\mathsf{Obj} = \mathsf{Reach}_G(\{\mathtt{L3}\})$. Let $W_0 = \{\mathtt{L3}\}$, and let us compute the set of controllable predecessors of W_0. The locations L2 and L3 are controllable predecessors of W_0. So $W_1 = \{\mathtt{L2}, \mathtt{L3}\}$ is the set of locations from which Player 1 can ensure a visit in $\{\mathtt{L3}\}$ in 0 or 1 rounds. It should be clear from the computation of the controllable predecessors of W_0 that Player 1 has to choose the action a when the pebble lies on L2. This gives a winning strategy for Player 1 in $\mathtt{L}_2$. Now, let us consider the locations that are controllable predecessors of W_1. This set is $\{\mathtt{L1}, \mathtt{L2}, \mathtt{L3}\}$. Indeed in L1 Player 1 can choose b and in this case, either Player 2 moves the pebble to L3 using edge $(\mathtt{L1}, b, \mathtt{L3})$ or she moves the pebble to L2 using the edge labeled by $u3$. In the two cases, the pebble lies on $\{\mathtt{L2}, \mathtt{L3}\}$ when starting the next round of the game. If we continue like that we obtain that all the locations of G are winning for the objective Obj and in the process we can construct a winning strategy for Player 1. □

Let us now turn to safety games. Remember that in a safety game defined by a set $S \subseteq L$ of locations, Player 1 has the objective to stay within set S forever, i.e. $\mathsf{Obj} = \mathsf{Safe}_G(S)$. Let us define, as for reachability, a sequence of sets of locations that approximate the set of winning locations for Player 1. Clearly, $W_0 = S$ is the set of locations from which Player 1 can ensure to stay within S for at least 0 rounds. Now, $W_1 = W_0 \cap \mathsf{CPre}(W_0)$ is the set of locations from which Player 1 can ensure to stay within S for at least 1 round, and more generally, $W_i = W_{i-1} \cap \mathsf{CPre}(W_{i-1}), i \geq 1$ is the set of locations from which Player 1 can ensure to stay within S for ta least i rounds. Clearly, we have that $L \supseteq W_0 \supseteq W_1 \supseteq \cdots \supseteq W_i \supseteq \cdots \supseteq \varnothing$. As L is a finite set, we must reach a fixed point W for some $k \leq |L|$ and $W = W_k = W_{k-1}$, and so the sequence eventually stabilizes on the set of locations from which Player 1 can

force to stay within S forever, i.e. on the set of locations from which Player 1 has a strategy to win the safety game defined by S.

Example 7. Let us consider again example of Fig. 1.2 but now with the objective $\mathsf{Obj} = \mathsf{Safe}_G(\{\mathtt{L}_0, \mathtt{L}_2\})$. So the objective for Player 1 is now to avoid locations $\mathtt{L}_1$ and $\mathtt{L}_3$. Let us compute the sequence of sets of locations that approximate the winning set for Player 1. By definition of this sequence, $W_0 = \{\mathtt{L}_0, \mathtt{L}_2\}$. Let us compute the controllable predecessors of this set of locations: $\mathsf{CPre}(W_0) = \{\mathtt{L2}\}$. Indeed, L0 is not a controllable predecessor of $\{\mathtt{L0}, \mathtt{L2}\}$ as, from L0, Player 2 can force to move the pebble onto the location L1 by choosing to play the edge labeled by $u2$. While $\mathtt{L}_2$ is a controllable predecessor of the set W_0 as in $\mathtt{L}_2$ Player 1 can move the pebble onto the location $\mathtt{L}_0 \in \{\mathtt{L}_0, \mathtt{L}_1\}$ by playing the action b, so $W_1 = \{\mathtt{L2}\}$. And clearly, $\mathsf{CPre}(W_1) = \varnothing$. So, there is no location in G from which Player 1 can ensure to stay within $\{\mathtt{L0}, \mathtt{L2}\}$ forever and Player 1 cannot win the game.

Let us now change the objective and consider $\mathsf{Obj} = \mathsf{Safe}_G(\{\mathtt{L}_0, \mathtt{L}_1, \mathtt{L}_2\})$. We start the computation with $W_0 = \{\mathtt{L}_0, \mathtt{L}_1, \mathtt{L}_2\}$, and compute $W_1 = W_0 \cap \mathsf{CPre}(W_0)$. All the edges leaving $\mathtt{L}_0$ reach a location in W_0 so $\mathtt{L}_0 \in \mathsf{CPre}(W_0)$, in $\mathtt{L}_1$ all edges of Player 2 and all edges of Player 1 labeled with a reach a location in W_0 so $\mathtt{L}_1 \in \mathsf{CPre}(W_0)$, and in $\mathtt{L}_2$ all edges of Player 2 and all edges of Player 1 labeled with b reach a location in W_0 so $\mathtt{L}_2 \in \mathsf{CPre}(W_0)$. The sequence of sets stabilizes as $W_1 = W_0$, and so Player 1 has a strategy to win the safety objective $\mathsf{Obj} = \mathsf{Safe}_G(\{\mathtt{L}_0, \mathtt{L}_1, \mathtt{L}_2\})$ from all locations in $\{\mathtt{L}_0, \mathtt{L}_1, \mathtt{L}_2\}$. □

Remark 1. The main drawback of the algorithms that we have outlined above is that they compute winning information about locations that are not necessarily reachable by an interaction between Player 1 and Player 2 from the initial location. In practice, that can deteriorate the performances of the algorithms. There are solutions to avoid that problem, see for example the on-the-fly algorithm of [2], but the description of those solutions goes beyond the objectives of this introduction.

1.5 Adding time to game structures

To add time to game structures, we adapt the syntax of timed automata as defined in Chapter XXX and partition discrete transitions as controllable and uncontrollable. A timed game automaton $G = (L, \ell_{\mathsf{init}}, X, \mathsf{Inv}, \mathsf{Act}_1, \mathsf{Act}_2, E)$ is a structure, where:

- L is a finite set of discrete locations and ℓ_{init} is the initial location of the timed game;
- X is a finite set of clocks, and we denote by $\mathsf{Constr}(X)$ the set of *clock constraints*, i.e. conjunctions of atomic constraints of the forms $x \sim c$ or $x - y \sim c$, where $c \in \mathbb{N}$ and $x, y \in X$;
- $\mathsf{Inv} : L \to \mathsf{Constr}(X)$ is a function that labels each location $\ell \in L$ with an invariant $\mathsf{Inv}(\ell)$ that restricts the possible values that clocks in X can take when the control of the automaton is in location ℓ,

- Act_1 are the actions of Player 1, Act_2 are the actions of Player 2 such that $\mathsf{Act}_1 \cap \mathsf{Act}_2 = \varnothing$, and $E \subseteq L \times (\mathsf{Act}_1 \cup \mathsf{Act}_2) \times \mathsf{Constr}(X) \times 2^X \times L$ is the set of discrete transitions of the timed game. A tuple $(\ell, \alpha, \phi, R, \ell') \in E$ is a transition that goes from location ℓ to location ℓ', that is labeled with action α (if $\alpha \in \mathsf{Act}_1$ then the transition is controllable, otherwise it is uncontrollable), with guard ϕ (the transition can be taken only if the values of clocks satisfy the guard), and reset set R (the clocks in the set R are reset when the transition is taken).

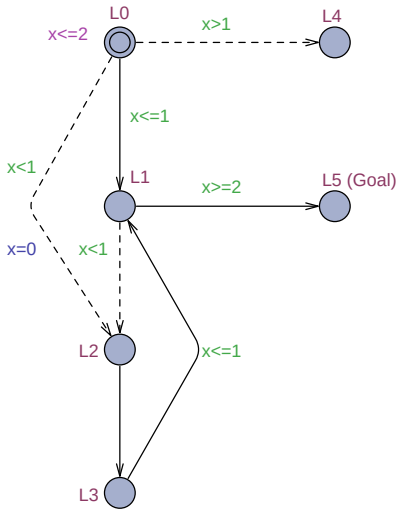

Fig. 1.4 An example of a timed game automaton

Example 8. An example of a timed game automaton is given in Fig. 1.4. The only syntactical difference with plain timed automata is induced by the partition of the alphabet of labels for the transitions: the transitions labeled with an element of Act_1 belong to Player 1, and the transitions labeled with an element of Act_2 belong to Player 2. As for untimed games, the edges controlled by Player 1 are depicted by plain edges, and the edges controlled by Player 2 are depicted as dashed edges.

A state of a timed automaton is a pair (ℓ, v), where ℓ is a location and v is a valuation for the clocks, i.e. a function $v : X \to \mathbb{R}_{\geq 0}$ that assigns to each clock $x \in X$ a positive real number $v(x)$. In a timed automaton, when the automaton is in a state (ℓ, v), time can elapse as long as it does not violate $\mathsf{Inv}(\ell)$ (the invariant that labels ℓ). For example in the timed game automaton of Fig. 1.4, from state $(\mathrm{L0}, v)$ with $v(x) = 1$, time can elapse for t time units if $1 + t \leq 2$, in that case state $(\mathrm{L0}, v')$ is reached with $v'(x) = v(x) + t$.

A transition $(\ell_1, a, \phi, R, \ell_2)$ can be taken in state (ℓ, v) whenever $\ell = \ell_1$, the guard ϕ is satisfied by v, which is denoted by $v \models \phi$, and the clock valuation $v[R := 0]$,

which maps a clock $x \in X \setminus R$ to $v(x)$, and a clock $x \in R$ to 0, is such that it satisfies $\mathsf{Inv}(\ell_2)$, i.e. $v[R := 0] \models \mathsf{Inv}(\ell_2)$. For instance, in the timed game automaton of Fig. 1.4, in state $(\ell_0, \frac{1}{2})$, Player 2 can take the uncontrollable transition to ℓ_2 as the guard on x is satisfied ($\frac{1}{2} < 1$.) The state that is reached after this transition is the pair $(\ell_2, 0)$ as the clock x is reset by this transition.

For a more systematic presentation of the semantics of timed automata, the reader is referred to Chapter XXX. In this section, we focus on intuitions and do not always give all the formal definitions.

1.5.1 Rounds in timed games

Remember that *untimed* two-player games are played for an infinite number of rounds. Each round is played as follows: Player 1 chooses one action $\alpha \in \mathsf{Act}_1$ among the actions that label the controllable transitions leaving the location where the pebble lieson , and then Player 2 moves the pebble by using a transition that is labeled either by α or by an action from Act_2 (an uncontrollable transition.)

In timed games, we additionally need to know at what time Player 1 wants to play. So in addition to an action to play, Player 1 chooses a delay t. Then given a pair (t, α), Player 2 either decides to wait for t time units and to take a transition that is labeled with the letter $\alpha \in \mathsf{Act}_1$, and for which the guard on the transition is satisfied, or Player 2 decides to wait for a delay of $t' \leq t$ and use a transition labeled by an action from Act_2, and for which the guard evaluates to true.

Example 9. Let us consider the timed game automaton of Fig. 1.4. As in this example, there are at most one controllable and one uncontrollable transition out of each location, we did not give names to the transitions. This example is a timed game automaton with a reachability objective for Player 1: her objective is to reach the location labeled with goal. Initially the pebble lies on L0 and the value of the clock x is equal to 0. Let us assume that Player 1 proposes to wait exactly for 1 time unit and to take the transition that leads to L1. In this case, the two following scenarios are possible. Either Player 2 lets time elapse for at least 1 time unit, the value of the clock x is then equal to 1, and the pebble can be moved on location L1 as proposed by Player 1 (indeed, the guard $x \leq 1$ is satisfied.) Or, Player 2 decides to wait for $t < 1$ time units, and to move the pebble to location L2 using the uncontrollable edge from L0 to L2. Again, this is possible because after waiting for $t < 1$ time units, the value of x is less than 1 time unit, and so the guard $x < 1$ on the transition from L0 to L2 is satisfied.

Assume for now that Player 2 follows the second scenario. The pebble is now lying on L2 and the value of clock x is equal to 0 (as it has been reset when moving the pebble using the transition from L0 to L2.) From that position, let us assume that Player 1 chooses to wait for $\frac{1}{2}$ time units and proposes to move to location L3. As there is no alternative for Player 2, time elapses for $\frac{1}{2}$ time units, and the pebble is moved from L2 to L3.

From there, Player 1 chooses to wait for $\frac{1}{2}$ time units and proposes to move the pebble to L1. Again, as there is no alternative for Player 2, time elapses for $\frac{1}{2}$ time units and the pebble is moved from L3 to L1. When the pebble arrives on L1, the value of the clock x is equal to $\frac{1}{2}+\frac{1}{2}=1$. Then Player 1 chooses to wait say for $1\frac{1}{4}$ time units and to move the pebble to location goal. This is a valid move as the value of x is then equal to $1+1\frac{1}{4}=2\frac{1}{4}$ and so the guard $x \geq 2$ is satisfied, again as Player 2 has no other alternatives, the pebble is moved on location goal, and the play is winning for Player 1. □

When moving the pebble according to the rules defined above, a *timed play* of the form $(\ell_0,v_0) \xrightarrow{(t_0,e_0)} (\ell_1,v_1) \xrightarrow{(t_1,e_1)} \dots \xrightarrow{(t_{n-1},e_{n-1})} (\ell_n,v_n) \xrightarrow{(t_n,e_n)} \dots$ is generated by the interaction between the two players. In this timed play, each (t_i,e_i) specifies the time that has elapsed and the transition that has been taken during the round i.

As for untimed games, objectives are defined by a set of discrete locations $T \subseteq L$ of the automaton that Player 1 wants to reach for reachability games, or by a set of discrete locations $S \subseteq L$ in which Player 1 wants to stay in for safety games. For reachability and safety objectives, it can be shown that Player 1 has a winning strategy if and only if she has a winning memoryless strategy [3]. For timed games, a *memoryless strategy* is a function $\lambda_1 : L \times \mathbb{R}_{\geq 0}{}^{|X|} \to \mathbb{R}_{\geq 0} \times \mathsf{Act}_1$ that specifies, given the current state of the game (ℓ,v), the time $t \in \mathbb{R}_{\geq 0}$ to wait and the action $\alpha \in \mathsf{Act}_1$ to play.

1.6 Solving two-player timed games

We have seen that, in the case of untimed games, reachability and safety objectives can be solved using a notion of *controllable predecessors*. This notion can be extended to timed games. Again, we do not formalize all the details here but we give enough intuition so that the reader can understand the main ideas behind the algorithms for solving timed games.

Intuitively, a state (ℓ,v) is a controllable predecessor of a set of states $T = \{(\ell_0,v_0),(\ell_1,v_1),\dots,(l_n,v_n),\dots\}$, if there exist $\alpha \in \mathsf{Act}_1$ and a delay $t \in \mathbb{R}_{\geq 0}$ such that the following four conditions hold:

1. for all delays t', $0 \leq t' \leq t$, $v+t' \models \mathsf{Inv}(\ell)$, i.e. time can elapse from (ℓ,v) for t time unit without violating the invariant labeling ℓ;
2. there exists a transition $e=(\ell,\alpha,\phi,R,\ell')$ such that $v+t \models \phi$ and $v+t[R:=0] \models \mathsf{Inv}(\ell')$, i.e. there is a transition labelled with α that can be taken after t time units;
3. for all transitions $e=(\ell,\alpha,\phi,R,\ell')$ such that $v+t$ satisfies ϕ, $(\ell',v+t[R:=0])$ belongs to T, i.e. any choice of a transition labelled with α taken after t time units leads to T;
4. for all transitions $e=(\ell,u,\phi,R,\ell')$ and delays t' such that $0 \leq t' \leq t$, $u \in Act_2$, and $v+t'$ satisfies ϕ, then $(\ell',v+t'[R:=0])$ belongs to T, i.e. any uncontrollable transition that can be taken within t time units leads to T.

The sets of states that we have to handle are infinite, so they cannot be represented in extension. We need a symbolic data structure able to represent infinite sets. Those sets can be represented symbolically using formulas in an adequate constraint language. All sets manipulated during the computation of the timed controllable predecessors are representable by union of clock constraints. To illustrate the use of clock constraints and how the computation of the controllable predecessor in the timed setting is done, we consider our running example of Fig. 1.4.

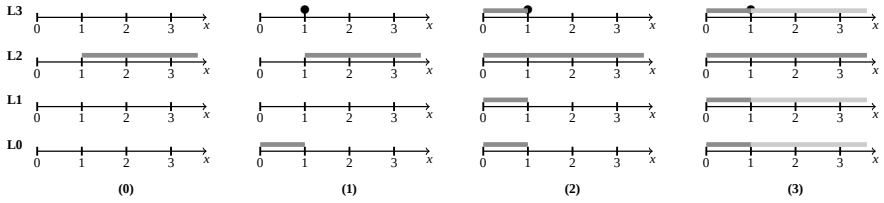

Fig. 1.5 Computation of the timed controllable states.

Example 10. The computation of the set of winning states is depicted in Fig. 1.5. The first part of the picture, marked (1), depicts the set of states of the form $(\mathtt{L1}, v)$ with $v \geq 1$. All those states are winning in 1 step because when $x \geq 1$, the uncontrollable transition from L1 to L2 cannot be taken by Player 2 (as it is guarded by $x < 1$), and by waiting until clock x reaches a value equal to or greater than 2, Player 1 can move the pebble from L1 to location goal. The part marked (2) of the picture depicts the set of states that are winning in at most 2 discrete steps. The states that have been added are controllable predecessors of the states that are winning in 1 step. First, let us consider $(\mathtt{L0}, v)$ with $v(x) = 1$. This state is winning as, on the one hand, none of the uncontrollable transitions is enabled in this state, and on the other hand, the controllable transition from L0 to L1 is enabled, and when it is taken the game reaches a winning state (in 1 step.) Second, consider the set of states $(\mathtt{L3}, v)$ with $v(x) \leq 1$. From all those states, Player 1 can wait until $x = 1$ and then she can take the controllable transition to L2, reaching a set of winning states in 1 step. The states depicted in part (3) and (4) are computed in a similar manner. □

1.7 The Chinese juggler control problem in UPPAAL-TIGA

UPPAAL-TIGA is a tool developed at Aalborg University. It handles timed game automata as presented in the previous section. The tool can be downloaded from `http://www.cs.aau.dk/~adavid/tiga/download.html`. We refer the reader to the user manual for details about the features and the usage of UPPAAL-TIGA in practice. In this section, we show how to model the Chinese juggler control problem with timed game automata. We use screenshots from the tool to illustrate

Kim, can we add this on the tiga web site ???

its user interface. The interested reader can download the UPPAAL-TIGA model of our running example from `http://...` .

Note that for the sequel, we assume that the reader is familiar with notation of the UPPAAL tool as described in Chapter YY. The models that are used here are game extensions of the UPPAAL models, we make it clear what are those extensions in the sequel.

1.7.1 Modeling of the components

A timed game in UPPAAL-TIGA is modeled compositionally by defining timed game automata that specify the behavior of the components of the system. This modeling approach is similar to the one used for regular models in UPPAAL (see Chapter YY for additional material on compositional modeling). As in UPPAAL models, components in UPPAAL-TIGA synchronize using shared events (implemented by channels). For the rest of this section, we assume that the reader is familiar with this modeling paradigm.

Fig. 1.6 shows the timed automaton model for `Plate` $i \in \{1,2\}$. The timed game automaton has the set of locations $\{$`Stable`, `Spinning`, `Longspinning`, `Crashed`$\}$. The location `Stable` intends to model the situation when the plate is stable, the location `Crashed` models the situation when the plate has crashed, `Spinning` models the situation when the juggler does spin the plate spin for a time $t \leq$ `STABSHORT` seconds (where `STABSHORT` is a constant equal to 2), and `Longspinning` when the juggler does spin the plate for more than `STABSHORT` seconds.

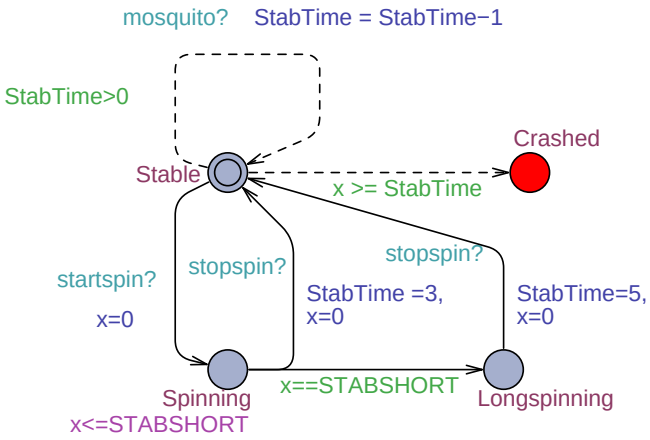

Fig. 1.6 A model for the plate.

The automaton uses one clock x. The use of x is twofold. First, when the control is in location `Stable`, the variable x records the time elapsed since the plate was last spun by the juggler. When the control is in `Spinning` or `Longspinning`, x records the time elapsed since the plate has last been spun under the impulsion of the juggler. Let us now have a look at the transitions between control locations.

First, we consider the uncontrollable transitions. There are two uncontrollable transitions that leave `Stable`. The self loop is taken whenever the mosquito touches the plate (this is ensured by the synchronization on the event `mosquito?`). The effect is to substract value 1 from the integer variable `StabTime` that models the length of the time interval during which the plate is guaranteed to stay stable without being spun by the juggler. This can be done only if the guard $\texttt{StabTime} > 0$ is true (making sure that the value of `StabTime` cannot become negative.)

The uncontrollable transition going from `Stable` to `Crashed` can be taken (by Player 2) whenever the value of the clock x exceeds the time for which the plate is guaranteed to be stable (since the last time it has been spun by the juggler.) As this transition is uncontrollable, Player 2 can decide to take it at any time when the guard is true. Player 2 may not take the transition immediately when the guard becomes true but we cannot rely on this: that is why it is an uncontrollable transition in our model.

Second, we consider the controllable transitions. The transition between locations `Spinning` and `Longspinning` is taken exactly when the value of x is equal to `STABSHORT`. It accounts for the fact that the juggler is spinning the plate for an interval of more than `STABSHORT` seconds. In fact, the behavior of this transition is deterministic and so it could have been defined as uncontrollable, that would not make any difference. The other three controllable transitions are related to actions controlled by the juggler. When the plate is `Stable`, the juggler can decide to give it more spinning by emitting the event `startspin!`. This has the effect to trigger this transition (reception of the event `startspin!`) and to move the control to location `Spinning`. The control leaves the location `Spinning`:

- either because the juggler has decided to stop spinning (event `stopspin?`) before `STABSHORT` seconds, in that case, the control moves back to location `Stable`, the clock x is reset, and the interval for which the plate is guaranteed to be stable is 3 seconds (update `StabTime=3`),
- or because the juggler has spun the plate for `STABSHORT` seconds, and the control moves to `Longspinning`. This later location is left when the event `stopspin?` occurs, in that case the control moves back to `Stable` and the plate is guaranteed to be stable for 5 seconds (update `StabTime=5`).

This template timed game automaton is instantiated twice, one time for `Plate 1` and one time for `Plate 2`.

We can now have a look at the other components of our model. Fig. 1.7 depicts a model of the mosquito. The mosquito can at any time touch one of the two plates provided that he has not touched a plate within the last D times units (this is forced by the guard $y \geq D$). This last constraint is enforced using the clock y which is reset each time a plate is touched. The self-loop is labeled with the event `mosquito!`

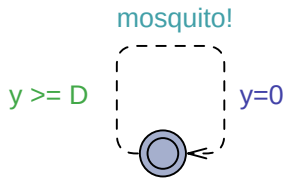

Fig. 1.7 A model for the mosquito

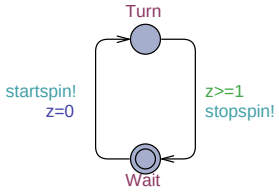

Fig. 1.8 A model for the juggler

which is either received by Plate 1 or Plate 2. The transition is uncontrollable as it belongs to the mosquito and not to the controller that we want to synthesize.

Finally, the juggler is modeled by the timed automaton given in Fig. 1.8. The juggler can be in two different states that are modeled by two locations: `Wait` models the situation when the juggler does not spin any of the two plates, `Turn` models the situation when the juggler spins one of the plates. The events `startspin!` and `stopspin!` are synchronized with either Plate 1 or Plate 2. Clock z is used to express that the juggler should spin a plate for at least 1 time unit.

1.7.2 Analysis of the model

We can now analyze the model of the Chinese Juggler presented above with the tool UPPAAL-TIGA. We want to determine if the Juggler has a strategy to win the timed game for the safety objective 'none of the two plates ever crashes"". This control objective is expressed by the following expression in the UPPAAL-TIGA syntax:

```
control: A[] not (Plate1.Crashed or Plate2.Crashed)
```

This formula asks to find a control strategy (keyword `control`) for the juggler such that on all resulting plays (modality `A`), it is always the case (modality [ ]) that `(Plate1.Crashed or Plate2.Crashed)` is false.

If we impose to the mosquito to stay away from the two plates for at least $D = 2$ seconds after touching one of the plates, then the Juggler has a strategy to win. UPPAAL-TIGA is able to determine that property, and furthermore, the tool also synthesizes a winning strategy. The strategy that the tool synthesizes is as follows:

.... Kim, can you provide a picture of the winning strategy for the parameters as described above ?

Now, if we set $D = 1$, then the Juggler does not have a strategy to win as the mosquito can act very fast.

1.8 Conclusion

In this chapter, we have introduced the basic concepts and algorithmic ideas that underly the automatic synthesis of discrete and timed controllers for systems modeled by game automata and timed game automata. We have shown that the game

metaphor is natural to model control problems. Even if those ideas are relatively recent, they have been implemented into the tool UPPAAL-TIGA and they can be applied to interesting case studies.

In the next chapter, we show how to use UPPAAL-TIGA to automatically synthesize a controller to regulate a pressure accumulator and to optimize its energy consumption.

References

1. Gerd Behrmann, Agnès Cougnard, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen, and Didier Lime. Uppaal-tiga: Time for playing games! In *CAV - International Conference on Computer Aided Verification*, volume 4590 of *Lecture Notes in Computer Science*, pages 121–125. Springer, 2007.
2. Franck Cassez, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen, and Didier Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR - International Conference Concurrency Theory*, volume 3653 of *Lecture Notes in Computer Science*, pages 66–80. Springer, 2005.
3. Oded Maler, Amir Pnueli, and Joseph Sifakis. On the synthesis of discrete controllers for timed systems. In E.W. Mayr and C. Puech, editors, *STACS - Theoretical Aspects of Computer Science*, volume 900 of *Lecture Notes in Computer Science*, pages 229–242. Springer-Verlag, 1995.
4. Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *POPL - Annual Symposium on Principles of Programming Languages*, pages 179–190. ACM Press, 1989.
5. Peter J. Ramadge and W. Murray Wonham. Supervisory control of a class of discrete-event processes. *SIAM Journal of Control and Optimization*, 25(1):206–230, 1987.