A Time Abstraction Method for Efficient Verification of Communicating Systems

Eric Verlind, Tilman Kolks, Gjalt de Jong, Bill Lin and Hugo De Man

IMEC, Kapeldreef 75, B-3001 Leuven, Belgium

Abstract — *An important practical approach to automatic verification of finite state concurrent systems is temporal logic model checking. However, a major barrier towards wider application of such methods is the state explosion problem that often occurs during the composition of complex communicating systems. In addition to being large, many systems have very deep state spaces as well.*

In this paper, we propose an abstraction method based on time abstraction *which can significantly reduce both the size as well as the depth of the state space that must be explored during model checking. It is especially suited for applications involving systems that are loosely coupled in the sense that communication activity is relatively sparse, such as applications involving DSPs, which have large intervals of autonomous processing, with communication activity in between. All properties expressible with the temporal logic CTL* or CTL*-X are* strongly preserved *under the proposed abstraction. In particular, we give a method that can replace a* chain *of states by a single state with a* time label*. The abstracted model is represented by means of a* timed label transition system *model. We give a* composition algorithm *for composing the individually time abstracted models to form a global model that can be converted back to a conventional, but potentially much reduced, state space suitable for model checking. Furthermore, a similar approach can be followed in dealing with wait states in which a system is idling while waiting for an escape signal. For many cases, our approach is successful in alleviating the state explosion problem that often arises in composition of communicating systems.*

I Introduction

Verification of communicating digital hardware is an essential task in the development of correctly working complex systems. Today, simulation is still the principal method of design validation, but it is known to be a very time consuming task, not feasible to check exhaustively for realistic cases. Formal verification based on formal property checking is an alternative approach to this problem that can potentially offer more confidence than simulation. An important and practically feasible approach to automatic verification of finite state concurrent systems is temporal logic model checking. However, a major barrier for such methods is the state explosion problem that often occurs when composing complex communicating systems. In addition to being enormous, many systems often have very deep state spaces as well. While *symbolic* methods based on Binary Decisions Diagrams (BDDs) have enabled much progress, they have mainly been successful in tackling state explosion due to *data paths* and *arithmetic computations* where the state space is relatively regular and shallow [5, 4, 7, 11, 13].

Alternatively, researchers have investigated into *abstraction* methods to reduce state space complexity. Abstraction methods such as those proposed by [6, 10, 2, 12] are *conservative* in the sense that *false-negatives* can occur. That is, the verifier may report that an error may occur in the *abstracted* model where in fact the error will never occur in the *original* system. For a number of verification tasks, these conservation approaches may not lead to meaningful results.

In today's applications such as those found in mobile radio, systems of various nature cooperate in executing an overall highly complex functionality. In such a heterogeneous system, individual DSP processors perform subtasks, executing algorithms requiring a substantial number of computation cycles in each computation frame due to the complexity of the calculation. During the computation, there is no interaction with the environment, as only before and after such a processing phase communication occurs. Therefore, such systems are characterized by relatively sparse communication activity separating long time intervals during which no externally visible activity occurs.

Hence, for such loosely coupled data processing systems, representing these long chains of "uninteresting" states more efficiently will significantly reduce the number of items in the systems in a natural way. Procedures such as reachability analysis will seriously benefit from our approach, as the depth of the search space of the abstract representation will potentially be drastically reduced.

This paper proposes an abstraction method based on *time abstraction* which can significantly reduce both the size and the depth of the state space. The method is *strongly preserving* in the sense that a property *holds* in the abstracted model *if and only if* it holds in the original system. We propose a method of representation which represents these chains more efficiently by replacing each chain by a single so-called *timed state*, in which a delay value equal to the length of the chain is associated. We formalize this into a simple abstraction model called a *timed label transition system (TLTS)*. Although there are more powerful and elaborate *time automata* models (e.g. [1]), the goal of our model is different – it is intended to provide a means for abstraction and efficient verification, not to address timing issues *per se*.

The abstraction method can be applied to a system of communicating modules by abstracting each module to its *timed abstracted* form *before* composition. We give a *composition algorithm* for composing these timed abstracted modules together to form a global state space that can be converted back to a conventional state space, but one that may be substantially *smaller* and *much less deep* than the original global state space without abstraction. In some cases, the representation can be several orders of magnitude smaller than the original one.

We will illustrate our approach by means of a simple example involving composition of two communicating processors P_1 and P_2 as depicted by figure 1. Both processors execute a signal processing algorithm, as shown by figure 2: P_1 calculates a weighted average of the 8 latest samples, while P_2 computes an approximation of the natural logarithm of its input. The bracketed figures at the end of a line indicate the number of required execution cycles. The only relevant signal for communication is the *write* signal which signals to P_2 that data can be picked up. The state graphs that correspond with the system are shown by figure 3. A useful correctness check expressed in CTL [8, 3] on the system would be:

$$\text{AG}\,(write \Rightarrow (\text{state\_P}_2 = id)).$$

This property states that whenever P_1 is issuing a *write*, P_2 is always at the idle state *id* waiting to accept the data. This is in fact a *safety property*. In this case, we see that this is always true because P_1 has over 100 cycles to perform between writes whereas P_2 has only about 30 cycles between reads, and therefore P_2 will always be in time to accept the next data.

In this example, both processors contain a chain, a linear sequence of states in which no decisions are taken, nor external output labels change. For P_1, it is due to the *for-loop*, and for P_2, it is due to the computation in the routine LN_approx. In both cases, the computations are internal. These chains are candidates for abstraction.

In [12], a conservative reduction method was propose for replacing such chains, or counter structures, by a self-looping state that nondeterministically repeats. Though it has been shown to be useful

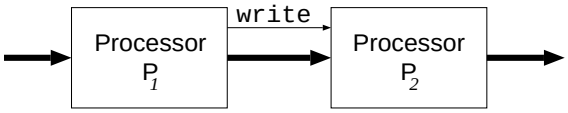

Figure 1: Time abstraction example schematic

```
process P1 {
    forever {
        p = (p + 1) MOD 8; a[p] = read; (1)
        f = 0; (1)
        for q = 0 to 7
            f = f + w[q] * a[(p − q) MOD 8]; (13)
        write (f >> 3); (1)
    }
}

process P2 {
    forever {
        do nop while !write;
        a = read; (1)
        sa = LN_approx(a); (30)
        write sa; (1)
    }
}
```

Figure 2: Processing in $P1$ and $P2$

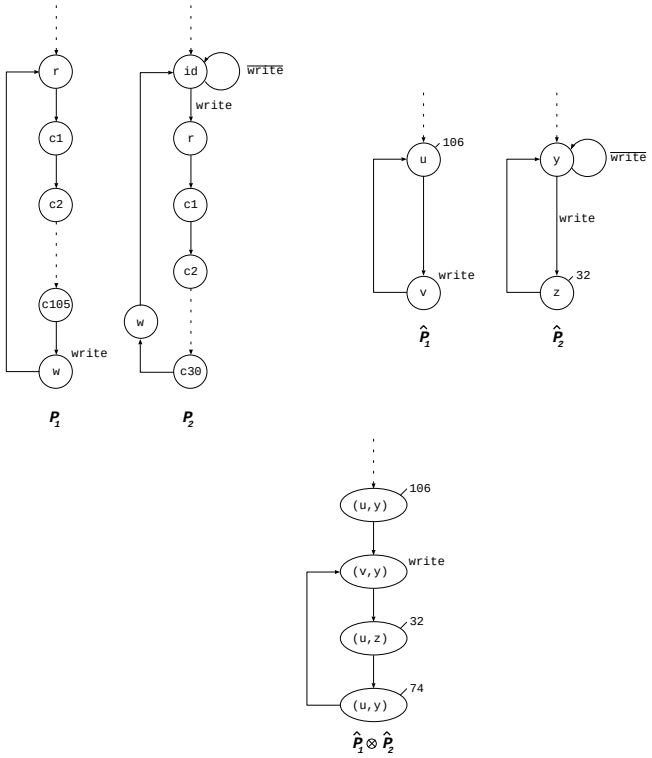

Figure 3: State graphs

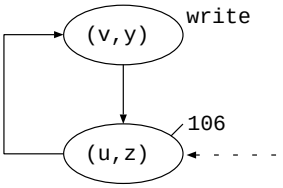

Figure 4: Maximally reduced timed composed machine

for other types of verification tasks, this type of reduction would not allow us to get a meaningful answer to the question posed by the aforementioned CTL formula. This results from the fact that the added but un-present behavior causes the model check to produce false negatives. On the other hand, our method would find the system to operate correctly. Figure 3 shows $\hat{P}_1$ and $\hat{P}_2$ being the abstracted systems, in which each of the chains is represented as a single *timed state* with a time label corresponding to the length of the chain. The chain of P_1 contains 106 operations whereas P_2 contains 32 operations. The time abstracted model can then be composed together to form a global model using a composition algorithm for this time model. The composed model is shown in figure 3; its size is over two orders of magnitude smaller than that of the untimed composed system. Moreover, its *depth* is also over two orders of magnitude shorter, which is an important consideration for BDD-based implicit enumeration methods.

To avoid false negatives in verification, taking account of timing information is crucial for composition in many cases. As already illustrated with the example, our method strongly preserves the properties of CTL* [3], hence it is meaningful to perform model checking on the timed system description obtained by composition. However, properties to be checked on the composed machine are usually *qualitative* and can therefore be formulated in a temporal logic without using the *next* operator, such as CTL*-X or CTL-X . Therefore, *after* application of our composition procedure, the composed system can be converted back to an ordinary untimed system description by disregarding the time labeling. This description can then be checked using a model checker such as *smv* [11] for FairCTL, which contains CTL-X .

The sections to follow elaborate the approach outlined above. Section II gives the definition of timed label transition systems. The subject of section III is the composition of timed label transition systems. Section IV treats the strong property preserving character of our method. To show the feasibility of our approach, section V discusses the application of the proposed techniques to two design examples. Finally, section VI concludes on the approach.

II Timed label transition systems

A Definitions

A finite state system has a number of signal names, some are input, other are output. We assume that we select *subsets* I and O of input and output signals which we consider relevant for communication or model checking.

Definition 1 Let W be a set of unique signal names $w_1, ..., w_n$. The set of **symbols** $\mathcal{S}(W)$ over W is defined to be the set of *minterms* over W.

Definition 2 A **label transition system** (LTS) is a structure

$$M = (S, I, O, R)$$

- S is the set of *states*;
- I is a set of *input signal names*;
- O is a set of *output signal names*;
- $R \subseteq S \times \mathcal{S}(I \cup O) \times S$ is the *transition relation.*

$A = \mathcal{S}(I \cup O)$ is the *alphabet* of M. In this definition, we make a distinction between input and output signals, as this is important in composition of systems, described later. We use a *Moore* machine model, therefore all outgoing edges of a particular state s have identical output labels.

In the following the notation α denotes a symbol in an alphabet $\mathcal{S}(I \cup O)$, $\alpha_{in} \in \mathcal{S}(I)$ an input symbol and $\alpha_{out} \in \mathcal{S}(O)$ an output symbol.

Definition 3 Given an LTS M. A **partial chain** c is a sequence of states $\pi = s_1, s_2, ..., s_n$, with $s_i \in S$, such that:

- For all $i, 1 \leq i < n, (s_i, \alpha, s') \in R \Rightarrow (s' = s_{i+1})$.
- For all $i, 1 < i \leq n, (s', \alpha, s_{i+1}) \in R \Rightarrow (s' = s_i)$.
- There exists exactly *one* output symbol $\alpha_{out,c} \in \mathcal{S}(O)$ such that: $\forall s_i \in \pi . \forall \alpha_{in} \in \mathcal{S}(I)$. $[(s_i, \alpha_{in} \times \beta_{out}, s') \in R \Rightarrow (\beta_{out} = \alpha_{out,c})]$.
- Given $\alpha_{out,c}$ as above, $\forall \alpha_{in} \in \mathcal{S}(I) . \forall (s_i, s_{i+1}) \in \pi$. $[(s_i, \alpha_{in} \times \alpha_{out,c}, s_{i+1}) \in R]$.

The *delay* of a partial chain c, $\delta(c)$, is equal to its length n. *first*(c) and *last*(c) denote the first, resp. the last state of partial chain c.

Definition 4 Given an LTS M. A **maximum chain** or simply **chain** c is a partial chain c which is not strictly contained in any partial chain c'.

Definition 5 Given an LTS M. A **nontrivial chain** c is a chain c, for which $\delta(c) \geq 2$. A **trivial chain** is a chain c, for which $\delta(c) = 1$ (this is a single state).

Definition 6 A **timed label transition system** (TLTS) is a structure

$$T = (S, I, O, R, \tau)$$

- S is the set of *timed states*;
- I is a set of *input signal names*;
- O is a set of *output signal names*;
- $R \subseteq S \times \mathcal{S}(I \cup O) \times S$ is the *transition relation*;
- $\tau : S \rightarrow N^+$ associates a *delay* with every state.

$A = \mathcal{S}(I \cup O)$ is the *alphabet* of the TLTS.

The states $s \in S$ represent either trivial chains, in which case $\tau(s) = 1$, or nontrivial chains, in which case $\tau(s) \geq 2$.

Definition 7 Given a TLTS T, consider the set of **simple states** $Q = S \times N^+ = \{q = (s, t) \mid s \in S \wedge 1 \leq t \leq \tau(s)\}$. One can see these as the states implicitly represented by the timed states.

Consider a simple state $q = (s, t)$. There are two cases to be considered for the set $Q_n(q)$ of next simple states:

$$Q_n(q) = \begin{cases} \{(s, t+1)\} & \text{if } t < \tau(s) \\ \{(s', 1) \mid (s, a, s') \in R\} & \text{if } t = \tau(s) \end{cases}$$

The first case corresponds to a transition internal to a chain, the second case to transitions from the last simple state of a chain to the first simple state of a chain. Performing a reachability analysis does not require an explicit traversal at the level of the simple states. Instead, the traversal is possible on the higher level of the timed states: after entering a chain structure at its first state $(s, 1)$, one can skip over the internal states and proceed in the next step with the state(s) reachable from its last state, $(s, \tau(s))$.

Definition 8 Given an LTS M and the set C of all chains in M. $\xi : S \rightarrow C$ is the function which associates with each $s \in S$ the chain $c_i \in C$ to which it belongs.

Definition 9 Given M and C as in definition 8 and $\hat{S}$, $|\hat{S}| \geq |C|$ a set of state markings. $\varphi_s : C \rightarrow \hat{S}$ is defined to be the bijective function which maps each chain $c_i \in C$ to an $\hat{s} \in \hat{S}$.

Definition 10 Given $\mathcal{M}$ the class of possible LTSs and $\mathcal{T}$ the class of possible TLTSs, ξ and φ_s. The **time abstraction function** $\Phi : \mathcal{M} \rightarrow \mathcal{T}$ maps a given LTS M as follows to TLTS $T = \Phi(M) = (\hat{S}, I, O, \hat{R}, \hat{\tau})$, with

- $\hat{S} = \{\hat{s} \mid \exists s \in S . \hat{s} = \varphi_s(\xi(s))\}$;
- $\hat{R} \subseteq \hat{S} \times \mathcal{S}(I \cup O) \times \hat{S} =$ $\{(\varphi_s(\xi(s)), \alpha, \varphi_s(\xi(s_n))) \mid R(s, \alpha, s_n) \wedge \xi(s) \neq \xi(s_n)\}$;
- $\hat{\tau} : \hat{S} \rightarrow N^+ . \hat{\tau}(\hat{s}) = \delta(\varphi_s^{-1}(\hat{s}))$.

III Composition

A Introduction

This section discusses the synchronous composition of two (timed) label transition systems. The technique can be generalized straightforwardly to the composition of $n > 2$ systems; it suffices to discuss the two-system case for ease of description. Composition of label transition systems is the subject of the first part of this section. The latter part of it is devoted to describing an algorithm which takes advantage of the explicit timing information present in the description of the constituent TLTSs, for determination of the synchronous product in an efficient way.

B Composition of label transition systems

Definition 11 The **synchronous product** $M = M' \otimes M''$, with $M' = (S', I', O', R')$ and $M'' = (S'', I'', O'', R'')$ label transition systems, is defined as follows:

$$M = (S, I, O, R)$$

- $S \subseteq S' \times S''$;
- $I = (I' \cup I'') \setminus (O' \cup O'')$;
- $O = O' \cup O''$;
- $R = \{(s, a, s_n) = ((s', s''), a, (s'_n, s''_n)) \in S \times A \times S \mid$ $(a = a' \wedge a'') \wedge (s', a', s'_n) \in R' \wedge$ $(s'', a'', s''_n) \in R''\}$.

This definition applies under the restriction that $O' \cap O'' = \emptyset$. The alphabet A of the product machine then equals $\mathcal{S}(I \cup O)$. Note the following:

- an output may not be connected to an other output;
- an input connected to an output is an output;
- an input connected to other inputs, but not to an output, is an input.

C Composition of timed label transition systems

The *synchronous product* $T = T' \otimes T'' = (S, I, O, R, \tau)$, with $T' = (S', I', O', R', \tau')$ and $T'' = (S'', I'', O'', R'', \tau'')$ timed label transition systems, is defined here algorithmically. For deriving the reachable states of the product machine, we apply the algorithm of figure 10.

Because of conciseness considerations, again we use predicate notation. In the algorithm, the S sets denote sets of states (markings), as in the TLTS definition. The Q sets denote sets of states, composed out of markings and time values: $Q' \subseteq S' \times N^+$ and $Q'' \subseteq S'' \times N^+$ for the component systems and $Q \subseteq (S' \times N^+) \times (S'' \times N^+)$ for the product. Each time a timed state pair is investigated, a mapping from that pair to a delay value is added to the τ_q function, by invoking the *add_map* function. Note that, in contrast with definition 6:

- $T = (Q, I, O, R_q, \tau_q)$;
- $Q \subseteq S \times N^+$;
- $\tau_q : Q \rightarrow N^+$;
- $R_q \subseteq Q \times A \times Q$.

The reason for putting a time value in the state here is that we need *both* state marking and time to indicate a (simple) state uniquely during this calculation. After applying the algorithm, we could do a renaming, in which each state $q \in S \times N^+$ gets a unique mapping out of a set $\hat{S}$ (see definition 10). Finally we have a delay value associated with each reachable state in the product machine.

Apart from chains, in systems self-looping wait states occur, where the system stays idling until it is allowed to proceed. In an extension of the composition algorithm given, we deal with these as well, based on the observation that in a sense these wait states can be regarded as infinite chains with a preemption arc.

IV Preservation of properties

A Equivalence and stuttering equivalence

In the remainder, the following definitions of *equivalence* and *stuttering equivalence* are used, as taken from [3].

Definition 12 Given two Kripke structures K' and K'' with set of atomic propositions AP, we define the **equivalence** relation $\simeq \subseteq S' \times S''$ as follows, for all $(s', s'') \in S' \times S''$:

$$
\begin{aligned}
& s' \simeq s'' \iff \\
& \qquad (L'(s') = L''(s'')) \wedge \\
& \qquad \forall s'_0.[s' R' s'_0 \Rightarrow \exists s''_0.[s'' R'' s''_0 \wedge s'_0 \simeq s''_0]] \wedge \\
& \qquad \forall s''_0.[s'' R'' s''_0 \Rightarrow \exists s'_0.[s' R' s'_0 \wedge s'_0 \simeq s''_0]]
\end{aligned}
$$

Two states s' and s'' are equivalent iff $s' \simeq s''$.

Definition 13 Given two Kripke structures K' and K'' with set of atomic propositions AP, we define the **stuttering equivalence** relation $\simeq_S \subseteq S' \times S''$ as follows, for all $(s', s'') \in S' \times S''$:

$$
\begin{aligned}
& s' \simeq_S s'' \iff \\
& \quad (L'(s') = L''(s'')) \wedge \\
& \quad \forall s'_0.[s' R' s'_0 \Rightarrow \exists (n \geq 0).\exists s''_0, ..., s''_n. \\
& \qquad [s'' = s''_0 \wedge \\
& \qquad \forall i, 0 \leq i < n.[s''_i R'' s''_{i+1} \wedge s' \simeq_S s''_i \wedge s'_0 \simeq_S s''_n]]] \wedge \\
& \quad \forall s''_0.[s'' R'' s''_0 \Rightarrow \exists (n \geq 0).\exists s'_0, ..., s'_n. \\
& \qquad [s' = s'_0 \wedge \\
& \qquad \forall i, 0 \leq i < n.[s'_i R' s'_{i+1} \wedge s'' \simeq_S s'_i \wedge s''_0 \simeq_S s'_n]]]
\end{aligned}
$$

B Strong property preservation by our method

In this paper, a procedure was discussed which first abstracts to TLTSs with time abstraction function Φ two given LTSs and then composes them. We claim some results of strong property preservation for our method, as formalized in theorems 1 and 2.

As the above equivalence relations and model checking are defined w.r.t. Kripke structures, we transform an LTS M to a Kripke structure $K = (S_K, R_K, L_K)$, with the set AP of atomic propositions equal to the set of symbols $\mathcal{S}(I \cup O)$. Similarly, we transform a TLTS T to a Kripke structure, via the notion of *simple states*. These conversion procedures are not further discussed here. Note however, that in the following lemmas and theorems, if we refer to a LTS or TLTS, we assume with no further comment that first a conversion to a Kripke structure was done.

Lemma 1 $\Phi(M) \simeq M$

Proof: A timed state $\hat{s} = \varphi_s(c_i)$ is present in $\Phi(M)$ representing chain c_i in M, being a sequence $s_1, ..., s_{\delta(c_i)}$. The sequence of *simple states* $q_1, ..., q_{\delta(c_i)}$ corresponding with $\hat{s}$ is in fact identical to the chain c_i. It is then easily seen that the structures are identical but for names of the states. Hence, they are equivalent. □

Lemma 2 $(\Phi(M') \otimes \Phi(M'')) \simeq (M' \otimes M'')$

Proof: The composition of TLTSs is described by the algorithm of figure 10. Assume that in the algorithm we are at simple state $q = q_t = (q', q'')$, with $q' = (s', t')$ implicitly represented by $\Phi(M')$ and $q'' = (s'', t'')$ implicitly represented by $\Phi(M'')$. Assume that the remaining part of the chain in M' is smaller than that in M'' ($l' < l''$). We can then proceed until a timed state change occurs, i.e. we arrive in $((s'_n, 1), (s'', t'' + l''))$. In the algorithm we perform this as *one* transition, which however corresponds with passing all simple states in between, i.e. as it were the algorithm operates on the 'stretched' structure of simple states. The structure M and this structure of simple states implicitly present in $\Phi(M)$ can then be seen to be equal, which implies equivalence. □

A consequence of lemmas 1 and 2 is the following, expressing that we can apply the abstraction before as well as after composition.

Lemma 3 $(\Phi(M') \otimes \Phi(M'')) \simeq \Phi(M' \otimes M'')$

Definition 14 $\Psi : \mathcal{T} \rightarrow \mathcal{M}$ maps a given TLTS to an LTS as follows: $\Psi((S, I, O, R, \tau)) = (S, I, O, R)$. In other words, Ψ removes the time labels from a TLTS.

Lemma 4 $(T \simeq M) \Rightarrow (\Psi(T) \simeq_S M)$

Proof: From definitions 4 and 13 follows that all states in a chain are stuttering equivalent. Removing or adding internal states will not change this. Therefore, removing the time label by Ψ, which can be interpreted as reducing the chain to a trivial one, preserves stuttering equivalence. Furthermore, from definitions 12 and 13 follows that equivalence implies stuttering equivalence. Hence, this lemma. □

Lemma 5 $\Psi(\Phi(M') \otimes \Phi(M'')) \simeq_S (M' \otimes M'')$

From [3] we have the following two lemmas.

Lemma 6 Given two Kripke structures with initial states s_0 and s'_0 respectively.
$s_0 \simeq s'_0 \Leftrightarrow \forall f \in \text{CTL}^* .(K, s_0 \models f \Leftrightarrow K', s'_0 \models f)$

Lemma 7 Given two Kripke structures with initial states s_0 and s'_0 respectively.
$s_0 \simeq_S s'_0 \Leftrightarrow \forall f \in \text{CTL}^*\text{-}X .(K, s_0 \models f \Leftrightarrow K', s'_0 \models f)$

The following theorem expresses the *strong preservation* of the properties of CTL* under our chain abstraction.

Theorem 1 $\forall f \in \text{CTL}^* : (\Phi(M') \otimes \Phi(M'') \models f) \Leftrightarrow (M' \otimes M'' \models f)$

The following theorem expresses the *strong preservation* of the properties of CTL*-X under chain abstraction and discarding of time information *after* composition.

Theorem 2 $\forall f \in \text{CTL}^*\text{-}X : (\Psi[\Phi(M') \otimes \Phi(M'')] \models f) \Leftrightarrow (M' \otimes M'' \models f)$

V Examples

A Two-machine communicating DSP example

This case study deals with the problem of verifying the communication between two signal processing elements. It is a simplified version of a realistic situation involving two signal processors. We assume here that such a system, needing 100,000 states is connected to another DSP using 87,000 machine cycles to complete its frame computation. These numbers are realistic for this type of application. The reason for such large number of cycles is due to a number of *nested for-loops* in the algorithm. Figure 5 shows the hardware configuration.

Data processors P_1 and P_2 (figure 6) communicate via the two-place buffer in between (figure 7), making it a three machine problem. After initialization, P_1 operates in *steady state* mode, processing frame by frame. At the beginning of such a frame, P_1 reads in 180 data items and then computes 3 output items in 100,000 clock cycles of processing. These are then transferred in a burst following a condition-less strobe signal via the buffer to P_2, which then can start its processing.

Figure 8 shows the state graphs of the processors as abstracted using chain abstraction, where nontrivial chains c are shown as timed

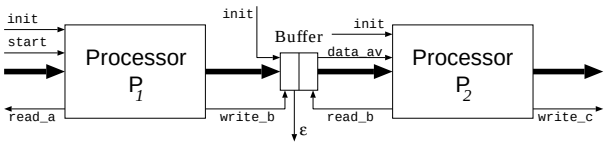

Figure 5: Example circuit

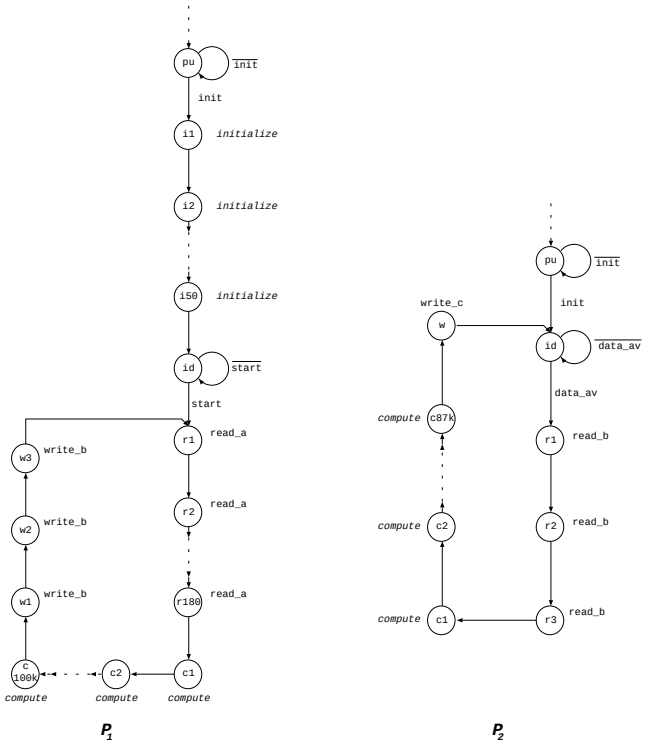

Figure 6: State graphs of processors

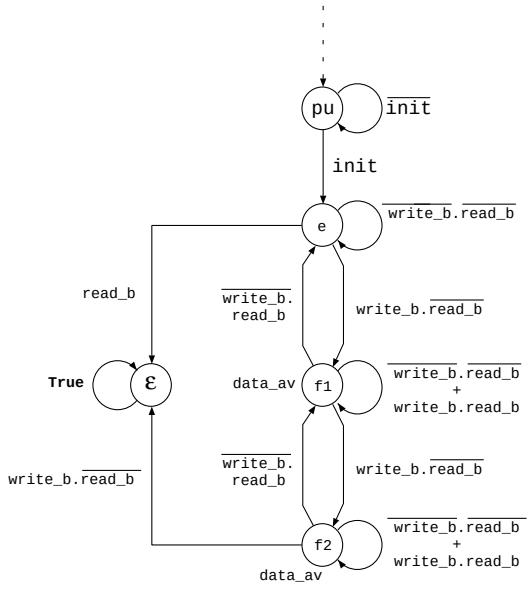

Figure 7: Two-element buffer state graph

states $\hat{s}$ annotated with a time label $\tau(\hat{s})$. P_1 and P_2 have 100,235 and 87,006 states, respectively, practically all residing in chains, while the buffer has 5 states. The number of product states, including unreachable ones, amounts to $4.46.10^{10}$. The number of *reachable states* in the product machine is 187,237. Applying our time abstraction and composition algorithm, using abstractions for both chains and self-looping wait states, leads to a timed representation of 13 timed states, of which 7 represent single states and 6 represent nontrivial chains. A necessary correctness condition expressed in CTL*-X is AG $\neg\varepsilon$, expressing that no buffer overflow nor underflow may occur. The description of the composed machine yielded by our procedure does not contain any state in which the buffer is in state ε, so we know that our correctness condition is fulfilled without further checking. Another useful check is checking whether the buffer could be completely filled (EF (buffer_state = *f2*)), requiring 0.21 s of CPU time using *smv* on a DECStation 5000/120. Therefore, we conclude that our method has been successful in reducing a large problem into an almost trivial one.

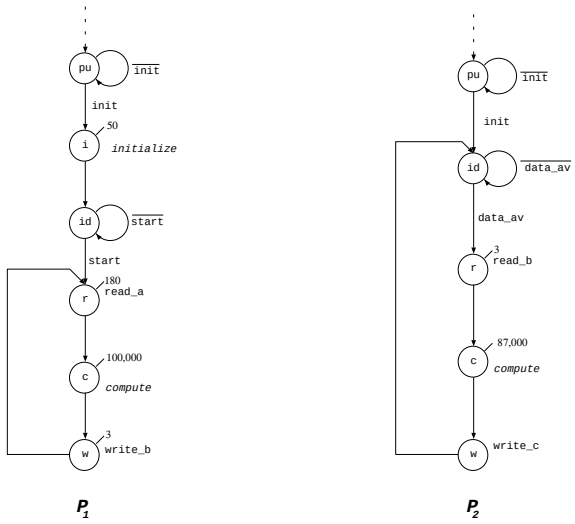

Figure 8: Timed state graphs of processors

B Mobile terminal example

To further demonstrate our method, we consider now a part of a mobile terminal transmitter. Figure 9 shows this three process system. The first process is an A/D converter that produces data consumed by a

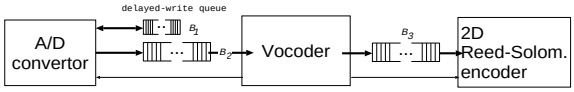

Figure 9: Partial Block diagram of Mobile Terminal Transmitter.

voice coder (Vocoder), a complex DSP designed using the Cathedral-II silicon compiler. In between, two buffers are needed. The data produced by the Vocoder is encoded by a 2-D Reed-Solomon encoder. Here we also find a buffer.

The problem of synthesizing or verifying buffers in this context can be solved by translating the processes and buffers to the state space domain using *counters*, as explained in [9]. However, the vocoder is a complex signal processing algorithm, requiring over 100,000 clock cycles per execution frame. This implies that for a complete pass over the state space of the vocoder itself at least as many iterations are required, making this method prohibitively expensive. Hence, composition and model checking without abstraction is infeasible for this application. However, chains of states are present in the description of the processes that do not act upon the buffers. Therefore we

applied our abstraction technique to obtain an abstract system, upon which we applied a model check using *smv*, which checks on all buffers whether their maximum value is reached and not exceeded. Applying the chain abstraction enabled us to reduce the size of the state space to 1663 timed states, while the model check required 35.6 seconds of DECStation 5000/120 CPU time.

VI Conclusions

The *chain abstraction*, as proposed by this paper, is an approach to the *state explosion problem*, which often occurs in verification of communicating finite state systems. The abstraction is based on representing certain sequences of states more efficiently, preserving the information present in the original system description. Therefore, we defined *timed labeled transition systems (TLTS)*, a model suitable for representing systems containing chains efficiently. Furthermore, we presented a composition procedure for TLTSs. This way, the overall procedure allows us to first abstract systems before composition. The procedure we use now also deals effectively with self-looping wait states.

A user can describe many systems manually using this abstraction, use the composition algorithm and perform model checking on the result. Furthermore, we also have an automatic procedure capable of determining the chains in a system, based on BDD operations.

Case studies show that the abstraction indeed leads to a significant reduction of both size and depth of the state space and to feasibility of the model check for an important class of systems. Here, we mainly focus on systems composed out of subsystems that have large intervals of autonomous computation during which no interaction occurs, separated by relatively sparse communication events.

Summarizing, we can state that our abstraction method offers considerable reduction potential, yet is fully compositional and strongly preserves the formulas in CTL* .

References

[1] R. Alur, C. Courcoubetis, N. Halbwachs, D. Dill, and H. Wong-Toi. Minimization of Timed Transition Systems. In *Proc. CONCUR*, LNCS vol. 630, pages 340–354. Springer-Verlag, 1992.

[2] S. Bensalem, A. Bouajjani, S. Graf, C. Loiseaux, and J. Sifakis. Property Preserving Abstractions for the Verification of Concurrent Systems. In *Formal Methods in Computer Science (subm.)*, volume 17, 1993.

[3] M.C. Browne, E.M. Clarke, and O. Grumberg. Characterizing Finite Kripke Structures in Propositional Temporal Logic. *Theoretical Computer Science*, pages 115–131, 1988.

[4] J.R. Burch, E.M. Clarke, and D.E. Long. Symbolic Model Checking with Partitioned Transition Relations. In *Proc. Int. Conference on VLSI*, August 1991.

[5] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. In *Proc. IEEE Symposium on Logic in Computer Science*, June 1990.

[6] E.M. Clarke, O. Grumberg, and D.E. Long. Model Checking and Abstraction. In *Proc. ACM Symposium on Principles of Programming Languages*, 1991.

[7] O. Coudert and J.C. Madre. A Unified Framework for the Formal Verification of Circuits. In *Proc. IEEE Int. Conference on Computer-Aided Design*, pages 126–129, 1990.

[8] E.A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 996–1071. Elsevier Science Publishers B.V. (North Holland), 1990.

[9] T. Kolks, Bill Lin, and H. De Man. Sizing and Verification of Communication Buffers for Communicating Processes. In *Proc. IEEE Int. Conference on Computer-Aided Design*, 1993.

[10] R.P. Kurshan. Analysis of Discrete Event Coordination. In *Proc. REX Workshop on Stepwise Refinement of Distributed Systems*, LNCS vol. 430. Springer-Verlag, 1989.

[11] K.L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem.* PhD thesis, Carnegie Mellon University, 1992.

[12] F. Somenzi. Verification of Systems Containing Counters. In *Proc. IEEE Int. Conference on Computer-Aided Design*, 1992.

[13] H.J. Touati, H. Savoj, B. Lin, R.K. Brayton, and A. Sangiovanni-Vincentelli. Implicit State Enumeration of Finite State Machines using BDD's. In *Proc. IEEE Int. Conference on Computer-Aided Design*, pages 130–133, 1990.

$Q = Q'_0 \times Q''_0$; $Q_{new} = Q$; $R = \emptyset$; $\tau_q = \emptyset$;
do
{
  $Q_n = \emptyset$; $R_n = \emptyset$;
  forall $q \in Q_{new}$ /* $q = (q', q'') = ((s', t'), (s'', t''))$ */
  {
    $l' = \tau'(s') - t' + 1$;
    $l'' = \tau''(s'') - t'' + 1$;
    $l = \min(l', l'')$;
    $\tau_q = \text{add\_map}(\tau_q, q, l)$;
    forall $s'_n \in S'$, $s''_n \in S''$,
        $(i', o') \in \mathcal{S}(I' \cup O')$,
        $(i'', o'') \in \mathcal{S}(I'' \cup O'')$
    {
      if $l' < l''$
        if $R'(s', (i', o'), s'_n) \wedge (L_O(s'') = o'')$
        {
          $R_n = R_n \cup \{((q', q''), ((i', i''), (o', o'')), ((s'_n, 1), (s'', t'' + l)))\}$;
          $Q_n = Q_n \cup \{((s'_n, 1), (s'', t'' + l))\}$;
        }
      if $l' > l''$
        if $R''(s'', (i'', o''), s''_n) \wedge (L_O(s') = o')$
        {
          $R_n = R_n \cup \{((q', q''), ((i', i''), (o', o'')), ((s', t' + l), (s''_n, 1)))\}$;
          $Q_n = Q_n \cup \{((s', t' + l), (s''_n, 1))\}$;
        }
      if $l' = l''$
        if $R'(s', (i', o'), s'_n) \wedge R''(s'', (i'', o''), s''_n)$
        {
          $R_n = R_n \cup \{((q', q''), ((i', i''), (o', o'')), ((s'_n, 1), (s''_n, 1)))\}$;
          $Q_n = Q_n \cup \{((s'_n, 1), (s''_n, 1))\}$;
        }
    }
  }
  $R = R \cup R_n$;
  $Q_{new} = Q_n \setminus Q$;
  $Q = (Q \cup Q_n)$;
} **while** $Q_{new} \neq \emptyset$;
return (Q, τ_q, R);

Figure 10: Synchronous composition using chain abstraction