# Reductions and Abstractions for Optimization of Modular Timed Automata

Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnLine 55-28 (2022) 344–349

# Reductions and Abstractions for Optimization of Modular Timed Automata [★]

**Bengt Lennartson** [*]

[*] *Division of Systems and Control, Chalmers University of Technology, SE-412 96 Göteborg, Sweden (e-mail: bengt.lennartson@chalmers.se).*

Abstract: Time optimization of concurrent sequences of operations is in this paper solved by timed automata. To reduce the complexity of this classical problem, including applications such as planning and scheduling, an abstraction method has recently been proposed based on local optimization (Hagebring and Lennartson, 2019). In a modular subsystem, local paths without any communication with other subsystems are optimized with respect to time, and when subsystems are synchronized more local behavior appears. The proposed method has shown to be successful, drastically reducing computational complexity for important classes of planning problems. The only drawback is that the synchronous composition includes a heuristic non-standard synchronous composition procedure to achieve true concurrency. In this paper a simple solution to this problem is presented based on the original synchronous composition of timed automata. In the transformation of the timed automaton to an ordinary automaton, where time weights are generated, it is first observed that the state space often increases dramatically in this transformation. To solve this complexity problem, an efficient reduction is proposed as a complement to local optimization, and both methods are demonstrated to be very efficient when they are applied to realistic benchmark examples.

*Keywords:* Timed systems, timed automata, optimization, automata, modular systems.

## 1. INTRODUCTION

Time-optimal control and synthesis of time-optimal supervisors (Su et al., 2012) are mainly formulated based on time weighted automata, often based on tick automata models (Ware and Su, 2017). Abstractions have also been proposed (Hill and Lafortune, 2016, 2017), recently including even weak bisimulation (Vilela and Hill, 2022). This means that invisible $\tau$ transitions can be removed, but some restrictions are introduced which sometimes make such abstractions less powerful for timed transitions.

Hagebring has instead proposed an abstraction based on local optimization, where only the best (minimal time) local paths are preserved, while invisible tau transitions are also abstracted in the same way as in for instance branching bisimulation (Van Glabbeek and Weijland, 1996). This local optimization abstraction was first developed for tick automata models (Hagebring and Lennartson, 2018), but has also been defined for continuous time models where time is represented by real variables (Hagebring and Lennartson, 2019).

This framework was formulated for time weighted automata, but not based on the original formulation in (Su et al., 2012), where concurrency is separated from the time weighted automaton by only considering the starting time of the events. This means that the time duration of operations is hidden in the weighted automaton. To obtain the local optimization, Hagebring includes an explicit concurrency in the automata model by a special heuristic formulation of the synchronous composition. This formulation is very efficient but is for special cases not well defined.

In this paper we therefore propose an alternative model framework, based on the ordinary synchronous composition of timed automata (Alur and Dill, 1994; Baier and Katoen, 2008). The start of an operation is then modeled by an ordinary zero-time event, while a separate transition condition includes a clock constraint on the completion time of an operation. Since this clock constraint can be combined with the start condition of the next operation, the same number of transitions is achieved as in earlier proposed methods, including the heuristic method in (Hagebring and Lennartson, 2019).

Generally, timed automata do not include the full state information, where the values of the clocks also need to be added to get the complete state. This problem is analyzed in the paper, and it is shown that the transformation to a time weighted model often increases the state space significantly. To solve this complexity problem a specific reduction method is proposed which is able to remove most of the additional states that are generated due to the incorporation of the clocks in the total state vector.

The main contribution of this paper is that the very efficient compositional time optimization approach in Hagebring and Lennartson (2019) is further improved and simplified. More specifically a powerful reduction method is proposed in the transformation to a time weighted automaton, which is required to obtain the time-optimal path. The method is flexible in the sense that both state and transition labels can be included in the model, and it can be applied also to Petri nets including shared variables (Lennartson et al., 2014). The proposed model is presented in Section 3, and in Section 4 the reduction method related to the clock states is presented, while optimization based abstractions are presented in Section 5.

## 2. TIMED AUTOMATA

In a *timed automaton* (Alur and Dill, 1994), time is introduced by a set of clocks $C$, where constraints on these clocks are used to specify time conditions. An operation time greater or equal $\Delta$ can be specified as $c \geq \Delta$, assuming that the clock

[★] This work was supported by SyTec – Systematic Testing of Cyber-Physical Systems, a Swedish Science Foundation grant for strong research environment and Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation. The support is gratefully acknowledged.

10.1016/j.ifacol.2022.10.364

$c \in C$ is reset to zero when the operation starts. To be able to define a timed automaton, let $\Phi(C)$ denote the set of clock constraints over $C$. The reset operation is here also incorporated as a constraint on the immediate clock value after a transition.

*Definition 1.* A *timed automaton* $G$ is defined by a 7-tuple $G = \langle L, \Sigma, C, T, L_0, AP, \lambda\rangle$ where $L$ is a finite set of locations, $\Sigma$ is a finite set of events, $C$ is a finite set of clocks, $T \subseteq L \times \Sigma \times \Phi(C) \times L$ is a transition relation where $t = (\ell, a, \varphi, \ell') \in T$ includes the source location $\ell$, the event label $a$, the clock constraint $\varphi \in \Phi(C)$, and the target location $\ell'$ of the transition $t$, $L_0 \subseteq L$ is a set of possible initial locations, $AP$ is a set of atomic propositions, and $\lambda : L \to 2^{AP}$ is a state labelling function. A transition $(\ell, a, \varphi, \ell')$ is also denoted $\ell \xrightarrow{a:\varphi} \ell'$. □

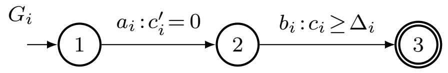


Figure 1. Timed automata $G_i$, $i = 1, 2$ where the clock $c_i$ is reset when the event $a_i$ occurs, and a time delay greater or equal $\Delta_i$ must pass before event $b_i$ can occur.

*Limited set of clock constraints* Focusing on time optimization of concurrent sequences of operations, only a limited set of clock constraints is required. In Fig. 1, an operation starts when the event $a_i$ occurs. The clock $c_i$ is then reset to zero by the *next clock value constraint* $c_i' = 0$ (next value is denoted by prime). The clock constraint $c_i \geq \Delta_i$ specifies that the duration of the operation is at least $\Delta_i$ before the event $b_i$ occurs, specifying that the operation is completed. An equality constraint $c_i = \Delta_i$ is often enough, but when subsystems are synchronized, this may be too restrictive. The reason is that if $b_i$ is a shared event, this results in a conjunction of clock constraints that may lead to further delays caused by other subsystems. Based on this discussion, the only clock constraints that are required in time optimization are included in the following definition.

*Definition 2.* For a finite set of clocks $C$, the clock constraints $\varphi \in \Phi(C)$ required in time optimization are inductively defined by the grammar

$$\varphi ::= c \geq \Delta \;\big|\; c' = 0 \;\big|\; \varphi_1 \wedge \varphi_2$$

where $c \in C$, $\Delta \in \mathbb{R}_{\geq 0}$, $c \geq \Delta$ [1] specifies that the value of the clock $c$ is greater or equal $\Delta$, $c' = 0$ specifies that the next value of $c$ after a transition is reset to zero, and $\varphi_1 \wedge \varphi_2$ is the conjunction of the clock constraints $\varphi_1$ and $\varphi_2$. □

## 2.1 Synchronous composition of timed automata

One strength of timed automata is the ability to easily define the composition of timed subsystems. The simplicity comes from the fact that the timing information is separated from the location nodes, where the actual time is expressed symbolically by predicate logic expressions (the clock constraints).

The synchronous composition of two timed automata (Baier and Katoen, 2008; Hoare, 1978) is here adapted to hidden $\tau$ events, where such events in different subsystems are not synchronized, although they share the same event label $\tau$. They are simply considered as local events, since the hiding mechanism where an event is replaced by the invisible $\tau$ event is only applied to local events.

*Definition 3.* Let $G_i = \langle L_i, \Sigma_i, C_i, T_i, L_{0i}, AP_i, \lambda_i\rangle$, $i = 1, 2$, be two timed automata. The synchronous composition of $G_1$ and $G_2$ is then defined as

$$G_1 \,\|\, G_2 = \langle L_1 \times L_2, \Sigma_1 \cup \Sigma_2, T, L_{01} \times L_{02}, AP_1 \cap AP_2, \lambda\rangle$$

where $\lambda : L_1 \times L_2 \to 2^{AP_1 \cap AP_2}$ and

$$(\ell_1, \ell_2) \xrightarrow{a:\varphi_1 \wedge \varphi_2} (\ell_1', \ell_2') \in T : a \in (\Sigma_1 \cap \Sigma_2) \setminus \{\tau\},$$
$$\ell_1 \xrightarrow{a:\varphi_1} \ell_1' \in T_1,\; \ell_2 \xrightarrow{a:\varphi_2} \ell_2' \in T_2,$$
$$(\ell_1, \ell_2) \xrightarrow{a:\varphi_1} (\ell_1', \ell_2) \in T : a \in (\Sigma_1 \setminus \Sigma_2) \cup \{\tau\},\; \ell_1 \xrightarrow{a:\varphi_1} \ell_1' \in T_1,$$
$$(\ell_1, \ell_2) \xrightarrow{a:\varphi_2} (\ell_1, \ell_2') \in T : a \in (\Sigma_2 \setminus \Sigma_1) \cup \{\tau\},\; \ell_2 \xrightarrow{a:\varphi_2} \ell_2' \in T_2.$$
□

A more general definition of composed state labels is given in (Lennartson and Jia, 2020), where both disjunctive and conjunctive state labels are accepted. Examples of disjunctive state labels are forbidden states where the union is taken, while marked states are conjunctive, meaning that the intersection is applied as in Def. 3. In this paper, marked state labels will mainly be used, see the double circle in Fig. 1. Introducing more arbitrary state labels can be used to specify more complex state based properties, including for instance temporal logic specifications.

*Example 1.* The synchronous composition $G_1 \,\|\, G_2$ of the two timed automata in Fig. 1 is shown in Fig. 2. Since all events are local, no synchronized events are involved, and the result is an interleaving of the transitions in the individual timed automata, including their transition labels. □

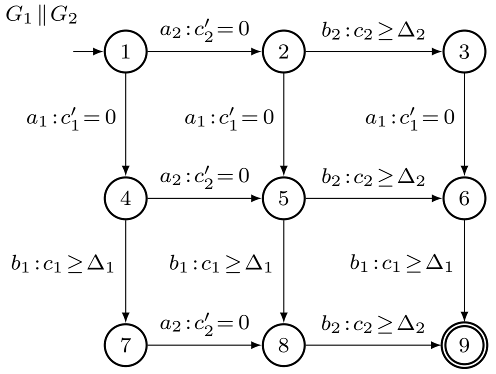


Figure 2. Synchronous composition of the two timed automata in Fig. 1.

*True concurrency* In order to obtain a natural concurrency, every event including a time delay is separated in two transitions. It means that concurrent operations can be started at any time, as the start events $a_1$ and $a_2$ in Fig. 2. When one operation is started by say event $a_1$, the event $a_2$ can be executed already in location 3. It does not need to wait until the operation delay $\Delta_1$ has passed and the first operation has reached location 6. This separation of the start event and its duration in two transitions naturally introduces the required concurrency. In Hagebring and Lennartson (2019) events and their time durations are joined together in time weighted transitions and a heuristic synchronization procedure is introduced to still obtain concurrent operations. The operation model proposed in this paper achieves concurrent operations by an ordinary synchronous composition, without any additional complications. In (Su et al., 2012), concurrency is separated from the time weighted automaton by only considering the starting time of the events, meaning that the time duration of operations is hidden in the weighted automaton.

## 3. TIME OPTIMIZATION

Optimization with respect to time means that we are searching for the shortest time to reach any marked location $\ell \in L_m$ from

[1] A more correct notation is to introduce a valuation function $v : C \to \mathbb{R}_{\geq 0}$ and write $v(c) \geq \Delta$. If $c$ means the variable or its value is, however, clear from the context.

a given initial state. To be able to represent the desired time-optimal solution as a sequence of events, it is assumed that the timed automaton $G$ is deterministic. This means that the execution of a string of events from a given initial state results in a unique location $\ell \in L$.

### 3.1 States including clocks

The actual time when a location $\ell_k$ is reached depends on the path of locations that has been taken to reach this location from the initial location $\ell_0$. Such a path, $\rho_k = (\ell_0, \ell_1, \ldots, \ell_k)$, is a member of a set $\Upsilon(\ell_k)$, including all paths such that $\ell_k$ can be reached from $\ell_0$.

*Entry time clock vector* For a given path $\rho_k$, the minimal time to reach location $\ell_k$ is called the *minimal location time*, and it is denoted $t^*(\rho_k)$. This time depends on clock constraints according to Def. 2, where the values of each clock $c_i \in C$ in location $\ell_k$ are collected in a *clock vector*

$$\boldsymbol{c}(\ell_k) = \big[c_1(\ell_k), \ldots, c_n(\ell_k)\big].$$

To be more precise, some clocks can be reset when location $\ell_k$ is reached. Following the notation in Def. 2, see also Fig. 1, the value of the clocks when location $\ell_k$ has been reached, and specified clocks have been reset, is denoted $\boldsymbol{c}'(\ell_k)$. Indeed, this is the entry time of the clocks in location $\ell_k$, and $\boldsymbol{c}'(\ell_k)$ is called the *entry time clock vector* for location $\ell_k$.

*Minimal waiting time* To obtain the minimal location time $t^*(\rho_k)$, all time delay constraints must be reduced to a minimum, still satisfying all specified clock constraints. The minimal time one has to wait in location $\ell_k$ before a transition can occur to a location $\ell_{k+1}$ is called *minimal waiting time* $w$. This minimal waiting time depends on the entry time clock $\boldsymbol{c}'(\ell_k)$, and the clock constraint $\varphi(\ell_k, \ell_{k+1})$ for the transition from $\ell_k$ to $\ell_{k+1}$.

In the following proposition, an expression for the minimal waiting time is given for a generic clock constraint $\varphi(\ell_k, \ell_{k+1})$, as well as updates of the minimal waiting time $t^*$ and the entry time clock vector $\boldsymbol{c}'$. For time optimization, this proposition summarizes the required time update of a timed automaton.

*Proposition 1.* For a transition from location $\ell_k$ to $\ell_{k+1}$, consider the clock constraint

$$\varphi(\ell_k, \ell_{k+1}) = \bigwedge_{i \in I_d(\ell_k, \ell_{k+1})} c_i(\ell_k) \geq \Delta_i(\ell_k, \ell_{k+1}) \wedge \bigwedge_{i \in I_r(\ell_k, \ell_{k+1})} c'_i(\ell_{k+1}) = 0,$$

where $I_d(\ell_k, \ell_{k+1})$ is the index set of the clocks involved in time delay clock constraints, and $I_r(\ell_k, \ell_{k+1})$ is the index set for the clocks to be reset when location $\ell_{k+1}$ is reached. The *minimal waiting time* is then

$$w(\ell_k, \ell_{k+1}) = \max_{i \in I_d(\ell_k, \ell_{k+1})} \big(\Delta_i(\ell_k, \ell_{k+1}) - c'_i(\ell_k), 0\big)$$

and

$$t^*(\rho_{k+1}) = t^*(\rho_k) + w(\ell_k, \ell_{k+1})$$

$$c'_i(\ell_{k+1}) = \begin{cases} c'_i(\ell_k) + w(\ell_k, \ell_{k+1}) & i \notin I_r \\ 0 & i \in I_r \end{cases}$$

*Proof:* Follows directly by the fact that $c_i(\ell_k) \geq c'_i(\ell_k) + w(\ell_k, \ell_{k+1})$ must be valid for all $i \in I_d(\ell_k, \ell_{k+1})$ before a transition to location $\ell_{k+1}$ is admissible. □

*Complete state and timed transition* Since the entry time clocks $\boldsymbol{c}'(\ell_k)$ may be different, depending on which path has been taken to reach location $\ell_k$, the complete state is defined by the actual combination of location and entry time clock values. In other words, for a given location $\ell_k$, the explicit system state is $(\ell_k, \boldsymbol{c}'(\ell_k))$. The minimal waiting time $w(\ell_k, \ell_{k+1})$ for a given state $(\ell_k, \boldsymbol{c}'(\ell_k))$ also defines a *timed transition*

$$(\ell_k, \boldsymbol{c}'(\ell_k)) \xrightarrow{w(\ell_k, \ell_{k+1})} (\ell_{k+1}, \boldsymbol{c}'(\ell_{k+1}))$$

### 3.2 Time-optimal path

The path to the marked location $\ell_m \in L_m$ with the shortest minimal location time $t^*$ is the *time-optimal path* $\rho^*$, i.e.

$$\rho^* = \arg\min_{\rho_m} t^*(\rho_m)$$

where $\rho_m \in \Upsilon(\ell_m)$ and $\ell_m \in L_m$. The minimal location time $t^*(\rho^*)$ that generates the time-optimal path is called *minimal makespan*. The *time-optimal sequence of events* $\mu \in \Sigma^*$ is finally obtained by observing the events in transitions between the locations in the time optimal path $\rho^*$ in the original timed automaton $G$.

Due to the recursive update of the minimal location time in Prop. 1, the time-optimal solution can also be expressed as

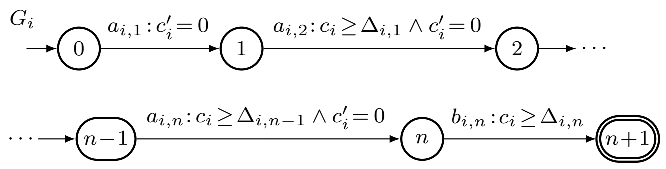
$$\rho^* = \arg\min_{\rho_m} \sum_{k=0}^{N-1} w(\ell_k, \ell_{k+1})$$

where $\rho_m \in \Upsilon(\ell_m)$ and $\ell_N = \ell_m \in L_m$. The minimal waiting times can then be regarded as weights $w(\ell_k, \ell_{k+1})$ and the timed automaton has been extended to a weighted ordinary automaton, where the shortest path gives the optimal solution. On the other hand, the weights $w(\ell_k, \ell_{k+1})$ need to be computed, and doing that following Prop. 1 gives at the same time the time-optimal solution. The following example illustrates how additional states are obtained due to different paths to the final state 9 in the timed automaton in Fig. 2, at the same time as the time optimal solution is generated.

*Example 2.* When minimal waiting times are applied in the synchronous composition $G_1 \,\|\, G_2$ in Fig. 2, passing location 3 gives $\boldsymbol{c}'(3) = (\Delta_2, \Delta_2)$, $\boldsymbol{c}'(6) = (0, \Delta_2)$, and $\boldsymbol{c}'(9) = (\Delta_1, \Delta_1 + \Delta_2)$, and passing location 7 gives symmetrically $\boldsymbol{c}'(7) = (\Delta_1, \Delta_1)$, $\boldsymbol{c}'(8) = (\Delta_1, 0)$, and $\boldsymbol{c}'(9) = (\Delta_1 + \Delta_2, \Delta_2)$. In both cases the makespan is $\Delta_1 + \Delta_2$, while the optimal path has the minimal makespan $\Delta_{12} = \max(\Delta_1, \Delta_2)$ passing location 5 and either 6 or 8 with $\boldsymbol{c}'(6) = (\Delta_2, \Delta_2)$, $\boldsymbol{c}'(8) = (\Delta_1, \Delta_1)$, and $\boldsymbol{c}'(9) = (\Delta_{12}, \Delta_{12})$. Thus, three different clock values are obtained in the final location 9, either $\boldsymbol{c}'(9) = (\Delta_1, \Delta_1 + \Delta_2)$ passing location 3, $\boldsymbol{c}'(9) = (\Delta_1 + \Delta_2, \Delta_2)$ passing location 7, or $\boldsymbol{c}'(9) = (\Delta_{12}, \Delta_{12})$ passing location 5. □

## 4. REDUCTION OF CLOCK STATES

In this section we will show that the number of states often increases dramatically when a timed automaton is transformed to a time-weighted automaton and the complete states including clock values are computed. This is first illustrated by an example with two arbitrarily large sequences of concurrent operations. A simple solution to this problem is then presented, where the true state space including clocks often can be reduced to a size similar to the number of locations in the timed automaton. The proposed reduction exploits the fact that the time-optimal solution is requested.



$G_i$
$0$
$a_{i,1}: c_i' = 0$
$1$
$a_{i,2}: c_i \geq \Delta_{i,1} \wedge c_i' = 0$
$2$
$n-1$
$a_{i,n}: c_i \geq \Delta_{i,n-1} \wedge c_i' = 0$
$n$
$b_{i,n}: c_i \geq \Delta_{i,n}$
$n+1$

Figure 3. Timed automaton $G_i$ for a sequence of operations with start events $a_{i,1}, \dots, a_{i,n}$ and completion event $b_{i,n}$ for the last operation.

*Example 3.* In this example a straight sequence of $n$ operations is considered. The completion event $b_{i,j}$ of operation $j$ in a timed automaton $G_i$ is then also the starting event $a_{i,j+1}$ of the next operation $j+1$. With the time delay constraint $c_i \geq \Delta_{i,j}$ for operation $j$, a sequence of operations with start events $a_{i,1}, \dots, a_{i,n}$ and completion event $b_{i,n}$ for the last operation, is modelled by the timed automaton in Fig. 3.

For the synchronous composition $G_1 \parallel G_2$, the number of locations $|L|$ and the number of complete states $|X|$ including clocks are shown in Table 1. Note first that there are no shared events. Thus, no mutual exclusion or other types of restrictions are introduced. The result is an interleaving of the operations, in the same way as in Fig. 2. Table 1 shows that for a modest increase of the number of locations (49 locations for 5 operations), the complete number of states is nearly 10 times larger, still far less than the theoretical upper limit when no common clock states exist in the synchronous composition. This upper limit is 70 times higher than the number of locations. The increase of the state space is much higher when the number operations increases further, resulting in a significantly larger computational cost when time-optimal paths are requested. □

Table 1. State space evaluation for $n$ operations in both $G_1$ and $G_2$ according to Fig. 3. $|L|$ is the number of locations in $G_1 \parallel G_2$, and $\max |X|$ is the maximum number of states if no common clock states exist in an arbitrary $G_1 \parallel G_2$ with arbitrary clock constraints, while $|X|$ is the number of states when the common clock states are taken into account in $G_1 \parallel G_2$ according to Fig. 3.

| $n$ | $\lvert L\rvert$ | $\max \lvert X\rvert$ | $\lvert X\rvert$ |
|---|---|---|---|
| 1 | 9 | 19 | 14 |
| 2 | 16 | 69 | 43 |
| 3 | 25 | 251 | 109 |
| 4 | 36 | 923 | 236 |
| 5 | 49 | 3431 | 454 |

## 4.1 Clock state reduction based on time-optimality

When minimal waiting times are computed according to Prop. 1, some updated clock values can be neglected when a time-optimal solution is requested. This follows from the next proposition, where the minimal location time $t^*$ and the entry time clock vector $\boldsymbol{c}'$ are compared for two different paths. Since not only $t^*$ but also $\boldsymbol{c}'$ depends on the actual path, location is here replaced by path as argument also in $\boldsymbol{c}'$.

*Proposition 2.* Consider the entry time clock vector $\boldsymbol{c}'(\rho_k)$ and the minimal location time $t^*(\rho_k)$ for two specific paths $\rho_k^1, \rho_k^2 \in \Upsilon(\ell_k)$ where $\ell_k^1 = \ell_k^2$. If

$$\boldsymbol{c}'(\rho_k^1) \geq \boldsymbol{c}'(\rho_k^2) \wedge t^*(\rho_k^1) \leq t^*(\rho_k^2),$$

the state $(\ell_k, \boldsymbol{c}'(\rho_k^2))$ can be excluded from the list of current states to be considered in the evaluation of possible time-optimal paths.

*Proof:* Based on the inequality $\boldsymbol{c}'(\rho_k^1) \geq \boldsymbol{c}'(\rho_k^2)$, every individual clock $c_i$ also satisfies $c_i'(\rho_k^1) \geq c_i'(\rho_k^2)$. Due to Prop. 1, this implies that the minimal waiting time $w(\ell_k, \ell_{k+1})$ for path $\rho_k^1$ is always less or equal to the waiting time for path $\rho_k^2$, regardless of the actual value of the involved time delays $\Delta_i(\ell_k, \ell_{k+1})$. Indeed, this condition is valid for all future transitions until the final marked state is reached. Since $t^*(\rho_k^1) \leq t^*(\rho_k^2)$, and the fact that the recursive updates of all future minimal location times only depend on the future minimal waiting times, the final minimal location time $t^*(\rho^*)$ that generates the time-optimal path will always be lower, selecting the clocks based on path $\rho_k^1$ in location $\ell_k$, compared to path $\rho_k^2$ . □

The condition in this proposition is implemented as one single vector inequality

$$\left[\, \boldsymbol{c}'(\rho_k^1) \;\; - t^*(\rho_k^1) \right] \geq \left[\, \boldsymbol{c}'(\rho_k^2) \;\; - t^*(\rho_k^2) \right]$$

when the clock reduction proposed in this proposition is evaluated in the following example.

*Example 4.* To increase the applicability and complexity in the evaluation of the clock reduction in Prop. 2, $m$ sequences of operations defined by the timed automata $G_1$, $G_2$,…,$G_m$ in Fig. 3 are assumed to be executed concurrently as $G_1 \parallel G_2 \parallel \cdots \parallel G_m$, with an added mutual exclusion such that only one at a time of the subsystems can execute operation $j$ for $j = 1, 4, 7, \dots$, i.e. every third operation has this restriction. This is modelled by a shared resource $R_j$, shown in Fig. 4 for $m = 3$. The operation times are varied such that $\Delta_{i,j} = 10 + \mathrm{mod}(i+j, 3)$ for $j = 1, \dots, m$.

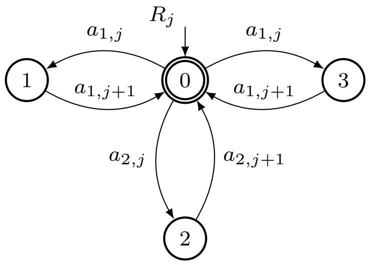


Figure 4. Automaton for a resource $R_j$, shared by $m = 3$ sequences of operations defined in Fig. 3.

The optimal make span $t^*$, the number of locations the total number of states $|X|$, and the computation time $ct$, including subscript $r$ when the reduction technique based on Prop. 2 is applied, are shown in Table 2. The results in this table show that the reduction technique is very impressive, mainly due to the fact that it is able to remove most of the additional states that are generated because of the incorporation of the clocks in the total state vector. The additional states are caused by the mutual exclusions. When the shared resources are removed, the number of states in the reduced model are always the same is the number of locations, i.e. $|X_r| = |L|$. □

## 5. TIME-OPTIMAL ABSTRACTION

As a complement to the reduction of clock states, we will now show how the number of locations can also be reduced by local optimization. This type of reduction is obtained by joining states with the same future behavior into equivalence classes, also called *block states*. Transitions between such states must be labeled by local non-shared events to be able to encapsulate them into block states.

Table 2. State space evaluation and computation times for $G_1 \,\|\, \cdots \,\|\, G_m$ with $n$ operations in every subsystem according to Fig. 3. The computation times with and without reduction are denoted $ct_r$ and $ct$, the optimal make span $t^*$, and the number of states including reduction $|X_r|$. t.o. means time out (>200 sec.), and for the rest of the notations we refer to Table 1

| $m$ | $n$ | $\lvert L\rvert$ | $\lvert X\rvert$ | $ct$ | $\lvert X_r\rvert$ | $ct_r$ | $t^*$ |
|---|---|---|---|---|---|---|---|
| 2 | 20 | 477 | 16000 | 2.36 | 759 | 0.037 | 230 |
| 2 | 60 | 3824 | 441727 | 199.3 | 6265 | 0.179 | 670 |
| 2 | 100 | 10370 | - | t.o. | 17103 | 0.486 | 1111 |
| 3 | 5 | 375 | 30935 | 46.32 | 1268 | 0.081 | 76 |
| 3 | 10 | 2064 | - | t.o. | 11353 | 0.969 | 132 |
| 3 | 20 | 13245 | - | t.o. | 148055 | 26.22 | 241 |

*Single entry-exit timed automata* To generate local explicit state models including clock states, a single entry and a single exit transition are assumed, but also a well defined clock state after the first entry transition. Such models are called *single entry-exit timed automata*, for which a time-optimal abstraction is easily obtained.

*Definition 4.* Let $G$ be a *single entry-exit timed automaton* with a clock vector $\boldsymbol{c}$, and single entry and exit transitions

$$\ell_0 \xrightarrow{a:\boldsymbol{c}'=\boldsymbol{c}_0} \ell_1 \qquad \text{and} \qquad \ell \xrightarrow{b:\varphi} \ell_m,$$

where the entry event $a$ and the exit event $b$ are shared with other timed automata, and $\varphi$ is a clock constraint. Between these transitions there may be a number of time delayed and event transitions, where all events are local and alternative paths are accepted. The next clock value constraint $\boldsymbol{c}' = \boldsymbol{c}_0$ in the entry transition defines the initial value of all clocks. □

The assumption that the initial value of all clocks is known is critical to be able to determine the transition weights and a time-optimal path for a single entry-exit timed automaton.

*Time-optimal abstraction* For a single entry-exit timed automaton, a time-optimal abstraction is presented in the following proposition.

*Proposition 3.* A single entry-exit timed automaton $G$ can be abstracted to a time-optimal three state timed automaton $\mathcal{A}(G)$ shown in Fig. 5, where the minimal makespan from location $\ell$ to location $\ell_m$ is $w^*(\ell, \ell_m)$.

*Proof:* Since the values of all clocks are defined at the entry transition and events after the first transition are local and therefore not restricted by any other subsystems, the minimal waiting time of each time delayed transition is easily defined by Prop. 1 as well as the minimal makespan from the entry location to the exit location, in the abstracted model denoted $w^*(\ell, \ell_m)$. Also note that all clocks that are only used in $G$ can be replaced by a single clock $c$ that only defines the time delay equal to the minimal makespan $w^*(\ell, \ell_m)$. □

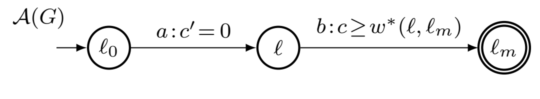


Figure 5. Time-optimal abstraction $\mathcal{A}(G)$ of a single entry-exit timed automaton $G$.

A single entry-exit timed automaton $G_1$ can be a part of a timed automaton $G$ that is temporarily separated from the rest of the system such that $G = G_1 \,\|\, G_2$. A time-optimal abstracted automaton $\mathcal{A}(G_1)$ then generates an abstraction of the original system $\mathcal{A}(G) = \mathcal{A}(G_1) \,\|\, G_2$. This procedure can be extended to other parts of the timed automaton $G$ where single entry-exit timed automata are naturally achieved.

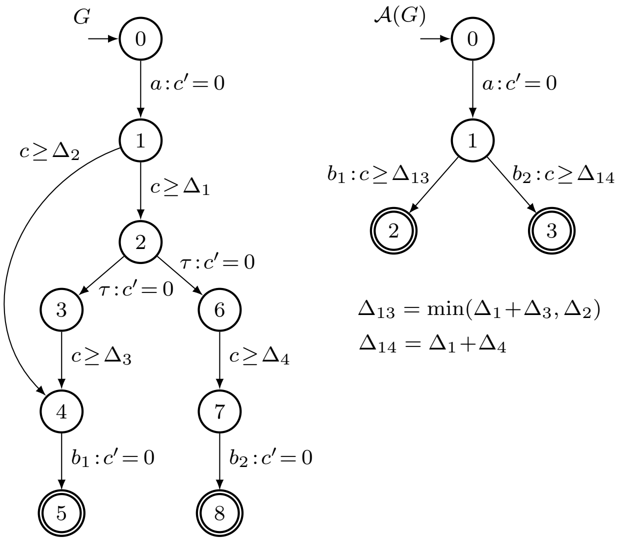


Figure 6. Timed automaton $G$ and its time-optimal abstraction $\mathcal{A}(G)$.

*Example 5.* Consider the timed automaton $G$ in Fig. 6. Since this model has two local events, here labeled by the hidden event $\tau$, an abstracted time-optimal model can be generated. From the entry event $a$ we find two exit events $b_1$ and $b_2$, which means that two single entry-exit timed automata can be formulated. The first one includes the states $\{0, 1, 2, 3, 4, 5\}$ and the exit event $b_1$, and the second one includes the states $\{0, 1, 2, 6, 7, 8\}$ and the exit event $b_2$.

Since the clock is reset before every time delayed transition, all non-zero minimal waiting times are equal to their lower limit time delays $\Delta_i$, $i = 1, \ldots, 4$. Thus, the shortest path from the entry event $a$ to the exit event $b_1$ is $\min(\Delta_1 + \Delta_3, \Delta_2)$, while the shortest path (the only path) from $a$ to $b_2$ is $\Delta_1 + \Delta_4$. These minimal path times are therefore according to Prop. 3 and Fig. 5 introduced as time delay clock constraints in the time-optimal abstraction $\mathcal{A}(G)$ in Fig. 6. □

In the next example it is illustrated how this abstraction technique can reduce the computation time significantly for larger systems.

*Example 6.* Consider the system in Example 4 where now the operation time for every second operation including mutual exclusion is $\Delta_{i,j} = 10 + \mathrm{mod}(i + j, 3)$ for $j = 1, \ldots, m$, while the other non-interacting local operations are assumed to be varied such that $\Delta_{i,j} = 20 + \mathrm{mod}(i + j, 3)$. These local operations can be abstractions of a sequences of local operations where for simplicity we assume that $\Delta_{i,j} = \sum_{k=1}^{r} \Delta_{i,j}/r$, i.e. every abstracted operation consists of $r$ local operations. In this way the number of original operations $n$ increases, and in Table 3 different values of $r$ and resulting $n$ are compared with the computation time for the the abstracted model ($r = 1$). This is a very simple type of abstraction, but from a computational point of view, including more complex behavior involving for instance alternative local paths generates the same type of computations. The computational burden is mainly in the optimization, which according to Table 3 is greatly simplified for $r = 1$. In this example, a final transition is also added when all operations in the individual sequences have beed completed. □

Table 3. State space evaluation and computation times for $G_1 \| \cdots \| G_m$ with $n$ operations in every subsystem according to Fig. 3. $r = 1$ corresponds to an abstraction, where $r > 1$ local operations without mutual exclusion are joined into one operation.

| $m$ | $n$ | $r$ | $\|L\|$ | $\|X_r\|$ | $ct_r$ | $t^*$ |
|---|---|---|---|---|---|---|
| 2 | 40 | 1 | 1745 | 2567 | 0.065 | 651 |
| 2 | 120 | 5 | 14865 | 49324 | 1.75 | 651 |
| 2 | 220 | 10 | 49265 | 370465 | 19.0 | 651 |
| 3 | 7 | 1 | 874 | 3935 | 0.25 | 129 |
| 3 | 10 | 2 | 2065 | 12551 | 1.16 | 129 |
| 3 | 13 | 3 | 3976 | 85210 | 37.5 | 129 |

This type of abstraction is easily performed on the original submodels, while requiring the value of all involved clocks at the entry transition makes this abstraction harder to apply after synchronization, when a number of clocks are involved. However, significant reductions can be achieved for special system structures also when a number of subsystems have been synchronized. This is illustrated in the last example.

*Example 7.* A synchronized system which satisfies the single entry-exit assumption in Prop. 3 is a group of concurrent operations where all operations are completed before a new set of independent and concurrent operations are started again. In Table 4 this is illustrated by first computing a model for $n = 4$ and $n = 5$ with $m = 3$ parallel sequences as in the earlier examples, and mutual exclusion added every third operation as in Example 4. Adding a first and final common transition before and after the $m$ concurrent sequences means that the time-optimal sequence can be repeated, without any additional computations. Alternatively, a global model can be generated where a copy of the first model is synchronized with itself, using local events except for the shared last event in the first submodel and the first event in the second copied submodel. The computation of a time-optimal path for this global model generates a model with $2nm$ operations, but due to the concurrency the computational complexity increases dramatically, compared to just repeating the optimal solution from the first subsystem a second time. □

Table 4. State space evaluation and computation times for $G_1 \| \cdots \| G_m$ with $n$ operations in every subsystem according to Fig. 3. In the second and fourth row, the first and third examples are repeated a second time after a synchronization, where the last shared event in the first subsystem is joined with the first event from a copy of this subsystem.

| $m$ | $n$ | $\|L\|$ | $\|X_r\|$ | $ct_r$ | $t^*$ |
|---|---|---|---|---|---|
| 3 | 4 | 234 | 756 | 0.047 | 66 |
| 3 | 8 | 466 | 12525 | 8.18 | 132 |
| 3 | 5 | 390 | 1426 | 0.090 | 76 |
| 3 | 10 | 778 | 42206 | 42.9 | 152 |

## 6. CONCLUSIONS AND FUTURE WORK

Time optimization of concurrent sequences of operations is in this paper solved by timed automata. To reduce the complexity an abstraction method has recently been proposed based on local optimization (Hagebring and Lennartson, 2019). In a modular subsystem, local paths without any communication with other subsystems are optimized, and when subsystems are synchronized more local behavior appears. The proposed method has shown to be successful, reducing computational complexity for important classes of planning problems. The original heuristic non-standard synchronous composition procedure to achieve true concurrency is in this paper replaced by a synchronous composition of timed automata. In the transformation to time weighted automata, a powerful state reduction is proposed and evaluated.

This modeling approach is simple but also general, making it possible to handle both automata and modular Petri nets with shared variables. An interesting future direction is to combine the proposed abstraction method with for instance Z3-Opt, since the modeling framework is suitable for the logic equation based input format that is available in such satisfiability solvers.

## ACKNOWLEDGEMENTS

Many thanks to my former PhD student Dr Fredrik Hagebring who initiated and formulated the basic concepts around compositional optimization.

## REFERENCES

Alur, R. and Dill, D.L. (1994). A theory of timed automata. *Theoretical Computer Science*, 126(2), 183 – 235. doi: https://doi.org/10.1016/0304-3975(94)90010-8.

Baier, C. and Katoen, J.P. (2008). *Principles of Model Checking*. The MIT Press, Cambridge, MA.

Hagebring, F. and Lennartson, B. (2018). Compositional optimization of discrete event systems. In *14th IEEE International Conference on Automation Science and Engineering*.

Hagebring, F. and Lennartson, B. (2019). Time-optimal control of large-scale systems of systems using compositional optimization. *Discrete Event Dynamic Systems*. doi: 10.1007/s10626-019-00290-0.

Hill, R. and Lafortune, S. (2016). Planning under abstraction within a supervisory control context. In *2016 IEEE 55th Conference on Decision and Control (CDC)*.

Hill, R. and Lafortune, S. (2017). Scaling the formal synthesis of supervisory control software for multiple robot systems. In *2017 American Control Conference (ACC)*.

Hoare, C. (1978). *Communicating Sequential Processes*, volume 21. ACM, New York, NY, USA. doi: 10.1145/359576.359585.

Lennartson, B., Basile, F., Miremadi, S., Fei, Z., Noori-Hosseini, M., Fabian, M., and Åkesson, K. (2014). Supervisory control for state-vector transition models - A unified approach. *IEEE Transaction on Automation Science and Engineering*, 11(1).

Lennartson, B. and Jia, Q.S. (2020). Reinforcement learning with temporal logic constraints. In *Proc. 15th International Workshop on Discrete Event Systems (WODES)*. IFAC-PapersOnLine.

Su, R., van Schuppen, J.H., and Rooda, J.E. (2012). The synthesis of time optimal supervisors by using heaps-of-pieces. *IEEE Transactions on Automatic Control*, 57(1), 105–118. doi:10.1109/TAC.2011.2157391.

Van Glabbeek, R.J. and Weijland, W.P. (1996). Branching time and abstraction in bisimulation semantics. *Journal of the ACM*, 43, 555–600.

Vilela, J. and Hill, R. (2022). Hierarchical planning in a supervisory control context with compositional abstraction. *Discrete Event Dynamic Systems*, 32, 89–113.

Ware, S. and Su, R. (2017). Time optimal synthesis based upon sequential abstraction and its application to cluster tools. *IEEE Transactions on Automation Science and Engineering*, 14(2), 772–784.