Keywords: static analysis, program slicing, system verification, timed systems, Timed Automata

Abstract. This paper presents a method of slicing timed systems to create reduced models for model checking verification. The reduction is made at the very beginning of the verification process and this makes it beneficial and effective in handling the state explosion problem. The method uses techniques of static analysis to examine the syntax of a program and to remove irrelevant fragments of the code. The timed extension of the classical slicing definition is given and applied to Intermediate Language of the verification system Verics and Estelle specification language.

1. Introduction

Modern systems, especially systems with time aspects, are very complex but we need them to be correct and reliable. In recent years many verification methods have been presented and one of the most important and promising seems to be the model checking. This technique has the advantage of being automatic but suffers from the state explosion problem. One of the solutions to overcome this problem is to construct an abstract model of the system preserving properties of interest. We propose to use a static analysis of the program code which allows to remove its unimportant fragments in the context of considered properties. Such technique is called program slicing and was first proposed by Weiser [16].

The main idea of slicing is that some parts of the program have no impact on verified properties and removing them can significantly reduce the size of the model. Advantages of such approach are that the full transition system for a program is never produced and there is no need to change existing model checking tools or algorithms, because the reduction takes place in the earlier phase of the verification process. Tip in [15] presents the most comprehensive survey of various flavors of slicing.

Program slicing as a reduction method has been applied in the context of model checking of untimed systems. Millett and Teitelbaum [14] study slicing of Promela, the input language of SPIN model checker.
A. Janowska, P. Janowski / Slicing Timed Systems

[12]. They obtain the so called **imprecise slice** and they do not formalize their slicing methods. Hatchiff et al. [11] present a formal study of slicing sequential programs preserving Linear Temporal Logic and extend their techniques also to multi-threaded Java programs [10]. Their results are used in the Bandera Tool Set [8]. Slicing is also present in the IF [3] environment concerning timed systems, however, it is defined for its untimed subset only [2].

In this paper we show that the idea of slicing can be also applied in the context of timed systems. We present new notions of dependency which arise when considering time and the method of computing sliced programs based on the extended dependency relations. Our slicing method is property driven, because the slicing criterion is derived from the atomic propositions of the formula to be checked.

We present this technique for Intermediate Language, the part of verification system Verics [6] and Estelle specifications [13]. Verics is a platform for verification of temporal properties of timed systems. It consists of several components, which are implemented as independent tools. Input formalisms of Verics are Estelle, Intermediate Language or Timed Automata. An Estelle specification is automatically translated to a description in Intermediate Language. The obtained specification is passed to the timed automata generator, which can produce one global automaton or a set of automata corresponding to the system components. These automata are passed to components implementing model checking techniques.

The paper is organized as follows. Section 2 introduces the syntax and semantics of the Intermediate Language. Section 3 presents the translation of Intermediate Language to Timed Automata. In Section 4 the slicing technique and its application to Intermediate Language and Estelle is described. Section 5 concludes the paper.

2. Intermediate Language

Intermediate Language allows for describing a set of processes which communicate via buffers (message passing mechanism) or shared variables. A process is described by terms of states and transitions. There are no explicit time variables nor clocks in the language, but the time of transition execution can be restricted.

Before presenting the language formally we introduce an example which illustrates our method throughout the paper.

2.1. Example

The program presented in Fig. 1 is composed of two processes: **producer** and **consumer** which share the variable **count** and the buffer **buf**. Each process has the local variable **data** which represents a portion of data. The **producer** has the additional local variable **choice**.

The **producer** produces portions of data which takes from 1 to 2 units of time. Then it passes to the state **transmit**. It waits until the buffer is not full (maximal number of elements in the buffer is 4), then it immediately places data in the buffer **buf**, increments the variable **count** and goes to the state **choose**. There it assigns 0 or 1 (randomly) to the variable **choice** and passes immediately to the state **produce**. And so on.

The **consumer** waits until the buffer is not empty and then decrements the variable **count**, takes data from the buffer **buf** and consumes it. The transition from the state **consume** to the state **request** can last an
arbitrary amount of time. We assume that each transition, except the latter one, is urgent, which means that if it becomes enable, then it must be executed.

2.2. Syntax

Let $V$ be a set\(^1\) of variables and $B$ be a set of buffers. A buffer is a possibly empty sequence of values.

$Expr_V$ is a set of arithmetic expressions over $V$. Let $y \in V$, $m \in \mathbb{Z}$ and $\oplus \in \{ -, +, /, \ast, \% \}$\(^2\). $Expr_V$ is defined by the following grammar:

$$expr := expr \oplus expr | - expr | (expr) | y | m$$

$Bexpr_{V,B}$ is a set of boolean expressions. Let $b \in B$, $y \in V$ and $\sim \in \{ =, \neq, <, >, \leq, \geq \}$. $Bexpr_{V,B}$ is defined inductively as follows:

$$bexpr := bexpr \land bexpr | bexpr \lor bexpr | \lnot bexpr | (bexpr) | expr \sim expr \lor get(b,y)$$

$Act_{V,B}$ is a set of actions. Let $b \in B$, $y \in V$, $e \in Expr_V$. An action is a sequence of operations defined as follows:

$$action := skip \lor get(b,y) \lor put(b,e) \lor y := e \lor action; action$$

\(^1\)Hereinafter all sets are finite.
\(^2\)\mathbb{Z} denotes a set of integer numbers, $\mathbb{N}$ denotes a set of non-negative integer numbers and $\mathbb{R}_+$ denotes the set of positive real numbers.
Definition 2.1. (Abstract syntax) A program in Intermediate Language is a tuple \( P = (V, B, \{P_i\}_{i \in \{1, \ldots, n\}} \), where \( V \) is a set of variables, \( B \) is a set of buffers and \( \{P_i\}_{i \in \{1, \ldots, n\}} \) is a set of processes. A process \( P_i \) is a tuple \((id_i, Q_i, q^i_0, \Sigma_i, T_i)\), where \( id_i \) is the process name, \( Q_i \) is a set of control states, \( q^i_0 \in Q_i \) is an initial control state, \( \Sigma_i \) is a set of transition labels and \( T_i \) is a set of transitions of the form \((q_i, g, d, u, l, a, q^i_1)\), where:

- \( q_i \in Q_i \) is called the source,
- \( q^i_1 \in Q_i \) is called the target,
- \( g \in Bexpr_{V, B} \) is called the guard,
- \( d \in \{ (d_1, d_2), [d_1, d_2], [d_1, d_2] \mid d_1, d_2 \in \mathbb{N} \} \) is called the allowed delay,
- \( u \in \{ \text{true}, \text{false} \} \) is called the urgency attribute,
- \( l \in \Sigma_i \) is called the label and
- \( a \in Act_{V, B} \) is called the action.

We write \( source(t) \), \( guard(t) \), \( delay(t) \), \( urgent(t) \), \( label(t) \), \( action(t) \) and \( target(t) \) for \( q_i \), \( g \), \( d \), \( u \), \( l \), \( a \) and \( q^i_1 \) of a transition \( t \), respectively.

A delay \( t \) represents constraints on time of the transition execution. The transition \( t \) cannot be executed before \( d_1 \) and after \( d_2 \) time units since it became enabled. An urgent \( t \) tells whether the transition has the priority over time progress. If the transition has the priority, time can progress only as long as the constraints on time of the transition execution are satisfied. Note that for an urgent transition the upper bound has to be finite. A label \( t \) is the sequence number of the transition in the program with the prefix “a”. Sets of labels \( \Sigma_i \) are disjoint.

\( P \) is a set of atomic propositions. Let \( b \in B \), \( e_1, e_2 \in Expr_V \), \( 1 \leq i \leq n \), \( q \in Q_i \) and \( v \) be a relational operator. An atomic proposition \( p \in P \) is of the following form:

\[
p := e_1 \sim e_2 \mid \text{empty}(b) \mid \text{id}_i \text{ at } q
\]

2.3. Semantics

This section shows how to associate a labeled transition system with a program. We begin with some preliminary definitions.

Let \( D \) be a domain containing values of variables and \( D^* \) be the set of sequences of values from \( D \). We use \( \Omega \) to denote \( D \cup D^* \). We define the valuation to be the total mapping \( v : V \cup B \to \Omega \) such that \( v(V) \subseteq D \) and \( v(B) \subseteq D^* \). The function \( v \) associates a value from \( D \) with each variable and a sequence of values with each buffer (the empty sequence is denoted by \( \varepsilon \)). We extend this mapping to expressions in the usual way. The expression \( get(b, y) \) evaluates to \( true \) if the buffer \( b \) is nonempty and to \( false \) otherwise. The valuation \( v^0 \) such that for all \( b \in B \), \( v^0(b) = \varepsilon \) (all buffers are empty) is called the initial valuation.

Satisfiability of a boolean expression \( g \in Bexpr_{V, B} \) in a valuation \( v \) (we write \( v \models g \)) is defined inductively as follows:
the value of this element to
∀DD/D8/D9/CP/D6/CS /satisfies
\{=, \neq, <, >, \leq, \geq\}.

• $v \models get(b, y)$ iff $b \neq \varepsilon$.

Let $v \xrightarrow{a} v'$ denote the execution of an action $a$, where $v$ and $v'$ are defined according to the action structure $(v[u/y])$ denotes the valuation $v$ with the value $u$ assigned to the variable or buffer $y$:

\[
\begin{align*}
\text{skip} & \rightarrow \quad v \\
\text{assign}(e) & \rightarrow \quad v[x] \\
n\rightarrow \quad v[u/x] \\
v \rightarrow \quad v[w, u/b] \\
v \rightarrow \quad v[w, u/b]
\end{align*}
\]

The effect of the operation $\text{get}(b, e)$ is the elimination of the first element of the buffer $b$ and assigning the value of this element to $y$. The operation $\text{put}(b, e)$ appends the value of the expression $e$ to the end of the buffer $b$.

Let $T = \bigcup_{i=1}^{n} T_i$ denote the set of all transitions. We define the delay valuation to be the total mapping $\tau: T \rightarrow \mathbb{R}_+$ which associates a positive real number with each transition. The delay $\tau(t)$ tells for how long the transition $t$ has been enabled. For $\delta \in \mathbb{R}_+, \tau + \delta$ denotes a delay valuation $\tau'$, such that for all $t \in T$, $\tau'(t) = \tau(t) + \delta$. $\tau^0$ is called the initial delay valuation if for all $t \in T$, $\tau^0(t) = 0$.

For $Y \subseteq T$, $\tau[Y := 0]$ denotes a delay valuation $\tau'$, such that for $t \in Y$, $\tau'(t) = 0$ and for $t \in T \setminus Y$, $\tau'(t) = \tau(t)$.

Let $1 \leq i \leq n, q_i \in Q_i, t_i \in T_i, v \in \Omega^{V \cup B}, d_1, d_2 \in \mathbb{N}$ and $\tau \in \mathbb{R}_+^T$. We define:

• $\text{enabled}(t_i, q_i, v) \triangleq (\text{source}(t_i) = q_i) \land v \models \text{guard}(t_i)$,

• $\text{fireable}(t_i, q_i, v, \tau) \triangleq \text{enabled}(t_i, q_i, v) \land \tau(t_i) \in \text{delay}(t_i)$,

• $\text{delay}^{\uparrow}(t_i) \triangleq \begin{cases} [0, d_2] & \text{for delay}(t_i) \in \{(d_1, d_2), (d_1, d_2)\} \\ [0, d_2] & \text{for delay}(t_i) \in \{(d_1, d_2), (d_1, d_2)\} \end{cases}$

The transition $t_i \in T_i$ is enabled, if the control state of $i$-th process is the source state of $t_i$ and $v$ satisfies $\text{guard}(t_i)$. The transition is fireable, if it is enabled and the value of the delay of the transition $t_i$ is in $\text{delay}(t_i)$. The $\text{delay}^{\uparrow}(t_i)$ denotes the $\text{delay}(t_i)$ with the lower bound set to 0.

**Definition 2.2. (Semantics)** The semantics of a program $P = (V, B, \{P_i\}_{i \in \{1, \ldots, n\}})$ with $P_i = (id_i, Q_i, \Psi_i, \Sigma_i, T_i)$, for an initial valuation $v^0: V \cup B \rightarrow \Omega$, is a labeled transition system $\mathcal{S} = (S, s_0, \rightarrow)$, such that:

- $S \subseteq Q_1 \times \ldots \times Q_n \times \Omega^{V \cup B} \times \mathbb{R}_+^T$ is the set of states,
- $s_0 = (q_i^0, \ldots, q_n^0, v^0, \tau^0) \in S$ is the initial state,
- $\rightarrow \subseteq S \times \bigcup_{i=1}^{n} \Sigma_i \times \mathbb{R}_+$ is the smallest transition relation defined by the following rules:
\[ \neg (q_1, \ldots, q_n, v, \tau) \xrightarrow{l} (q'_1, \ldots, q'_n, v', \tau') \text{ iff } \exists i \leq n \exists t \in T_i \text{ fireable}(t_i, q_i, v, \tau) \]
\[ \land l = \text{label}(t_i) \land q'_i = \text{target}(t_i) \land \forall 1 \leq j \neq i \leq n \quad q'_j = q_j \]
\[ \land v \overset{\text{action}(t_i)}{\longrightarrow} v' \land \tau' = \tau[Y := 0] \land Y = \{ t' \in T \mid \exists 1 \leq j \leq n \left( \neg \text{enabled}(t'_j, q_j, v) \lor \text{label}(t'_j) = l \right) \land \neg \text{urgent}(t_j) \} \}, \]
\[ (q_1, \ldots, q_n, v, \tau) \xrightarrow{\delta} (q_1, \ldots, q_n, v, \tau + \delta) \text{ iff } \delta > 0 \land \forall 1 \leq i \leq n \forall t_i \in T_i \quad \neg \text{enabled}(t_i, q_i, v) \lor \neg \text{urgent}(t_i) \land (\tau(t_i) + \delta) \in \text{delay}(t_i). \]

Initially, all buffers are empty and all variables have some initial values. Being in a state \( s = (q_1, \ldots, q_n, v, \tau) \) the system can:

- either execute a fireable transition of one of the processes and move to the state \( s' = (q'_1, \ldots, q'_n, v', \tau') \) instantaneously, without increasing delay values; delay value for every transition, which is newly enabled in the state \( s' \), is reset to zero,

- or let time \( \delta \) pass and move to the state \( (q_1, \ldots, q_n, v, \tau + \delta) \), unless there is an enabled transition with the priority which would become not fireable because of the time passage.

Transitions are **atomic** in the sense that the action of one transition cannot be interrupted by any action of another transition.

A **model** is a pair \( M = (S, \mathcal{V}) \), where \( \mathcal{V} \) is the **labeling function** \( \mathcal{V} : S \rightarrow 2^P \) for some fixed set of propositions \( P \), such that for \( s = (q_1, \ldots, q_n, v, \tau) \):

\[ e_1 \sim e_2 \in \mathcal{V}(s) \text{ iff } v \models e_1 \sim e_2 \]

\[ \text{empty}(b) \in \mathcal{V}(s) \text{ iff } v \models b = \varepsilon \]

\[ \text{id}_{i} \text{ at } q \in \mathcal{V}(s) \text{ iff } q_i = q \]

A state \( s = (q_1, \ldots, q_n, v, \tau) \) is labeled by the proposition \( p = e_1 \sim e_2 \), if the valuation \( v \) satisfies \( e_1 \sim e_2 \). Similarly, it is labeled by \( p = \text{empty}(b) \), if the buffer \( b \) is empty at \( v \). Finally, it has a label \( p = \text{id}_{i} \) at \( q \), if the \( i \)-th process is at the control state \( q \).

3. **Translation to Timed Automata**

Let \( X \) be a finite set of variables, called **clocks** and \( |X| = |T| \). We define a bijection \( c : T \rightarrow X \) which associates a clock with each transition\(^3\). \( \Psi_X \) is a set of **clock constraints**. Let \( x \in X \) and \( d \in \mathbb{N} \). \( \Psi_X \) is defined inductively as follows:

\[ \psi := x < d \mid x \leq d \mid x > d \mid x \geq d \mid \psi \land \psi \]

\(^3\)To reduce the number of clocks the same clock can be associated with many transitions if they cannot be enabled at the same time (for details see [7]).
We define the function $\text{constraint} : T \times \{ (d_1, d_2), (d_1, d_2), [d_1, d_2], [d_1, d_2] \} \rightarrow \Psi_X$, where $d_1, d_2 \in \mathbb{N}$, which for a transition $t \in T$ and its delay constructs the suitable clock constraint:

- $\text{constraint}(t, (d_1, d_2)) = (d_1 < c(t) \land c(t) < d_2)$,
- $\text{constraint}(t, (d_1, d_2)) = (d_1 < c(t) \land c(t) < d_2)$,
- $\text{constraint}(t, [d_1, d_2]) = (d_1 \leq c(t) \land c(t) \leq d_2)$,
- $\text{constraint}(t, [d_1, d_2]) = (d_1 \leq c(t) \land c(t) \leq d_2)$.

Let $q_i \in Q_i$, $1 \leq i \leq n$ and $v \in \Omega^{V \cup B}$. $U_i(q_1, \ldots, q_n, v)$ denotes the set of enabled transitions of the $i$-th process which are urgent i.e. the transitions which have the priority over the time progress: $U_i(q_1, \ldots, q_n, v) = \{ t \in T_i \mid enabled(t, q_i, v) \land urgent(t) \}$.

We adopt the definition of Timed Automata given in [1] and define the timed automaton for a program in Intermediate Language as follows.

**Definition 3.1.** A Timed Automaton for the program $\mathcal{P} = (V, B, \{ P_i \}_{i \in \{1, \ldots, n\}})$, where $P_i = (id_i, Q_i, q^0_i, S_i, T_i)$, is a tuple $\mathcal{A} = (\Sigma, S, s^0, X, E, \mathcal{I})$ such that:

- $\Sigma = \bigcup_{i=1}^n \Sigma_i$ is the set of labels,
- $S \subseteq Q_i \times \ldots \times Q_n \times \Omega^{V \cup B}$ is the set of locations,
- $s^0 = (q^0_1, \ldots, q^0_n, v^0) \in S$ is the initial location,
- $X$ is the set of clocks,
- $E \subseteq S \times \Sigma \times \Psi_X \times 2^X \times S$ is the transition relation defined as follows:
  $$\langle q_1, \ldots, q_n, v \rangle \xrightarrow{\text{transition}, \text{delay}(t_i)} \langle q'_1, \ldots, q'_n, v' \rangle \in E \quad \text{iff} \quad \exists_{1 \leq i \leq n} \exists_{t_i \in T_i} \text{ enabled}(t_i, q_i, v)$$
  $$\land l = \text{label}(t_i)$$
  $$\land \psi = \text{constraint}(t_i, \text{delay}(t_i))$$
  $$\land Y = \{ x \in X \mid \exists_{1 \leq j \leq n} \exists_{t_j \in T_j} c(t_j) = x \land (\neg \text{enabled}(t_j, q_j, v) \lor \text{label}(t_j) = l) \land \text{enabled}(t_j, q_j, v') \}$$
  $$\land q'_i = \text{target}(t_i) \land \forall_{1 \leq j \neq i \leq n} q'_j = q_j \land v \xrightarrow{\text{action}(t_i)} v'$$
- $\mathcal{I} : S \rightarrow \Psi_X$ is an invariant function such that
  $$\mathcal{I}(s) = \bigwedge_{1 \leq i \leq n} \bigwedge_{t_i \in U_i(q_1, \ldots, q_n, v)} \text{constraint}(t_i, \text{delay}(t_i))$$

Each element $e$ of $E$ denoted as $s \xrightarrow{l, \psi, Y} s'$ represents a transition from the location $s$ to the location $s'$, executing an action of the label $l$, with the set $Y \subseteq X$ of clocks to be reset and $\psi \in \Psi_X$ defining an enabling condition for $e$. $Y$ contains all clocks associated with transitions which are newly enabled in the location $s'$ and $\psi$ represents constraints imposed on time of the transition execution. Invariants ensure that the passage of time cannot make impossible executions of urgent transitions.

For the verification purpose we extend the standard definition of Timed Automaton by the labeling function $\mathcal{V}$ defined like in Section 2.3.

**Proposition 3.1.** The labeled transition system $S$ which is the semantics of the program $\mathcal{P}$ is isomorphic with the labeled transition system which is the state space of the timed automaton $\mathcal{A}$ constructed from $\mathcal{P}$.
We have presented the construction of one timed automaton for a whole system, but constructing a set of timed automata, one for each component of the system is also implemented. A component is a process, a global variable or a buffer. A location of automaton for a process is a valuation of local variables (visible only to that process) and its control state. A location of automaton for a global variable is a value of the variable and a location of automaton for a buffer is its contents. A composition of these automata results in global one presented above.

4. Slicing

The slicing of a program requires a few steps. First, dependency relations should be given stating how operations and control states depend on each other. Next, given properties of a program, sets of relevant operations and states must be computed. That is the operations and the states which have an impact on properties of interest. Finally, the rules of constructing the slice of the program which contains only relevant fragments of its code should be defined.

In this section we introduce dependency relations with additional notions of dependency which arise when considering time of the transition execution. We present the method of computing sliced program based on extended dependency relations. We also give some hints for handling more complex languages dealing with time, such as Estelle.

4.1. Dependency relations

As before, let \( y \in V, b \in B, e, e_1, e_2 \in Expr_v \). We extend a notion of an operation here. Let an operation be one of atomic operations of the form \( y := e, \text{get}(b, y) \) or \( \text{put}(b, e) \), that an action is composed of, or an atomic boolean expression of the form: \( e_1 \sim e_2 \) which can occur in a guard. We use also \( \text{opers}(\text{action}(t)) \subseteq A \) to denote the set of operations of the action of the transition \( t \), \( \text{opers}(\text{guard}(t)) \subseteq A \) for the set of operations of its guard and \( \text{opers}(t) \subseteq A \) for \( \text{opers}(\text{action}(t)) \cup \text{opers}(\text{guard}(t)) \). Let \( A = \bigcup_{t \in T} \text{opers}(t) \) denote the set of all operations of a program.

Let \( \text{vars}(e) \subseteq V \cup B \) be the set of variables and buffers which occur in the expression \( e \). Also \( \text{vars}(a) \) denotes the set of all operations of a program. For an operation \( a \in A, \text{def}(a) \subseteq V \cup B \) is the set of defined variables, that is, variables to which values are assigned.

\[
\text{def}(a) = \begin{cases} 
\{ y \} & \text{if } a = (y := e) \\
\{ b, y \} & \text{if } a = \text{get}(b, y) \\
\{ b \} & \text{if } a = \text{put}(b, e) \\
\emptyset & \text{if } a = (e_1 \sim e_2)
\end{cases}
\]

Also, \( \text{use}(a) \subseteq V \cup B \) is the set of used variables which are referenced during the operation \( a \), defined as follows:

\[
\text{use}(a) = \begin{cases} 
\text{vars}(e) & \text{if } a = (y := e) \\
\{ b \} & \text{if } a = \text{get}(b, y) \\
\{ b \} \cup \text{vars}(e) & \text{if } a = \text{put}(b, e) \\
\text{vars}(e_1) \cup \text{vars}(e_2) & \text{if } a = (e_1 \sim e_2)
\end{cases}
\]
Let $V_G \subseteq V$ denote the set of global variables, i.e. variables that are used or defined by more that one process. 

For $q \in Q_i$, where $1 \leq i \leq n$, let $\text{out}(q) \subseteq T_i$ be the set of outgoing transitions: $\text{out}(q) = \{ t \in T_i \mid q = \text{source}(t) \}$ and $\text{in}(q) \subseteq T_i$ be the set of incoming transitions: $\text{in}(q) = \{ t \in T_i \mid q = \text{target}(t) \}$. 

A control path in $P_i$ from a state $q \in Q_i$ to a state $q' \in Q_i$ is a sequence of control states and transitions of the form $q_1 t_2 q_2 \ldots t_m q_m$ such that $q_1 = q$, $q_m = q'$, $m \geq 2$ and for $2 \leq j \leq m$ : $t_j \in T_i$, $q_{j-1} = \text{source}(t_j)$ and $q_j = \text{target}(t_j)$. We call a control path maximal if it ends in a state with no outgoing transitions or it contains a cycle, that is, it contains one of its states twice. A path is simple if it does not contain a cycle. As usual, we call the state $q' \in Q_i$ reachable from the state $q \in Q_i$ if there exists a control path from $q$ to $q'$.

We assume that each control state of a process is reachable from the initial state of that process. We denote by $\pi_{q_0}$ the set of all simple control paths from the state $q$ to the state $q'$. We use also $\text{states}(\pi_{q_0})$ to denote the set of states occurring in $\pi_{q_0}$.

Data dependence and control dependence are basic notions of dependence in the slicing literature ([15]). We extend these notions with time dependence and define each of them for Intermediate Language.

**Definition 4.1. (Data dependence)** For $t_1 \in T_i$, $t_2 \in T_j$, where $1 \leq i, j \leq n$, an operation $a_1 \in \text{opers}(t_1)$ is data dependent on $a_2 \in \text{opers}(t_2)$ (we write $a_1 \xrightarrow{\text{dd}} a_2$), if there is a variable $v \in V$ (or $b \in B$) such that $v \notin \text{def}(a_2) \cap \text{use}(a_1)$ (resp.) and

1. $t_1 = t_2$ and $a_2$ is followed\(^4\) by $a_1$ in $\text{action}(t_1)$ and for every $a_3 \in \text{opers}(\text{action}(t_1))$ between $a_2$ and $a_1$ in $\text{action}(t_1)$, $v \notin \text{def}(a_3)$ (or $b \notin \text{def}(a_3)$, resp.) or

2. the state $\text{source}(t_1)$ is reachable from the state $\text{target}(t_2)$ and for each $t_3$ on each simple path from $\text{target}(t_2)$ to $\text{source}(t_1)$ and for each $a_3 \in \text{opers}(t_3)$, $v \notin \text{def}(a_3)$ (or $b \notin \text{def}(a_3)$, resp.) or

3. $i \neq j$.

Intuitively, the operation $a_1$ depends on the operation $a_2$, if $a_2$ can influence the value of a variable (or a buffer) used at $a_1$. Data dependencies induced by the global variable $\text{count}$ are presented in Fig. 2.

**Definition 4.2. (Control dependence)** For two control states $q_1, q_2 \in Q_i$, where $1 \leq i \leq n$, we say that $q_1$ is control dependent on $q_2$ (we write $q_1 \xrightarrow{\text{cd}} q_2$), if $q_1$ is reachable from $q_2$ and one of the following holds:

1. $\forall t \in \text{out}(q_2) \; \text{guard}(t) \neq \text{true}$ or

2. there is a maximal path from $q_2$ which does not pass through $q_1$ and each maximal path from any other state of each path of $\pi_{q_2q_1}$ passes through $q_1$.

\(^4\)Actions are sequences of operations, so terms: “follow” and “between” are well defined.
The first item says that the state $q_1$ is control dependent on the state $q_2$ if there is a path from $q_2$ to $q_1$ and the process can be blocked at the state $q_2$ because there might be no enabled transitions. The second item describes the standard control dependence: it identifies states on which reachability of $q_1$ depends. Being at the state $q_2$ the decision is made whether or not the execution path passes through the state $q_1$. For example, states in a cycle are control dependent on the state which allows to leave the cycle. Examples of control dependencies are presented in Fig. 3.

**Definition 4.3. (Time dependence)** For two control states $q_i, q_j \in Q_i$ where $1 \leq i \leq n$, we say that $q_i$ is time dependent on $q_j$ (we write $q_i \xrightarrow{t} q_j$), if $q_i$ is reachable from $q_j$ and one of the following holds:

1. $\forall t \in \text{out}(q_i) \quad \text{urgent}(t) = \text{false}$
2. $\exists t_1, t_2 \in \text{out}(q_i) \quad \text{delay}(t_1) \neq \text{delay}(t_2) \lor \text{urgent}(t_1) \neq \text{urgent}(t_2)$
3. $\exists t \in \text{out}(q_i) \exists y \in V \cup B \quad y \in \text{vars(\text{guard}(t))}$

We call an operation $a_1 \in \text{op}(\text{action}(t_1))$ time dependent on $a_2 \in \text{op}(\text{guard}(t_2))$ ($a_1 \xrightarrow{t_1} a_2$), where $t_1 \in \text{out}(q_1)$, $t_2 \in \text{out}(q_2)$, $q_1, q_2 \in Q_i$, if $q_1 \xrightarrow{t} q_2$ or $t_1 = t_2$.

In the first case the state $q_1$ is time dependent on the state $q_2$, because the progress of time might disable all transitions going from the state $q_2$ — the process might be blocked and thus $q_1$ would not be reached. Justification for the second case is that in timed systems it is important not only whether a
state is reached but also when it is reached. The state $q_1$ can be reached at different moments of time and we do not want to exclude any possible execution. The last case concerns the moment of time when a transition becomes enabled. Although a global variable or a buffer might have no impact on variables of interest (no data dependence), it can still influence behaviour of other processes.

Time dependencies between control states are presented in Fig. 4.

4.2. Constructing the slice

Let $P_\varphi \subseteq P$ be the set of atomic propositions from the formula $\varphi$. Let $\text{states}(P_\varphi)$ denote the set of observable control states, i.e., states which appear in propositions of $P_\varphi$ or have no outgoing transitions. Reachability of any state depends on states with no outgoing transitions and that is why they are also considered as observable. Let also $\text{vars}(P_\varphi)$ denote the set of observable variables and buffers, i.e., variables and buffers which appear in propositions of $P_\varphi$. Slicing of a program depends on properties to be verified. The slicing criterion describes the program’s points of interest, so it is defined by the set of propositions $P_\varphi$. Recall, that $A$ denotes the set of all program operations.

**Definition 4.4. (Slicing criterion)** The slicing criterion $C_\varphi \subseteq A$ for the program $P = (V, B, \{P_i\}_{i \in \{1, \ldots, n\}})$, where $P_i = (id_i, Q_i, q_0^i, \Sigma_i, T_i)$, with respect to the set $P_\varphi$ of atomic propositions is defined as follows:

$$C_\varphi = \{a \in A \mid \text{def}(a) \cap \text{vars}(P_\varphi) \neq \emptyset\} \cup \{a \in A \mid \exists q_{\text{states}}(P_\varphi) \exists t_{\text{in}}(q) \ a \in \text{opers}(\text{guard}(t))\}$$
All operations defining observable variables and buffers are included in the slicing criterion. Operations which appear in guards of transitions going to observable control states directly influence reachability of these states and are also included in the slicing criterion.

For the example from Section 2.1 we check the formula

$$\varphi = AG(\text{consumer at request} \rightarrow AF(\text{consumer at consume}))$$

which means that always when the consumer requests some data to consume, then it eventually gets it. The set of atomic propositions $P_\varphi = \{\text{consumer in request}, \text{consumer in consume}\}$ and $C_\varphi = \{\text{count > 0}\}$, because the transition to the state consume is guarded by this operation.

We define the dependence relation $\frac{d}{dt}$ on operations of a program to be the union of previously defined relations: $\frac{dd}{dt}$, $\frac{cd}{dt}$, and $\frac{td}{dt}$. We use also $\frac{ctd}{dt}$ to denote union of relations $\frac{cd}{dt}$ and $\frac{td}{dt}$ on control states.

The set of relevant operations is defined in terms of the transitive closure of the dependence relation. Only relevant operations will be included in the program slice. This part can be called data slicing, because it reduces the amount of data needed to reason about the program.

**Definition 4.5. (Relevant operations)** The set $A^\varphi \subseteq A$ of relevant operations with respect to the set $P_\varphi$ of atomic propositions is defined as:

$$A^\varphi = C_\varphi \cup \{a \in A \mid \exists a_0 \in C_\varphi \ a_0 (\frac{d}{dt})^* a\}$$
In our example, the operation $\text{count} > 0$ is data dependent on operations $\text{count} = \text{count} - 1$ and $\text{count} = \text{count} + 1$. The operation $\text{count} = \text{count} + 1$ is control dependent on the operation $\text{count} < 4$. Thus, $A^\circ = \{\text{count} < 4, \text{count} = \text{count} + 1, \text{count} > 0, \text{count} = \text{count} - 1\}$. Relevant operations are underlined in Fig. 5.

Given the set of relevant operations, we can simply erase from the program all operations not included in this set. However, we also would like to reduce the control structure of each process. It is particularly feasible when we check properties related only to a proper subset of processes. Then, control structures of processes not executing any relevant operations can be reduced to a single control state with a single transition. The relevant control states are the states to be included in the slice.

**Definition 4.6. (Relevant control states)** The set $Q_i^R \subseteq Q_i$ of relevant control states of the $i$-th process with respect to the set $P_\varphi$ of atomic propositions is defined in four steps:

1. $Q_i^0 = \{q^0\} \cup \text{states}(P_\varphi) \cup \{q \in Q_i \mid \exists t \in \text{out}(q) \text{ target}(t) \in \text{states}(P_\varphi)\}$
   $\cup \{q \in Q_i \mid \exists t \in \text{in}(q) \text{ oper}(t) \cap A^\circ = \emptyset\}$

2. $Q_i^1 = Q_i^0 \cup \{q \in Q_i \setminus Q_i^0 \mid \exists q' \in Q_i^0 \ q' \xrightarrow{\text{ctrl} t} q\}$

3. Let $Q_i^2 = \{q \in Q_i \mid \exists q' \in Q_i^1 \pi_{q'q} \neq \emptyset\}, Q_i^3 = Q_i^1 \cup \{q \in Q_i^2 \mid \exists t \in \text{out}(q) \text{ delay}(t) \neq [0,0] \}$
   $\cup \{q \in Q_i \setminus Q_i^2 \mid \exists t \in \text{in}(q) \text{ source}(t) \in Q_i^2\}$

4. $Q_i^4 = Q_i^2 \cup \{q \in Q_i^2 \setminus Q_i^3 \mid \exists t \in \text{in}(q) \text{ source}(t) \in Q_i^2 \}$
   $\quad \land \pi_{q',q} \neq \emptyset \land \forall q' \in \text{states}(\pi_{q',q} \text{ source}(t))$
   $\quad q' \in Q_i \setminus Q_i^3 \setminus \exists \psi \in \text{out}(\text{source}(t)) \text{ target}(t') \in Q_i^3\}$

We call a path $q_1^1 \xrightarrow{t_1} q_2^2 \xrightarrow{t_2} \ldots q_m^m$ from the state $q_1^1 \in Q_i$ to the state $q_m^m \in Q_i$ invisible, if $m > 2$ and for $j = 1, \ldots, m$: $q_j^j \in Q_i \setminus Q_i^\circ$ and $t_j^j \in T_j$. By $q_1^1 \xrightarrow{\text{intr}} q_m^m$ we mean that there exists an invisible path from the state $q_1^1$ to the state $q_m^m$. $Q_i^R \subseteq Q_i$ is the set of control states of the $i$-th process such that for each $q \in Q_i^R$ there exists $q' \in Q_i^R$ reachable from $q$.

In the first step, we construct the set of relevant states with the initial state and the observable states. The states which are immediate predecessors of observable states are included in this set to ensure that an observable state would not be identified with non-observable one. All states with at least one outgoing transition executing or guarded by a relevant operation are also considered as relevant.

Next, the transitive closure of the union of control and time dependence relations of the set of states obtained in the first step is computed. Note, that all transitions going out of states added in the second step have empty guards. Otherwise these states would be added in the first step.

We would like the timing behaviour of the sliced program be exactly the same as the one of the original program. To this end in the third step we add to the set of relevant states all states from $Q_i^R$ which have non-immediate outgoing transitions. We also need to represent states from $Q_i \setminus Q_i^R$. This is why states from $Q_i \setminus Q_i^R$, which have immediate predecessors from $Q_i^R$, are added to the set of relevant states.

The last step considers the situation when there is an invisible path from a state included in $Q_i^R$ to the same state. We cannot replace a cycle of at least two states with a self loop because their behaviours are different, namely times of resetting delays of other transitions going out of the state are different. Thus, the first state of the path is added to the set of relevant states to break such invisible path.
Considering our example, the set of producer’s relevant states contains the initial state `produce` and the state `transmit` because there are relevant operations in the transition going out from it. The state `choose` is not relevant, because none of those two states depends on it and the other conditions of Definition 4.6 are not satisfied. Since both consumer’s states are observable, they are both relevant. Relevant states are underlined on Fig. 5.

![Diagram](image)

**Figure 5.** Relevant states and operations

A slice of a program is computed as follows.

**Definition 4.7. (Program slice)** Given a program \( P = (V, B, \{P_i\}_{i \in \{1, \ldots, n\}} \), where \( P_i = (id_i, Q_i, q^0_i, \Sigma_i, T_i) \) and a valuation \( \nu^0 : V \cup B \rightarrow \Omega \), we construct a sliced program \( P' = (V', B', \{P'_i\}_{i \in \{1, \ldots, n\}} \), where \( P'_i = (id_i, Q'_i, q'^0_i, \Sigma_i, T'_i) \) and a valuation \( \nu'^0 : V' \cup B' \rightarrow \Omega \), with respect to the set \( P_\nu \) of atomic propositions, as follows:

1. \( V' = \{ v \in V \mid \exists a \in A \forall \nu v \in \text{vars}(a) \} \)
2. \( B' = \{ b \in B \mid \exists a \in A \forall \nu b \in \text{vars}(a) \} \)
3. for each process \( P'_i \):
   - (a) \( Q'_i = Q^0_i \)
   - (b) \( q'^0_i = q^0_i \)
(c) \( T'_i = T'_i^{1} \cup T'_i^{2} \), where 
\[ T'_i^{1} = \bigcup_{q \in Q'_i} \{ t \in T_i \mid \text{source}(t) \in Q'_i \land (\text{source}(t) \xrightarrow{\text{inv}} q \lor \text{target}(t) = q) \}, \]
\[ T'_i^{2} = \{ t \in T_i \mid \text{source}(t) \in Q'_i \setminus Q'_i \} \] and for each \( t' \in T'_i \) and the corresponding \( t \in T_i \) and \( q \in Q'_i \) the following holds:

i. \( \text{source}(t') = \text{source}(t) \),

ii. if \( t' \in T'_i^{1} \), then \( \text{target}(t') = q \), otherwise \( \text{target}(t') = \text{source}(t') \),

iii. \( \text{guard}(t') \) and \( \text{action}(t') \) are composed of relevant operations of \( \text{guard}(t) \) and \( \text{action}(t) \), respectively,

iv. if \( t' \in T'_i^{2} \), then \( \text{delay}(t') = [0, \infty) \) and \( \text{urgent}(t') = \text{false} \), otherwise \( \text{delay}(t') = \text{delay}(t) \) and \( \text{urgent}(t') = \text{urgent}(t) \),

\[ v^O(V' \cup B') = v^O(V' \cup B') \]

Only relevant variables and buffers, that is, these which appear in relevant operations, are included in the slice of the program. The sets of control states contain only relevant control states. For each process the set of transitions is composed of transitions of the original program going from relevant states. If a transition goes to a non relevant state, then its target is changed. If there are invisible paths from the source of the transition to relevant states, then in the sliced system there are transitions to all such states because all possible executions should be represented. Otherwise the transition is a self loop. The guard and action of each transition are composed exclusively of relevant operations. If the source of a transition is non relevant, which means that there are no relevant states reachable from it, then its allowed delay is changed to arbitrary and the transition is made not urgent. Otherwise the allowed delay and urgency attribute are not changed.

Finally, we can reduce the program by removing a transitions redundancy. If two or more transitions differ only in their labels, then removing all but one of them does not change the behaviour of the system, because labels have only informational meaning. In this case the transition with the lowest label is left and the others are removed.

In our example, the variable \( \text{count} \) is the only variable in the slice of the program. There are no references neither to the buffer \( \text{buf} \) nor to local variables. The sets of control states are composed of relevant states. In the original program there is the transition from the state \( \text{transmit} \) to the state \( \text{choose} \), but there is no such state in its sliced version. Thus the target of this transition is changed to the nearest relevant state, that is the state \( \text{produce} \). The sliced program is presented in Fig. 6.

4.3. Results

The timed automaton for the original program from Section 2.1 has 620 locations and 1226 transitions and timed automaton for its slice, with respect to the set of atomic propositions defined in Section 4.2, has only 20 location and 36 transitions. It should be clear that this example can be generalized to larger ones. For example if producer produces consecutive natural numbers instead of 0 or 1 we could not obtain timed automaton for such program because the number of locations would be infinite. However, from the sliced program (with respect to the same set of propositions) we would obtain the same automaton with 20 locations and 36 transitions.
4.4. Correctness of slicing

We conclude with notions of correctness of our slicing method. Let $\mathcal{S}$ be the labeled transition system associated with program $\mathcal{P}$ for the initial valuation $v^0$ and $\mathcal{S}'$ be the labeled transition system of $\mathcal{P}'$, the slice of $\mathcal{P}$ with respect to the set of atomic propositions $P_\varphi$ for the initial valuation $v'^0$. Labeling functions $\mathcal{V}: \mathcal{S} \rightarrow 2^{F_v}$ and $\mathcal{V}': \mathcal{S}' \rightarrow 2^{F_v}$ are defined as before (Section 2.3).

**Definition 4.8.** Let $s = (q_1, \ldots, q_n, v, \tau) \in \mathcal{S}$, $s' = (q_1', \ldots, q_n', v', \tau') \in \mathcal{S}'$ and $\cong \subseteq \mathcal{S} \times \mathcal{S}'$. $s \cong s'$ iff:

1. for each pair $q_i \in Q_i$ and $q_i' \in Q_i'$, where $i = 1, \ldots, n$
   
   (a) if $q_i \in Q_i^R \cap Q_i^R$, then $q_i = q_i'$, otherwise

   (b) if $q_i \in Q_i^R \setminus Q_i^R$, then $q_i \xrightarrow{\text{in} v} q_i'$, otherwise

   (c) if $q_i \in Q_i \setminus Q_i^R$, then $q_i \xrightarrow{\text{in} v} q_i$ or $q_i' = q_i$ and

2. $v(V' \cup B') = v'(V' \cup B')$ and

3. for all $q \in Q_i \cap Q_i'$, where $i = 1, \ldots, n$, for all $t \in \text{out}(q)$, there exists $t' \in T_i'$ such that $t' \in \text{out}(q)$ and if $\text{en\text{abled}}(t, q, v)$ then $\tau'(t') = \tau(t)$.

**Lemma 4.1.** The relation $\cong \subseteq \mathcal{S} \times \mathcal{S}'$ is a stuttering bisimulation between two structures $M = (\mathcal{S}, \mathcal{V})$ and $M' = (\mathcal{S}', \mathcal{V}')$. The proof of Lemma 4.1 can be found in the appendix. From [4] we know that two structures $M = (\mathcal{S}, \mathcal{V})$ and $M' = (\mathcal{S}', \mathcal{V}')$ are equivalent w.r.t. $\text{CTL}_\varphi$ ([5]) formulas if there exists a stuttering
bisimulation between the states of two structures. Thus, the consequence of Lemma 4.1 is that a program satisfies a $\text{CTL}_{\text{X}}^*$ formula $\varphi$, if and only if the slice of the program with respect to the set of propositions $P_\varphi$ satisfy $\varphi$.

4.5. Slicing Estelle

Application of the presented method directly to Intermediate Language is suitable, because Intermediate Language is meant to be the target of translations from various high-level specification languages and this makes one universal solution. On the other hand, the translation is a part of verification process of Verics as well, so we can apply the method to the high-level specification in the process of translation.

Let’s consider the translation from Estelle specification language [13] to Intermediate Language. The execution semantics of the subset of Estelle we use and Intermediate Language are similar and the rules described for Intermediate Language are applicable to Estelle as well. However, Estelle allows for conditional statements (if/then/else) and loops (while and for) in transitions body, so additional rules for such statements should be supplied. Of course, those constructions are translated into Intermediate Language, but it is definitely more efficient to reduce them at the syntax level in Estelle than to consider the structures representing their execution in Intermediate Language. Here, we present some hints according to which slicing for Estelle-like language can be performed.

Let’s consider the language from Section 2.2 with the set of actions extended by operations: $\texttt{if} (\texttt{bexpr}) \texttt{\{action1; \} else \{action2; \} and while (\texttt{bexpr}) \{action; \}}$. Semantics of these operation is as usual.

Dependency relations are extended by dependences which arise when considering $\texttt{if}$ and $\texttt{while}$ statements. Thus, operations from $\texttt{action1}$ and $\texttt{action2}$ are dependent on operations from $\texttt{bexpr}$. Similarly, operations from $\texttt{action}$ are dependent on the condition of the $\texttt{while}$ statement. But also all operations reachable from the state to which goes a transition with $\texttt{while}$ statement are dependent on the loop condition, because they depend on whether or not the loop ends. On the contrary the rule for $\texttt{for}$ loop would be similar to the rule for $\texttt{if}$, because it always ends. If a condition of $\texttt{if}$ or $\texttt{while}$ statement is relevant, then the statement is included in a sliced program.

5. Conclusions

Program slicing has already been applied in the context of untimed systems. The paper shows how the slicing technique can also be adapted in model checking of timed systems. We have extended standard dependency relations by dependencies that arise when concerning time features of the language and presented the method of constructing the sliced program with respect to the set of propositions of the formula to be verified. The presented method of reduction preserves $\text{CTL}_{\text{X}}^*$.

References


We recall the notation first. \( P = (V, B, \{ P_i \}_{i \in \{1, \ldots, n\}} \), where \( P_i = (id_i, Q_i, q_i^0, \Sigma_i, T_i) \), is a program. \( P' = (V', B', \{ P'_i \}_{i \in \{1, \ldots, n\}} \), where \( P'_i = (id_i, Q'_i, q'_i, \Sigma'_i, T'_i) \), is the slice of the program constructed from \( P \) according to Definition 4.7. \( S = (S, s_0, \rightarrow) \) and \( S' = (S', s'_0, \rightarrow') \) are transition systems for programs \( P \) and \( P' \), respectively (see Definition 2.2). A state \( s \in S \) is of the form \( (q_1, \ldots, q_n, v, \tau) \), where \( v : V \cup B \to \Omega \) and \( \tau : T \to \mathbb{R}_+ \), and \( s' \in S' \) is of the form \( (q'_1, \ldots, q'_n, v', \tau') \), where \( v' : V' \cup B' \to \Omega \) and \( \tau' : T' \to \mathbb{R}_+ \). 5 \( \mathcal{V} : S \to 2^F \) and \( \mathcal{V}' : S' \to 2^F \) are labeling functions. \( M = (S, \mathcal{V}) \) and \( M' = (S', \mathcal{V}') \) are two structure which we claim are stuttering bisimilar.

5We will use this notation also for indexed states, for example \( s_i = (q_i^1, \ldots, q_i^n, v_i, \tau_i) \).
$Q_i$ is the set of control states of the $i$-th process of the program $P$. $Q_i^l \subseteq Q_i$ is the set of control states of the $i$-th process of the program $P^l$. $Q_i^R \subseteq Q_i$ is the set of control states of the $i$-th process such that for each $q \in Q_i^R$ there exists $q' \in Q_i^l$ reachable from $q$ (for details see Definition 4.6).

**Definition A.1. (Stuttering bisimulation)** A relation $\cong_b \subseteq S \times S'$ is a stuttering simulation between two structures $M = (S, \mathcal{V})$ and $M' = (S', \mathcal{V'})$, where $S = (S, s_0, \rightarrow)$ and $S' = (S', s'_0, \rightarrow')$, if the following conditions hold:

1. $s_0 \cong_b s'_0$ and

2. if $s \cong_b s'$, then $\mathcal{V}(s) = \mathcal{V'}(s')$ and for every maximal path $\sigma$ of $M$ that starts at $s$, there is a maximal path $\sigma'$ of $M'$ that starts at $s'$, a partition $B_1, B_2, \ldots$ of $\sigma$, and a partition $B'_1, B'_2, \ldots$ of $\sigma'$ such that for each $j \geq 1$, $B_j$ and $B'_j$ are nonempty and finite, and every state in $B_j$ is related by $\cong_b$ to every state in $B'_j$.

A relation $\cong_b$ is a stuttering bisimulation if both $\cong_b$ and $\cong_b^T$ (the transpose of $\cong_b$) are stuttering simulations.

**Proof:**

We begin by proving that $s_0 \cong s_0'$, notice that according to item 3(b) of Definition 4.7, $q_i^{0'} = q_i^0$ for $i = 1, \ldots, n$, which satisfies item 1(a) of Definition 4.8. Next, by item 4 of Definition 4.7 $v^{0'}(V' \cup B') = \tau^0(V' \cup B')$, hence item 2 of Definition 4.8 is satisfied. Finally, according to item 3(c) of Definition 4.7 for all $q \in Q_i^l \cap Q_i^R$ and for all $t \in \text{out}(q)$ there exists $t' \in T_i$ such that $t' \in \text{out}(q)$ and $\tau^0(t') = 0 = \tau^0(t)$, so item 3 of Definition 4.8 is also satisfied.

Let $s \in S$, $s' \in S'$ and $s \cong s'$. First, we show that $\mathcal{V}(s) = \mathcal{V'}(s')$ (for the definition of the function $\mathcal{V}$ see Section 2.3). By Definition 4.4 and item 1 of Definition 4.7 we have $\text{var}_s(P_\sigma) \subseteq V \cup B'$, Next, by item 2 of Definition 4.8 $v'(V' \cup B') = \nu(V' \cup B')$. From this it follows that $(e_1, e_2) \in V(s')$ iff $(e_1, e_2) \in V(s)$, or $(d_1, d_2) \in B_{expr}(V')$, and $\text{empty}(b) \in V(s)$ iff $\text{empty}(b) \in V(s)$, or $(b, b') \in B'$. It remains to show that $(id_i, a) \in V(s)$ iff $(id_i, a) \in V(s')$ for $q \in \text{states}(P_\sigma)$. According to item 1 of Definition 4.6 $\text{states}(P_\sigma) \subseteq Q_i^l \cap Q_i^R$. For $i = 1, \ldots, n$:

1. if $q_i \in Q_i^l \cap Q_i^R$, then by item 1(a) of Definition 4.8 $q_i = q_i$.

2. if $q_i \in Q_i^l \cap Q_i^R$ and $q_i \notin \text{states}(P_\sigma)$, then by item 1(b) of Definition 4.8 $q_i \overset{\text{inv}}{\rightarrow} q_i$. Since $q_i \notin \text{states}(P_\sigma)$, we have $q \neq q_i$.

According to item 1 of Definition 4.6 for any state $q' \in Q_i$, if there exists $t \in \text{out}(q')$ such that $t \in \text{in}(q)$, then $\tau^0(t') = 0 = \tau^0(t)$, which means that there is no invisible path going from any state $q' \in Q_i$ to the state $q$, so we have $q \neq q_i$.

3. if $q_i \in Q_i^l \cap Q_i^R$, then by item 1(c) of Definition 4.8 $q_i \overset{\text{inv}}{\rightarrow} q_i$. Since $q_i \notin \text{states}(P_\sigma)$, we have $q \neq q_i$. According to item 3 of Definition 4.6 there is no invisible path from any state $q' \in Q_i^R$ to the state $q_i \notin \text{states}(P_\sigma)$, hence $q_i \in Q_i^l \cap Q_i^R$ and we have $q \neq q_i$.

From this we see that $q = q_i$ iff $q = q_i$, which completes the proof of $\mathcal{V}(s) = \mathcal{V}(s')$. Let $\sigma$ be a maximal path of $M$ starting at $s$. We proceed to show how to construct a maximal path $\sigma'$ starting at $s'$, a partition $B_1, B_2, \ldots$ of $\sigma$ and a partition $B'_1, B'_2, \ldots$ of $\sigma'$ as required in item 2 of Definition A.1. The construction is inductive. We assume that we have already constructed a finite
prefix of $\sigma$: $s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots \xrightarrow{a_{k-1}} s_k$ and a finite prefix of $\sigma'$: $s'_1 \xrightarrow{a'_1} s'_2 \xrightarrow{a'_2} \cdots \xrightarrow{a'_{i-1}} s'_i$, where $s_1 = s$, $s'_1 = s'$ and $s_k \cong s'_i$ and that we have built partitions of $\sigma$ and $\sigma'$ into corresponding stuttering blocks. Let's assume that $s_k$ belongs to the block $B_m$ and $s'_i$ belongs to the corresponding block $B'_m$. Let $s_k \xrightarrow{a_k} s_{k+1}$. There are two cases. Either $s_{k+1}$ belongs to the new block $B_{m+1}$, or $s_{k+1}$ belongs to the same block as $s_k$, namely $B_m$ (see Fig. 7).

**Case 1.** $s_{k+1}$ belongs to the new block $B_{m+1}$. We show how to select $s'_{i+1}$ such that $s_{k+1} \cong s'_{i+1}$ and how to construct a new block $B'_{m+1}$ (Fig. 7(i)).

Again, there are two cases:

1. $a_k \in \Sigma$, which means that one of the processes executes a transition. Let $i$ be the number of this process and $t \in T_i$ be the executed transition.

First, we show how to select a corresponding transition of the $i$-th process in the system $S'$, such that after its execution suitable relations hold for (a) the control states of the processes, (b) the values of the relevant variables and (c) the transition delays.

(a) There are three cases (depicted in Fig. 8):

Figure 7.

Figure 8.
Next, we show that $\exists t' \in T_i$ such that $source(t') = source(t)$ and $target(t') = target(t)$. We need to show that $t'$ is fireable in $s_i'$. From the Definitions 4.1–4.3 follows that either $guard(t') = guard(t)$, or $guard(t') = true$. Next, from $s_k \cong s_i'$ by item 2 of Definition 4.8 we have $v^k(V' \cup B') = v^j(V' \cup B')$. Therefore, $v^j \models guard(t)$ implies $v^j = guard(t')$. Also according to item 3(c) of Definition 4.7 $delay(t') = delay(t)$ and $urgent(t') = urgent(t)$. Since by item 3 of Definition 4.8 $\tau^k(t) = \tau^j(t')$ and the transition $t$ is fireable, it follows that transition $t'$ is also fireable. Clearly, $q_i^{l+1} = q_i^{k+1}$ after executing the transition $t'$.

ii. $q_i^k \in Q_i \cap Q_i^R$ and $q_i^{k+1} = target(t) \in Q_i^R \setminus Q_i$. In this case there exists $s_{k+m} \in \sigma$ such that $q_i^{k+m} \in Q_i$ and $q_i^{k+1} \xrightarrow{in} q_i^{k+m}$. According to item 3(c) of Definition 4.7 there exists $t' \in T_i$ such that $source(t') = source(t)$ and $target(t') = q_i^{k+m}$. We can show that $t'$ is fireable using the same arguments as in the previous case. We see that $q_i^{l+1} = q_i^{k+m}$ after executing the transition $t'$ and $q_i^{k+1} \xrightarrow{in} q_i^{l+1}$.

iii. $q_i^j \in Q_i \setminus Q_i^R$ and $q_i^{k+1} = target(t) \in Q_i \setminus Q_i^R$. According to item 3(c) of Definition 4.7 we can select $t' \in T_i$ such that $target(t') = source(t')$ and $source(t') \xrightarrow{in} q_i^k$.

The transition $t'$ is fireable, because $guard(t') = true$ and $delay(t') = [0, \infty]$. By item 1(c) of Definition 4.8 $q_i^{k+1} \xrightarrow{in} q_i^k$ or $q_i^k = q_i^{k+1}$. In both cases $q_i^{l+1} \xrightarrow{in} q_i^{k+1}$. Obviously $q_i^{l+1} = q_i^k$ after executing the transition $t'$.

Note, that $q_i^{j+1} = q_i^j$ and $q_i^{j+1} = q_i^{l+1}$ for $j = 1, \ldots, n$ such that $i \neq j$.

As far we have shown that item 1 of Definition 4.8 is satisfied for $s_{k+1}$ and $s_{l+1}$.

(b) Next, we show that $v^{k+1}(V' \cup B') = v^{l+1}(V' \cup B')$. According to item 3(c)iii. of Definition 4.7 the action of the transition $t'$ selected above is composed of all relevant operations of the transition $t$, so all operations on variables and buffers from $V' \cup B'$ are included in the action. Since $v^k(V' \cup B') = v^j(V' \cup B')$, after executing the same sequence of operations we have that $v^{k+1}(V' \cup B') = v^{l+1}(V' \cup B')$. Thus item 2 of Definition 4.8 is satisfied for $s^{k+1}$ and $s^{l+1}$.

(c) Finally, we show that for $j = 1, \ldots, n$ for all $j = Q_j^l \cap Q_j^R$ and for all $t \in out(q)$ there exists $t' \in out(q)$, such that if $enabled(t, q_j^{k+1}, v^{k+1})$ then $\tau^j(t') = \tau^k(t)$. Since $source(t) \in Q_j \cap Q_j^R$, we need only to consider the first case of item 1(a). In case of $a_k \in \Sigma$, $\tau(t)$ can be changed (set to 0) only when $t$ is newly enabled in $s^{k+1}$. There are two possible ways to enable such a transition:

i. $t \in T_i$, where $i$ is the number of the process executing the transition with the label $a_k$.

   In this case we have $\neg enabled(t, q_j^k, v^k)$ or $label(t) = a_k$ and $enabled(t, q_j^{k+1}, v^{k+1})$, which means that $source(t) \neq q_j^k$, or $v^k \not\models guard(t)$, or $label(t) = a_k$ and $source(t) = q_j^k$ and $v^{k+1} \models guard(t)$. According to item 3(c) of Definition 4.7 there exists $t' \in out(q)$, such that $source(t') = source(t)$. Since $q_j^k \in Q_j \cap Q_j^R$, by item 1(a) of Definition 4.8 $q_j^l = q_j^k$, so $source(t) = q_j^k$ iff $source(t') = q_j^k$. Since $q_j^{l+1} = q_j^{k+1}$, as we have shown in item 1(a), it follows that $source(t) = q_j^{k+1}$ iff $source(t') = q_j^{l+1}$. It is easy to see that if $label(t) = a_k$, then there is $t'$ such that $label(t') = a_j$. Since either $guard(t') = guard(t)$, or $guard(t') = true$, and $v^{k+1}(V' \cup B') = v^{l+1}(V' \cup B')$ we
have that \( v^{k+1} \vdash \text{guard}(t) \) implies \( v^{l+1} \vdash \text{guard}(t') \). If \( \text{guard}(t') = \text{guard}(t) \), then \( v^k \vdash \text{guard}(t) \) iff \( v^l \vdash \text{guard}(t') \). Otherwise we have \( \text{source}(t') \neq q_i^j \) or \( \text{label}(t') = a_k \). To see this suppose that \( \text{source}(t') = q_i^j \). In this case the transition with the label \( a_k \) is a self-loop and according to item 2 of Definition 4.2 \( \text{guard}(t') = \text{guard}(t) \), which leads to a contradiction.

ii. \( t \in T_j \), where \( j \neq i \). In this case \( \text{source}(t) = q_j^{k+1} = q_j^l \), which forces \( v^k \not\vdash \text{guard}(t) \) and \( v^{k+1} = \text{guard}(t) \). It means that \( i \)-th process changes values of some global variables or buffers. Next, it exists \( t' \) such that \( \text{source}(t') = q_j^{l+1} = q_i^l \). According to item 3 of Definition 4.3 if the guard of a transition references to a global variable or a buffer, then all operations of the guard are included in \( A^v \). Thus, \( v^k \vdash \text{guard}(t) \) iff \( v^l \vdash \text{guard}(t') \) and \( v^{k+1} \vdash \text{guard}(t') \) iff \( v^{l+1} \vdash \text{guard}(t') \).

This shows that there exists a transition \( t' \in T \) as required in item 3 of Definition 4.8.

What we have shown so far is how to choose a transition \( a_k \) in the system \( S' \) such that \( s_i^l \xrightarrow{a_k} s_i^{l+1} \) and \( s_k + 1 \equiv s_i^{l+1} \) in case of untimed transition \( a_k \) in the system \( S \).

2. \( a_k \in \mathbb{R}_+ \), which means that the transition reflects time passage. In this case \( q_i^{k+1} = q_i^k \) for \( i = 1, \ldots, n \), \( v^{k+1} = v^k \) and \( \tau^{k+1} = \tau^k + a_k \). We recall that timed transition is possible in the state \( s^{k} \) for \( a_k > 0 \), if for all \( i = 1, \ldots, n \) and for all \( t \in T_i \) one of the following holds: \( \neg \text{enabled}(t, q_i^k, v^k) \) or \( \neg \text{urgent}(t) \) or \( (\tau^k(t) + a_k) \in \text{delay}(t) \).

By item 3 of Definition 4.8 for each process \( i = 1, \ldots, n \) for all \( q \in Q_i \cap Q_i^R \) and for all \( t \in \text{out}(q) \), there exists \( t' \in T_i \) such that \( t' \in \text{out}(q) \) and if \( \text{enabled}(t, q_i, v^k) \), then \( \tau^k(t) = \tau^k(t') \).

We show that the same timed transition is also possible from the state \( s_i^{l'} \), that is for any \( t' \in T_i \), where \( i = 1, \ldots, n \), \( a_k \) units of time may pass, that is \( \neg \text{enabled}(t', q_i^{l'}, v^{l'}) \) or \( \neg \text{urgent}(t') \) or \( (\tau^k(t') + a_k) \in \text{delay}(t') \).

According to item 3.c of Definition 4.7 for any \( t' \in T_i \) there is \( t \in T_i \) such that \( \text{source}(t') = \text{source}(t) \). There are three cases:

(a) \( \text{source}(t') \in Q_i \cap Q_i^R \) and \( \text{guard}(t') = \text{guard}(t) \). To see that \( \neg \text{enabled}(t, q_i^k, v^k) \) implies \( \neg \text{enabled}(t', q_i^{l'}, v^{l'}) \), note, that since \( \text{guard}(t') = \text{guard}(t) \), we get \( v^k \vdash \text{guard}(t) \) iff \( v^{l'} \vdash \text{guard}(t') \). Next, we observe that there are two cases for \( q_i^k \): Either \( q_i^k \in Q_i \cap Q_i^R \), or \( q_i^k \in Q_i \setminus Q_i^R \). The case of \( q_i^k \in Q_i \setminus Q_i^R \) is impossible because according to item 1 of Definition 4.2 at least one transition from the state \( q_i^k \) must be enabled and according to Definition 4.3 for all \( t \in \text{out}(q_i^k) \), \( \text{delay}(t) = [0, 0] \) and \( \text{urgent}(t) = \text{true} \), so a timed transition would be impossible in such a state. If \( q_i^k \in Q_i \cap Q_i^R \), then \( q_i^k = q_i^{l'} \) and we have that \( \text{source}(t') \neq q_i^k \) implies \( \text{source}(t') \neq q_i^{l'} \). In the other case as shown at the beginning of the proof \( q_i^l \in Q_i \setminus Q_i^R \) and since \( \text{source}(t) \in Q_i \cap Q_i^R \), it follows that \( \text{source}(t') \neq q_i^l \).

By item 3(c)iv. we have that \( \text{delay}(t') = \text{delay}(t) \) and \( \text{urgent}(t') = \text{true} \). Thus, if \( \text{enabled}(t, q_i^k, v^k) \) and \( \text{guard}(t') + a_k \in \text{delay}(t') \), then \( (\tau^k(t') + a_k) \in \text{delay}(t') \). Obviously, \( \neg \text{urgent}(t') \) gives \( \neg \text{urgent}(t') \).

(b) \( \text{source}(t') \in Q_i \cap Q_i^R \) and \( \text{guard}(t') \neq \text{guard}(t) \). This situation is possible only when none of the states from \( Q_i \) is time nor control dependent on the state \( \text{source}(t') \). Hence, according
to item 1 of Definition 4.2 there is no self loop from \( source(t) \) and there is at least one enabled transition \( t_a \in out(source(t)) \). As in the previous case we have that \( delay(t') = delay(t) \) and \( urgent(t') = urgent(t) \). According to Definition 4.3 \( delay(t_a) = delay(t) \) and \( urgent(t_a) = urgent(t) = true \). Since \( (\tau^k(t_a) + a_k) \in delay \uparrow (t_a) \) and \( \tau^l(t') = \tau^k(t_a) \), we have \( (\tau^l(t') + a_k) \in delay \uparrow (t') \).

(c) \( source(t') \in Q'_i \setminus Q^R_i \). In this case according to item 3(c)iv. of Definition 4.7 \( delay(t') = [0, \infty) \), so for arbitrary \( a_k \) we have \( (\tau^l(t') + a_k) \in delay \uparrow (t') \).

This completes our claim that being at the state \( s^l \) the timed transition of duration \( a_k \) is possible for any \( t' \in T' \).

Note that we have actually proved that for \( i = 1, \ldots, n \), for all \( q \in Q'_i \cap Q^R_i \) and for all \( t \in out(q) \), there exists \( t' \in T'_i \) such that \( t' \in out(q) \) and if \( enabled(t, q^{k+1}_i, v^{k+1}) \), then \( \tau^{l+1}(t') = \tau^l(t') + a_k = \tau^k(t) + a_k = \tau^{k+1}(t) \). It is clear that after executing a timed transition \( q^{l+1}_i = q^l_i \) for \( i = 1, \ldots, n \) and \( v^{l+1} = v^l \). Combining these we get \( s_{k+1} \cong s^l_{i+1} \).

In both cases the new block \( B'_{m+1} \) is composed exclusively of the state \( s^l_{i+1} \) selected above.

**Case 2.** \( s_{k+1} \) belongs to the same block as \( s_k \), namely \( B_m \). We show that \( s_{k+1} \cong s^l_i \).

1. First we show that suitable relations hold the for control states. Since \( s_{k+1} \) belongs to the same block as \( s_k \), we have \( q^{k}_i \in Q^R_i \setminus Q'_i \), so by item 1(b) of Definition 4.8 \( q^k_i \overset{inv}{\rightarrow} q^l_i \). Again, there are two cases (see Fig. 9):

(a) \( q^{k+1}_i \in Q^R_i \setminus Q'_i \). In previous inductive steps (item 1(a)ii. of case 1) we have chosen \( q^l_i \) such that \( q^{k+1}_i = q^l_i \).

(b) \( q^{k+1}_i \in Q^R_i \setminus Q'_i \). In previous inductive steps (item 1(a)ii. of case 1) we have chosen \( q^l_i \) such that \( q^{k+1}_i \overset{inv}{\rightarrow} q^l_i \).
By the above, item 1 of Definition 4.8 is satisfied for \( s_{k+1} \) and \( s'_1 \).

2. Since \( \text{source}(t) \not\in Q_i^f \) for the transition \( t \in T_i \) with the label \( a_k \), it follows that \( \text{opers}(t) \cap A^p = \emptyset \). Hence, \( \text{vars}(t) \cap (V' \cup B') = \emptyset \), which means that the transition \( t \) does not change any relevant variables. Therefore \( v_i^{k+1}(V' \cup B') = v_i^k(V' \cup B') = v_i^k(V' \cup B') \), which satisfied item 2 of Definition 4.8 for \( s_{k+1} \) and \( s'_1 \).

3. Finally, we show that for \( j = 1, \ldots, n \) for all \( q \in Q_j \cap Q_j^R \) and for all \( t \in \text{out}(q) \), there exists \( t' \in T_i \) such that \( t' \in \text{out}(q) \) and if \( \text{enabled}(t, q_i^{k+1}, v_i^{k+1}) \), then \( \tau^i(t') = \tau^i(t) \).

Such a transition \( t \) can be newly enabled only if \( t \in T_i \), where \( i \) is the number of the process executing the transition. Transitions of other processes cannot be enabled by the transition with the label \( a_k \), because this transition does not change relevant variables and according to item 3 of Definition 4.3 global variables and buffers included in guards of the transitions going from the states included in \( Q_i \cap Q_i^R \) are relevant. Thus, \( \tau^i(t') = \tau^i(t) \) is set to 0, if \( \text{source}(t) \not\in q_i^k \) or \( v_i^k \neq \text{guard}(t) \) or \( \text{label}(t) = a_k \) and \( \text{source}(t) = q_i^{k+1} \) and \( v_i^{k+1} = \text{guard}(t) \). According to item 3.c of Definition 4.7 for all \( q \in Q_j \cap Q_j^R \) and for all \( t \in \text{out}(q) \) there exists \( t' \in T_i \), such that \( t' \in \text{out}(q) \) and \( \text{source}(t') = \text{source}(t) \). Since \( \text{source}(t') = q_i^{k+1} \) and \( q_i^{k+1} = q_i^k \), \( \text{source}(t') = q_i^k \). There are two cases: either \( \text{guard}(t') = \text{guard}(t) \), or \( \text{guard}(t) = \text{true} \), thus if \( v_i^{k+1} = \text{guard}(t) \), then \( v_i^k = \text{guard}(t') \). All we need to show is that \( \tau^i(t') = 0 \). To see this, we need to look at the prefixes of \( \sigma \) and \( \sigma' \). According to the previous inductive steps in the part of \( \sigma \) from the state \( s_{k-m} \), such that \( a_{k-m} \in \Sigma_i, q_i^{k-m} \in Q_i^f \) and \( q_i^{k-m} \xrightarrow{m} q_i^{k+1} \), to the state \( s_{k+1} \) there is no timed transitions, because along this path for \( q_i^{k-1} \in Q_i^R \), where \( k-m < j < k+1 \), for all \( t \in \text{out}(q_i^k) \), \( \text{delay}(t) = [0, 0] \) and \( \text{urgent}(t) = \text{true} \). If so, then there is no time transition from the state \( s_{k-m} \) such that \( s_{k-m} \cong s'_1 \), to the state \( s'_1 \). Also \( q_i^{k-m} = q_i^{k+1} \). According to item 4 of Definition 4.6 there can be no invisible cycle, so it must hold either \( q_i^j \neq q_i^{l-r} \) or \( a_i = a_{l-r} \). In both cases the transition \( t' \) is enabled at the state \( s'_1 \) or later. If so, then \( \tau^i(t') \) is set to 0 at the state \( s'_1 \) or later and from the state \( s'_1 \) there has been no timed transition in \( \sigma' \). From this it follows that item 3 of Definition 4.8 is also satisfied.

According to the construction described above each of the blocks \( B_1, B_2, \ldots, B_i, B_2, \ldots \) is finite, because there is a finite number of processes and by item 2 of Definition 4.2 invisible paths of each process are finite.

For each path \( \sigma' \) in \( M' \) starting at \( s' \) we can construct a path \( \sigma \) in \( M \) starting at \( s \) and corresponding matching blocks. The construction is dual to this one presented above. For each transition of \( \sigma' \) we can show a finite sequence of transitions in \( M \) as required in Definition A.1.

Thus, the conditions of Definition A.1 are satisfied and the proof is complete.

\[ \square \]