Timed Interfaces

Stefan Stattelmann

Seminar Games in Verification and Synthesis, Summer Term 2008

Tutor: Hans-Jörg Peter, Reactive Systems Group, Saarland University

1 Introduction

This text summarizes the theory of *timed interfaces* that can be used to model the interaction of components in a system with timing constraints, as described in [dAHS02].
Complex real-time systems are made of several interacting components. The interaction between one such component and its environment is described in terms of its interface. When two or more components are combined, this could cause problems due to unexpected interaction between the components, which might result in faulty behavior of the system or even a deadlock.
One classical approach for modelling such systems are *timed automata* ([AD94]). Such automata are basically automata for ω-regular languages that are augmented with one or more clocks that increase over time and can be tested and reset on transitions in the automaton. With this extension, a time value can be attached to every symbol in a word of the ω-regular language the automaton accepts. Such an automaton then can be used to model systems or parts of a system that have to fulfill timing constraints. When modelling the components of a system as timed automata and then combining those automata to model the whole system, some problems arise. It might be the case that at some point in time, one of the components *can* create an output for another component which can not be handled by this component. In a very strict, sense this would mean that the whole system is faulty, because it is possible to reach an erroneous state. Nevertheless, the system might still be useful and correct as long as the "user" can avoid that the system enters the faulty state by using the system "in the right way".
In terms of interfaces this means that achieving correctness of the combined interface might be achieved by making it more strict than the interfaces it is composed of, and thus make it safely usable, but less general.
The interface of a component can modeled as a game between two players Input and Output, where the component is represented by Player Output and its environment by Player Input. The moves of Player Input are the inputs that are accepted by the component (input assumptions) and the moves of Player Output stand for the outputs the system can generate (output guarantees). An interface is *well-formed* as long as there is *some* environment that can fulfill its the input assumptions.
This motivates the modelling of *timed interfaces* as *timed games*.

2 Timed Interfaces as Timed Games

Timed interfaces are modeled as timed games between two players, Input and Output, abbreviated I and O. Those players have two types of moves, *immediate moves* which model a send or receive event in the interface and *timed moves*, that represent an amount of time the player wants to wait for an event. If both players decide to play a timed move from the time domain $\mathbb{T}$ (e.g $\mathbb{T} = \mathbb{R}_{\geq 0}$ or $\mathbb{T} = \mathbb{N}$), global time advances by the smaller value. Immediate actions always prevail timed actions and if both players play an immediate move, one of them happens nondeterministically.
Only outcomes for which the global time diverges are meaningful for modeling interfaces because liveness is required, but one player could block progress by playing a sequence of timed moves that get smaller and smaller. To forbid this so-called Zeno behavior ([MPS95]), it has to be defined for each round of the game

which player can be blamed for playing first. So, if one player is playing all the time but global time does not diverge, this player blocks the game. This is also the case if a player runs out of moves to play.
For a timed interface to be well-formed, both players have to be able to win the game with the goal that time diverges or the other player can be made responsible for blocking the progress of time.

Definitions

A *timed interface* is a tuple $\mathcal{P} = (S_\mathcal{P}, s^{init}_\mathcal{P}, Acts^I_\mathcal{P}, Acts^O_\mathcal{P}, \rho^I_\mathcal{P}, \rho^O_\mathcal{P})$ such that:

- $S_\mathcal{P}$ is a set of *states*.
- $s^{init}_\mathcal{P} \in S_\mathcal{P}$ is the *initial state*.
- $Acts^I_\mathcal{P}$ and $Acts^O_\mathcal{P}$ are disjoint sets of *immediate input* and *immediate output* actions, respectively. Furthermore, $Acts_\mathcal{P} = Acts^I_\mathcal{P} \cup Acts^O_\mathcal{P}$ denotes the set of all immediate actions, $\Gamma^I_\mathcal{P} = Acts^I_\mathcal{P} \cup \mathbb{T}$ is the set of all input actions and $\Gamma^O_\mathcal{P} = Acts^O_\mathcal{P} \cup \mathbb{T}$ describes the set of all output actions. The elements in $\mathbb{T}$ are *timed actions*. $Acts^I_\mathcal{P}$, $Acts^O_\mathcal{P}$ and $\mathbb{T}$ have to be disjoint.
- $\rho^I_\mathcal{P} \subseteq S_\mathcal{P} \times \Gamma^I_\mathcal{P} \times S_\mathcal{P}$ is the *input transition* relation and $\rho^O_\mathcal{P} \subseteq S_\mathcal{P} \times \Gamma^O_\mathcal{P} \times S_\mathcal{P}$ the *output transition* relation. Both relations are required to be deterministic.

For $\gamma \in \{I, O\}$ the set $\Gamma^\gamma_\mathcal{P}(s) = \{\alpha \in \Gamma^\gamma_\mathcal{P} \mid \exists s' \in S_\mathcal{P}.(s, \alpha, s') \in \rho^\gamma_\mathcal{P}\}$ describes the set of *moves* for player γ in state s. If there are any moves available for player γ, she always can also take the timed action 0: $\Gamma^\gamma_\mathcal{P}(s) \neq \emptyset \implies (s, 0, s) \in \rho^\gamma_\mathcal{P}$.
For all states $s \in S_\mathcal{P}$ and moves $\alpha_I \in \Gamma^I_\mathcal{P}(s)$ and $\alpha_O \in \Gamma^O_\mathcal{P}(s)$ the *outcome* $\delta_\mathcal{P}(s, \alpha_I, \alpha_O) = (\alpha, s', bl)$ with $(s, \alpha, s') \in \rho^I_\mathcal{P} \cup \rho^O_\mathcal{P}$ and $bl \in \{I, O\}$ is defined as follows:

- If α_I, $\alpha_O \in \mathbb{T}$, then $\alpha = min\{\alpha_I, \alpha_O\}$ and $bl = I$ if $\alpha_I < \alpha_O$ or $bl = O$ otherwise. Note that this defintion is asymmetric to capture the cause-effect relation between inputs and outputs.
- If $\alpha_I \in Acts_\mathcal{P}$ and $\alpha_O \in \mathbb{T}$, then $\alpha = \alpha_I$ and $bl = I$.
- If $\alpha_I \in \mathbb{T}$ and $\alpha_O \in Acts_\mathcal{P}$, then $\alpha = \alpha_O$ and $bl = O$.
- If α_I, $\alpha_O \in Acts_\mathcal{P}$, then choose either $\alpha = \alpha_I$ and $bl = I$ or $\alpha = \alpha_O$ and $bl = O$ nondeterministically.

A *strategy* for player $\gamma \in \{I, O\}$ is a partial function $\pi^\gamma : S^*_\mathcal{P} \to \Gamma^\gamma_\mathcal{P}$ that assigns a move $\pi^\gamma(\bar{s}) \in \Gamma^\gamma_\mathcal{P}(s)$ to every finite sequence of states $\bar{s} \in S^*_\mathcal{P}$ whose final state is s, if $\Gamma^\gamma_\mathcal{P} \neq \emptyset$. Otherwise, $\pi^\gamma(\bar{s})$ is undefined.
Some state $s \in S_\mathcal{P}$ is *reachable* in $\mathcal{P}$ if there are two strategies π^I and π^O such that there is an outcome that visits s when the players play the game according to their strategies, starting from $s^{init}_\mathcal{P}$.

A timed interface is *well-formed* if for every reachable state player I has strategy such that time diverges or O is always blamed after some point in time and symmetrically, O has a strategy such that time diverges or I can be blamed for blocking time.

Product and composition of timed interfaces

Two timed interfaces $\mathcal{P}$ and $\mathcal{Q}$ are composable if they have no overlapping output actions. Their *shared actions* are defined as $shared(\mathcal{P}, \mathcal{Q}) = Acts_\mathcal{P} \cap Acts_\mathcal{Q}$. The composition $\mathcal{P} \parallel \mathcal{Q}$ is constructed in two steps: at first, compute the product $\mathcal{P} \otimes \mathcal{Q}$ and then modify the transitions so that no error states can be entered.
Given two composable timed interfaces $\mathcal{P}$ and $\mathcal{Q}$, their product $\mathcal{P} \otimes \mathcal{Q}$ is defined as follows:

- $S_{\mathcal{P}\otimes\mathcal{Q}} = S_\mathcal{P} \times S_\mathcal{Q}$ and $s^{init}_{\mathcal{P}\otimes\mathcal{Q}} = (s^{init}_\mathcal{P}, s^{init}_\mathcal{Q})$.
- $Acts^I_{\mathcal{P}\otimes\mathcal{Q}} = Acts^I_\mathcal{P} \cup Acts^I_\mathcal{Q} \setminus shared(\mathcal{P}, \mathcal{Q})$ and $Acts^O_{\mathcal{P}\otimes\mathcal{Q}} = Acts^O_\mathcal{P} \cup Acts^O_\mathcal{Q}$.

- $\rho^I_{\mathcal{P}\otimes\mathcal{Q}}$ is the set of transitions $((s_1,s_2),\alpha,(s_1',s_2')$ such that if $\mathcal{P}$ can play the immediate input action α from state s_1 to s_1', $\mathcal{Q}$ must play the timed action 0. Symmetrically, if $\mathcal{Q}$ can play $\alpha \in Acts^I_{\mathcal{Q}}$, $\mathcal{P}$ must play the timed action 0:

  $\rho^I_{\mathcal{P}\otimes\mathcal{Q}} = \{((s_1,s_2),\alpha,(s_1',s_2')|(s_1,\alpha,s_1') \in \rho^I_{\mathcal{P}} \; and \; (s_2,0,s_2') \in \rho^I_{\mathcal{Q}}\} \cup$
  $\{((s_1,s_2),\alpha,(s_1',s_2')|(s_2,\alpha,s_2') \in \rho^I_{\mathcal{Q}} \; and \; (s_1,0,s_1') \in \rho^I_{\mathcal{P}}\}$

- $\rho^O_{\mathcal{P}\otimes\mathcal{Q}}$ is the set of transitions $((s_1,s_2),\alpha,(s_1',s_2')$ such that if $\mathcal{P}$ can play the immediate output action α from state s_1 to s_1', $\mathcal{Q}$ can either play the immediate *input* action α from s_2 to s_2' or the timed action 0. Symmetrically, if $\mathcal{Q}$ can play output move α, $\mathcal{P}$ can either play input move α or play the timed action 0. If it is possible for the players to synchronize a move, they have to:

  $\rho^O_{\mathcal{P}\otimes\mathcal{Q}} =$
  $\{((s_1,s_2),\alpha,(s_1',s_2')|(s_1,\alpha,s_1') \in \rho^O_{\mathcal{P}} \; and \; (s_2,\beta,s_2') \in \rho^I_{\mathcal{Q}}, \beta = \alpha \; if \; \alpha \in Acts^I_{\mathcal{Q}} \; or \; \beta = 0 \; otherwise\} \cup$
  $\{((s_1,s_2),\alpha,(s_1',s_2')|(s_2,\alpha,s_2') \in \rho^O_{\mathcal{Q}} \; and \; (s_1,\beta,s_1') \in \rho^I_{\mathcal{P}}, \beta = \alpha \; if \; \alpha \in Acts^I_{\mathcal{P}} \; or \; \beta = 0 \; otherwise\}$

The product of two interfaces might contain error states. There are two kinds of error states. If in some state one of the interfaces can produce an output that cannot be accepted by the other interfaces, this state is an *immediate error state.* To get a well-formed composition of two interfaces and to guarantee safety of the composed system, it is not only necessary to stay out of the immediate error states, but it is also necessary to avoid entering all states from which the Input Player does not have a winning strategy anymore after removing the immediate error states. The composition $\mathcal{P} \parallel \mathcal{Q}$ can be obtained by restricting the input behavior of $\mathcal{P}\otimes\mathcal{Q}$ in such a way that the error states are never entered. The intuition behind this is that when combining two components to form a new interface, it is not possible to modify the behavior of the original components. Nevertheless, it is possible to restrict the way the new component can be used or, in other words, which moves player I is allowed to make.

- *Immediate error states*: A state $(s,t) \in S_{\mathcal{P}\otimes\mathcal{Q}}$ is an *immediate error state* if there is an shared action $\alpha \in shared(\mathcal{P},\mathcal{Q})$ such that $\exists s' : (s,\alpha,s') \in \rho^O_{\mathcal{P}}$ and $\forall t' : (t,\alpha,t') \notin \rho^I_{\mathcal{Q}}$ or $\exists t' : (t,\alpha,t') \in \rho^O_{\mathcal{Q}}$ and $\forall s' : (s,\alpha,s') \notin \rho^I_{\mathcal{P}}$. The set of all immediate error states is denoted by $i\text{-}errors(\mathcal{P},\mathcal{Q}) \subseteq S_{\mathcal{P}\otimes\mathcal{Q}}$.

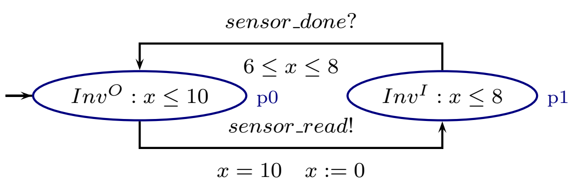
- *Time error states*: A state $(s,t) \in S_{\mathcal{P}\otimes\mathcal{Q}}$ is a *time error state* if it is reachable in $\mathcal{P}\otimes\mathcal{Q}$, but there is no strategy to win the game for player I in $S_{\mathcal{P}\otimes\mathcal{Q}} \setminus i\text{-}errors(\mathcal{P},\mathcal{Q})$. The set of all time error states is denoted by $t\text{-}errors(\mathcal{P},\mathcal{Q})$ with $i\text{-}errors(\mathcal{P},\mathcal{Q}) \subseteq t\text{-}errors(\mathcal{P},\mathcal{Q}) \subseteq S_{\mathcal{P}\otimes\mathcal{Q}}$.

Two well-formed and composable interfaces $\mathcal{P}$ and $\mathcal{Q}$ are *compatible* if $(s^{init}_{\mathcal{P}}, s^{init}_{\mathcal{Q}}) \notin t\text{-}errors(\mathcal{P},\mathcal{Q})$. The composition $\mathcal{P} \parallel \mathcal{Q}$ is then defined like $\mathcal{P}\otimes\mathcal{Q}$, except that the input transition relation is restricted to $\rho^I_{\mathcal{P}\parallel\mathcal{Q}} = \rho^I_{\mathcal{P}\otimes\mathcal{Q}} \cap (U \times Acts^I_{\mathcal{P}\otimes\mathcal{Q}} \times U)$ with $U = S_{\mathcal{P}\otimes\mathcal{Q}} \setminus t\text{-}errors(\mathcal{P},\mathcal{Q})$.

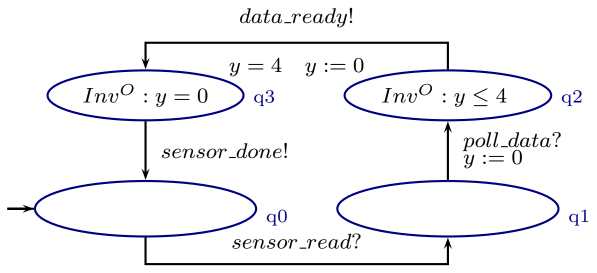
3 Timed Interface Automata

Timed interface automata are a different representation for *timed interfaces.* They are very similar to timed automata ([AD94]) and many algorithms for checking properties of timed automata also work for timed interface automata.
Timed interface automata use *clock variables* to model progress of time. Those variables can be tested and reset during transitions in the automaton to model timing constraints of the interface. Furthermore, in a timed interface automaton it is possible to specify *input* and *output invariants* for Player I and O respectively. These invariant are boolean formulas over clock values (also called *clock conditions*) that can express timing requirements for the moves of one player, namely how long the player is allowed to let time progress until she has to play a move. In other words, as long as the invariant for one player is true, she is allowed to do nothing but she has do something before her invariant gets false or she will be blamed for blocking time and hence, will lose the game.

Example



(a) SensorUser



(b) SensorControlUnit

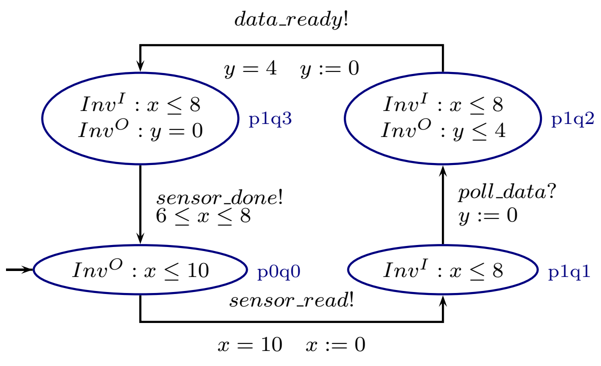
Figure 1: Two interfaces for sensor access

Consider the timed interface automata pictured in Figure 1. The first one describes a user interfaces which periodically requests data from a sensor by sending the output event *sensor_read*. The request has to be completed by the environment through sending the input event *sensor_done* to the interface within 8 time units. Note how the input/output invariants within the states of the automaton describe which moves are allowed for Players I and O in each when interpreting the interface as a game. No invariant means that a player is allowed to play any move, in particular she can also let time diverge. The meaning of the ouput invariant in state $p0$ is that Player O (a component with interface SensorUser) has to play an immmediate move at least every 10 time units, which in this case means request data from the sensor. Otherwise, Player O loses the game which means the interface specification is violated. Similarly, Player I (other components that are connected with SensorUser) has to reply to this request within some timeframe.
The second interface SensorControlUnit waits for the input signal *sensor_read* and then starts polling data for 4 time units at some point in time. After that, it acknowledges that data has been collected with the output signal *sensor_done*. Consider the two possible combinations of those interfaces in Figure 2. If the product SensorUser ⊗ SensorControlUnit is used to model the combined interface of both components, the combination might reach a faulty state if the SensorControlUnit starts polling data too early or too late. This is due to the fact that the product interface does not model the interaction between the two clock variables of the original interfaces. Namely, if SensorControlUnit starts its clock y before clock variable x of SensorUser has been running for at least 2 time units, SensorControlUnit will be done before SensorUser can process the result. Similarly, if SensorControlUnit starts y after the value of x is greater than 4, when SensorControlUnit is done it will be to late for SensorUser to accept the input.
Since the transition that starts the polling of data is triggered by the input event *poll_data*, it is possible to fix this problem by requiring the Input Player to play the move *poll_data* in the right time frame. In other words, the interface is restricted in a way such that no error states are entered. The resulting combined interface with more restrictive transitions is shown in Figure 2b.

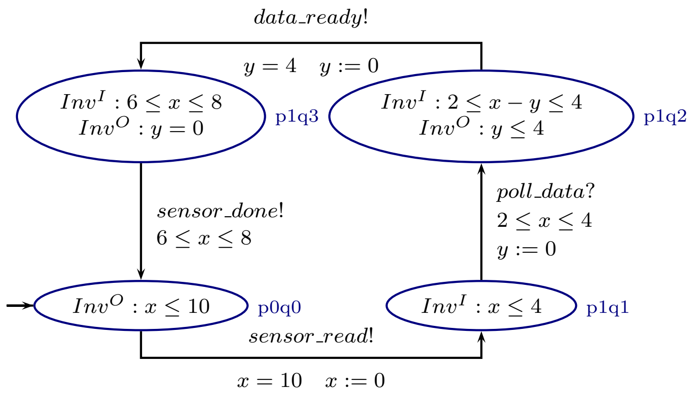
Definition

A *timed interface automaton* is a tuple $\mathcal{A} = (Q_\mathcal{A}, q^{init}_\mathcal{A}, \mathcal{X}_\mathcal{A}, Acts^I_\mathcal{A}, Acts^O_\mathcal{A}, Inv^I_\mathcal{A}, Inv^O_\mathcal{A}, \rho_\mathcal{A})$ such that:

- $Q_\mathcal{A}$ is a finite set of *locations* (states of the automaton).
- $q^{init}_\mathcal{A} \in Q_\mathcal{A}$ is the *initial location.*
- $\mathcal{X}_\mathcal{A}$ is the finite set of *clocks* of the automaton.



(a) SensorUser ⊗ SensorControlUnit



(b) SensorUser ∥ SensorControlUnit

Figure 2: Combined Interfaces

- $Acts^I_{\mathcal{A}}$ and $Acts^O_{\mathcal{A}}$ are finite and disjoint sets of *input* and *output actions*. $Acts_{\mathcal{A}} = Acts^I_{\mathcal{A}} \cup Acts^O_{\mathcal{A}}$ describes the set of all actions in $\mathcal{A}$.
- $Inv^I_{\mathcal{A}} : Q_{\mathcal{A}} \mapsto \mathcal{C}[\mathcal{X}_{\mathcal{A}}]$ and $Inv^O_{\mathcal{A}} : Q_{\mathcal{A}} \mapsto \mathcal{C}[\mathcal{X}_{\mathcal{A}}]$ map an *input/output invariant* to each location in $\mathcal{A}$, where $\mathcal{C}[\mathcal{X}_{\mathcal{A}}]$ is the set of clock conditions over clocks in $\mathcal{A}$.
- $\rho_{\mathcal{A}} \subseteq Q_{\mathcal{A}} \times \mathcal{C}[\mathcal{X}_{\mathcal{A}}] \times Acts_{\mathcal{A}} \times 2^{\mathcal{X}_{\mathcal{A}}} \times Q_{\mathcal{A}}$ is the *transition relation*. A transition $(p, g, a, r, q') \in \rho_{\mathcal{A}}$ from location p to location p' can only be taken with action a if the current values of the clock variables fulfil the clock condition g. During the transition, all clocks in r are set to 0.

A timed interface automaton $\mathcal{A}$ is *nonempty* if its initial state fulfils its input and output invariants when all clock variables are set to 0. A nonempty timed interface automaton *induces* a timed interface $\mathcal{P} = [\![\mathcal{A}]\!]$ with

- $S_{\mathcal{P}} = \{\langle p, v\rangle \mid p \in Q_{\mathcal{A}}, v \in \mathcal{X}_{\mathcal{A}} \mapsto \mathbb{T}\}$
- $s^{init}_{\mathcal{P}} = \langle q^{init}_{\mathcal{A}}, f(x) = 0\rangle$
- $Acts^I_{\mathcal{P}} = Acts^I_{\mathcal{A}}$ and $Acts^O_{\mathcal{P}} = Acts^O_{\mathcal{A}}$
- Transitions are such that Player I or O can let time pass (and increase the clocks in the interface state) as long as $Inv^I_{\mathcal{A}}$ or $Inv^O_{\mathcal{A}}$ is valid in the corresponding automaton state, respectively. Furthermore, the transitions $\rho_{\mathcal{A}}$ of $\mathcal{A}$ are translated to transitions $\rho^I_{\mathcal{P}}$ and $\rho^O_{\mathcal{P}}$ in $\mathcal{P}$ in the obvious way such that each player has to obey the invariants and reset the clocks encoded in the interface state according to the original transition in $\mathcal{A}$.

A timed interface automata is $\mathcal{A}$ *well-formed* if it is nonempty and its induced interface $[\![\mathcal{A}]\!]$ is well-formed. Product and composition of timed interface automata can be defined very similar to the way it is defined for timed interfaces. Two well-formed timed interface automata $\mathcal{A}$ and $\mathcal{B}$ are composable if they do not share any output actions or clock variables. For such automata the product can be defined similar to timed interfaces and it holds that $[\![\mathcal{A} \otimes \mathcal{B}]\!] = [\![\mathcal{A}]\!] \otimes [\![\mathcal{B}]\!]$. Furthermore, well-formed and composable automata $\mathcal{A}$ and $\mathcal{B}$ are

said to be compatible if their corresponding interfaces are compatible and, in this case, the equivalence $[\![\mathcal{A} \parallel \mathcal{B}]\!] = [\![\mathcal{A}]\!] \parallel [\![\mathcal{B}]\!]$ holds.
The representation of timed interfaces as timed interface automata has the advantage that existing algorithms for timed automata can be used to calculate reachable states and to check the well-formedness of the interface.

4 Algorithms for Timed Games

Deciding for which states player Input has a winning strategy is possible by reducing the timed game to an untimed game with an ω-regular goal. Consider the $Tick$ automaton in Figure 3. For every timed interface automata $\mathcal{A}$, we can build the automaton $\mathcal{A} \otimes Tick$. Player I has a strategy in $[\![A]\!]$ to win with the goal that time diverges or player O can be blamed for blocking, iff there is a strategy in $[\![\mathcal{A} \otimes Tick]\!]$ such that q_1 is visited infinitely often or Player Output can be blamed for blocking progress. This property can be checked with existing algorithms for solving games. The same construction can be used to determine the states where Player Output has a winning strategy.
The *region graph* of a timed interface automata can be defined in the same way as for timed automata. Since the reachable locations can be calculated on the region graph, the reachable states of a timed interface automaton can be calculated in the same way as for a timed automata [AD94].
Checking the well-formedness of an timed interface automaton $\mathcal{A}$ can therefore be done, with the techniques described above, by testing if the reachability of a location implies that Input and Output have a winning strategy from there.

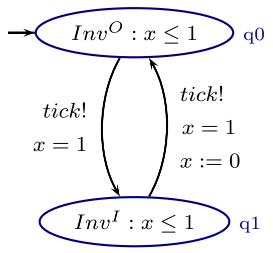


Figure 3: $Tick$ automaton

5 Summary

Timed interfaces offer a more optimistic way to model the interaction of components in a complex system. In contrast to other approaches for modeling such systems, like timed automata, the composition of timed interfaces can handle and avoid error states a system could enter due to badly synchronized component interaction, by restricting such actions. This approach is optimistic because it assumes that the less general system is still useful in some context. The crucial new idea is to model component interaction as game between Player Output (the component) and Player Input (environment), which makes it possible to enforce safety and liveness properties by modifying the possible moves of Player Input without changing the component. The representation of timed interfaces as timed interface automata allows the reuse of existing algorithms to check the properties of interfaces. Therefore timed interfaces provide an elegant new approach to interface theory.

References

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[dAHS02] L. de Alfaro, T. Henzinger, and M. Stoelinga. Timed interfaces, 2002.

[MPS95] Oded Maler, Amir Pnueli, and Joseph Sifakis. On the synthesis of discrete controllers for timed systems (extended abstract), 1995.