Worst-Case Analysis of Discrete Systems Based on Conditional Abstractions

Felice Balarin

Cadence Berkeley Laboratories

Abstract

Recently, a methodology for worst-case analysis of systems with discrete observable signals has been proposed [4]. We extend this methodology to make use of conditional system abstractions that are valid only in some system states. We show that the response-time analysis for single-processor systems is particularly well suited for use of such abstractions. We use an example to demonstrate that significantly better response-time bounds can be obtained using conditional abstractions.

1 Introduction

System verification is hard because system responses need to be checked for all legal behaviors of the environment. Typically, there are infinitely many such behaviors. Even when the problem can be reduced to enumerating finitely many internal system states, their number is usually prohibitive. Using abstractions and implicit state enumeration can simplify the problem, but complete verification is at best at (and often beyond) the limit of existing computers.

An alternative approach is the worst-case analysis, where the system response is analyzed only for the most demanding behaviors of the environment. Worst-case analysis is a well known engineering method, but so far it has been used ad-hoc, with separate techniques for specific system properties.

In [4], a general methodology for worst-case analysis of systems with discrete observable signals was proposed. The methodology can be used to verify different properties of systems, such as power consumption, timing performance, or resource utilization. In addition, an application of the general methodology to timing analysis of embedded systems implemented on a single processor was also proposed. The timing analysis problem considered there is to determine the response time of a system, given processing time requirements of system components, and taking into account that response to some requests may be delayed by responses to other requests. The previous approaches to this problem were either applicable to a very restricted class of systems [7, 2, 3], or they required use of formalisms (e.g. [1, 8]) which cannot be efficiently analyzed [6, 5].

The methodology proposed in [4] is based on a view of the worst-case analysis as an abstraction that maps behaviors to their worst-case representatives. These worst-case representatives are found by analysis of abstractions of systems components. For the analysis to be "worst-case", the component abstractions have to be conservative, i.e. they need to predict a system response that is at least as "bad" as the real response. Component abstractions have to predict system response regardless of the system state. This may force these abstractions to be overly conservative, because more accurate abstractions can often be found for specific system states.

In this paper, we show that the worst-case analysis is still possible even if the requirements on component abstractions are relaxed such that they have to be conservative only in some states of the system. We call such abstractions *conditional abstractions*. We also show that conditional abstractions suffice for the timing analysis proposed in [4], as that analysis naturally focus on certain system states for which a more accurate abstraction can be found.

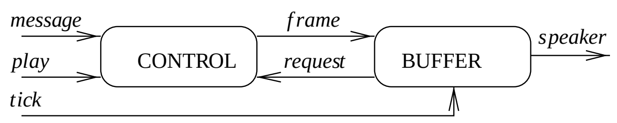
Voice mail pager Throughout this paper we use as an example a voice mail pager shown in Figure 1. To facilitate future references, we briefly explain its behavior. The pager receives messages from the environment. Each *message* consists of up to five frames, and each frame contains fifty samples. The CONTROL module stores messages internally (in variable $frames$) and initiates playing of the most recent message (by generating the $frame$ event) when the user requires so (by generating the $play$ event). The CONTROL module also generates a $frame$ if some are available, and the BUFFER module makes a *request*. The BUFFER module starts playing the message after it receives the initial $frame$ from CONTROL. A message is played by sending to the $speaker$ one sample per every $tick$ of the real-time clock. When there are fewer than 20 samples left to play, the BUFFER sends a *request* for the next frame to the CONTROL.[1]

The rest of this paper is organized as follows. In Section 2 we review the worst-case analysis proposed in [4]. In Section 3 we extend this analysis to use conditional abstractions, and show how this extension improves the timing analysis of the voice-mail pager. We give some final remarks and indications of the future work in Section 4.

2 Worst-case analysis

In this section we briefly review the worst-case analysis proposed in [4]. In that work, a *system* is defined to be a set of executions. An *execution* x of length $len(x)$ is a mapping from the interval $[0, len(x)]$ to some set of *signal values*. Given an execution x and

[1]The pager described here obviously lacks many features to be a successful product. However, it is a very (in fact overly) simplified version of a much more complex realistic design to which we have applied the analysis proposed in this paper.



```
   module CONTROL { frameType frames[5]; integer last := 0;
1    if( present( message )) {
2       frames := value( message );
3       last := size( value( message )); }
4    if( (present( play ) || present( request ) && last > 0 ) {
5       emit frame( frames[ last−− ] ); } }

   module BUFFER { sampleType samples[50]; integer last := 0;
6    if( present( frame )) {
7       samples := value( frame );
8       last := 50; }
9    if( present( tick ) && last > 0)) {
10      emit speaker( samples[ last−− ] );
11      if(last = 20)) {
12         emit request(); } } }
```

Figure 1: Voice mail pager structure and behaviors of components.

real numbers u,t such that $0 \le u \le t \le len(x)$, let $x[u,t]$ denote an execution of length $t-u$ defined by:

$$x[u,t](v) \;=\; x(u+v) \;\;.$$

If, in addition, $u > 0$, we use $x[u,t]^-$ to denote

$$x[u-\varepsilon, t-\varepsilon] \;\;,$$

where $\varepsilon > 0$ is small enough that x is constant in $[u-\varepsilon, u)$ and $[t-\varepsilon, t)$. If x satisfies some mild conditions stated in [4], then such ε always exists.

A *signature* is an abstract representation of executions (or their segments). Formally, a signature σ is a mapping from the set of executions to some set D_σ of *signature values*. Consider, for example, the pager in Figure 1 and assume that the signals are all state variables and communication events. For that example we consider the signature represented by the following vector of variables:

$$(pl, ms, tk, fr, f_1, \ldots, f_5, rq, sp) \;\;.$$

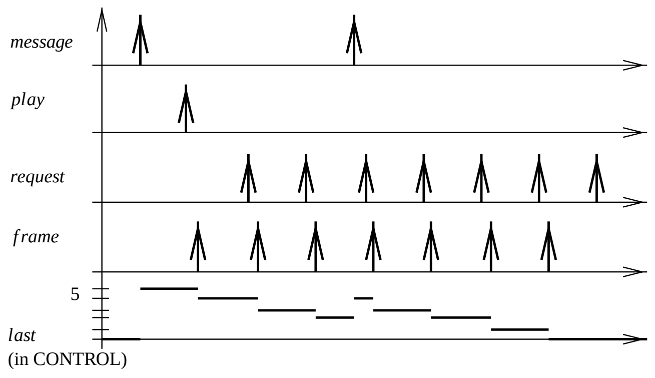
For a given execution, we choose variable pl to hold the number of occurrences of the *play* event in that execution. Variables ms, tk, fr, rq, and sp correspond similarly to events *message*, *tick*, *frame*, *request*, and *speaker* respectively. The rest of the variables abstract the state of the CONTROL module. For some i between 1 and 5, variable f_i holds the number of times line 4 in Figure 1 is executed with variable *last* having value i. Figure 2 shows a component of an execution corresponding to the CONTROL module and its signature.

Signatures provide abstract representation of signals in the system. Next, we define the abstract representation of system components. Given some system S, signature σ and some function $F : D_\sigma \times \mathbb{R}^+ \mapsto D_\sigma$, we say that F is a σ-*abstraction* of S if F is:

- *monotone*: if $T_1 \le T_2$ and $s_1 \le s_2$ then $F(s_1,T_1) \le F(s_2,T_2)$, and
- *future-bounding*: for every execution $x \in S$ and every u,t:

$$\sigma(x[u,t]) \;\le\; F(\sigma(x[u,t]^-),\, t-u) \;\;.$$

Consider, for example, the pager in Figure 1 and the signature σ discussed previously. A σ-abstraction of the pager is shown in



$ms = 2,\ pl = 1,\ rq = 7,\ fr = 7,\ f_1 = 1,\ f_2 = 2, \ldots$

Figure 2: An execution and its signature.

Table 1. It consists of a separate function for each variable in the signature. The variables can be divided into three groups: those generated by the environment, those generated by the CONTROL, and those generated by the BUFFER module. The σ-abstraction of the environmental variables depend only on time, and not on other signature variables. We use a very simple model, where the only constraint is a minimum time between two occurrences of the event. These times are 100, 200, and 500 time units for *tick*, *message* and *play* events, respectively. Therefore, the maximum number of *tick* (*message*, *play*) events in an execution segment of length T is $\lfloor \frac{T}{100} \rfloor + 1$ ($\lfloor \frac{T}{200} \rfloor + 1$, $\lfloor \frac{T}{500} \rfloor + 1$, respectively).

Expressions in Table 1 for σ-abstractions of the CONTROL and BUFFER variables are derived from the code in Figure 1. For example, the BUFFER generates a *request* event 30 *tick* events after it receives a *frame*. Thus, in the interval containing fr *frame* events and tk *tick* events, the BUFFER module can generate at most $\min(fr, \lfloor \frac{tk}{30} \rfloor)$ *request* events, except possibly an extra one at the beginning of the interval. Similarly, the CONTROL module can generate a *frame* event only if line 5 in Figure 1 is executed with value of *last* between 1 and 5. Therefore, fr is bounded by $\sum_{i=1}^{5} f_i$. On the other hand, except at the beginning, the CONTROL module can generate a *frame* event only if it receives a *play* or a *request* event. Therefore, fr is also bounded by $pl + req + 1$. Other bounds in Table 1 are derived by similar reasoning.

The problem of the worst-case analysis is the following:

> For a given interval length T, find a signature value s such that for every execution $x \in S$, and every $t \in [0, len(x) - T]$:
>
> $$\sigma(x[t,t+T]) \;\le\; s \;\;. \qquad (1)$$

In other words, s must be worse than the signature of any execution segment of length T. Such information can help answer many important questions in the design process. For example, if the signature contains information about bus requests, then the worst-case analysis indicates required average bus bandwidth for any period of time of length T. Similarly, if the signature contains information about energy required for an execution, then the worst-case analysis gives a bound the average power for any interval of time of length T. The worst-case analysis can also be used to analyze timing performance of the system, as shown in [4]. It has also been shown in [4] that a signature s satisfying (1) can be found for a given T and a given σ-abstraction.

To apply the worst-case analysis to timing analysis of a software system, the notions of a *busy period* and a *workload function* are introduced in [4]. A busy period is any interval of time in which

Table 1: σ-abstraction of the pager in Figure 1.

environment	CONTROL	BUFFER
$F_{pl} = \lfloor \frac{T}{500} \rfloor + 1$	$F_{fr} = \min(pl + rq + 1, \sum_{i=1}^{5} f_i)$	$F_{rq} = \min(fr, \lfloor \frac{tk}{30} \rfloor) + 1$
$F_{ms} = \lfloor \frac{T}{200} \rfloor + 1$	$F_{f_5} = \min(pl + rq, ms) + 1$	$F_{sp} = \min(tk + 1, 50 * (fr + 1))$
$F_{tk} = \lfloor \frac{T}{100} \rfloor + 1$	$F_{f_i} = \min(pl + rq, f_{i+1}) + 1$ for $i = 1, \ldots, 4$	

the processor is continuously not idle. A busy period is said to be *initialized* if it begins at time 0, or the processor was idle just prior to its beginning.

A workload function indicates how much processor time a system needs to process some input (as a function of that input's signature). Formally, for a given system S and signature σ, a workload function R is some mapping from D_σ to positive real numbers, such that it is:

- *monotone*: if $s_1 \leq s_2$ then $R(s_1) \leq R(s_2)$, and
- *workload-bounding* for every execution x, and every initialized busy period $[u,t]$ in x:

$$R(\sigma(x[u,t])) > t - u \ .$$

The workload-bounding property ensures that approximate processing time requirements given by the workload function is a strict upper bound on the actual processing time requirements which determines the length of a busy period.

For example, assume that the execution time of each executable line in Figure 1 is 10 time unit. Also, assume the signature discussed previously. Then, a workload function for the pager might be:

$$\begin{aligned} R &= 20 * (pl + rq + ms) + 20 * ms + 10 * fr + \\ &\quad 20 * (fr + tk) + 20 * fr + 20 * sp + 10 * rq \ . \end{aligned} \qquad (2)$$

The first line in (2) corresponds to the CONTROL module, while the second line corresponds to the BUFFER module. The workload function (2) is valid under the following conditions:

- *At the beginning of an initialized busy period, modules start executing from the entry point.* For example, the first time the CONTROL module gets the processor in any initialized busy period, it should be executing line 1 in Figure 1. This condition is more or less equivalent to the requirement that the processor cannot be idle if there are unfulfilled processing requirement.
- *A module is not executed unless there are some fresh events at its inputs.* For example, the condition implies that the CONTROL module is executed at most $pl + rq + ms$ times. Each execution will require at least 20 time units for two *if* statements, hence the term $20 * (pl + rq + ms)$ in (2).

The term $20 * ms$ in (2) is due to lines 2 and 3 which will be executed only if a new *message* is received. Similarly, the number of executions of line 5 is the same the number of generated $frame$ events (hence the term $10 * fr$). The terms in (2) corresponding to the BUFFER module were obtained by a similar analysis.

It has been shown in [4] that an upper bound T on the length of busy periods of some system S can be found by solving the system of equations:

$$\begin{aligned} s &= F(s,T) \\ T &= R(s) \ , \end{aligned}$$

where F and R are a σ-abstraction and a workload function of S.

3 Conditional abstractions

The future bounding property of σ-abstractions is quite strong: it requires $F(s,T)$ to be a conservative estimate of the system response to *any* execution segment of length T with signature s. In particular, no assumptions can be made about the state of the system at the beginning of the segment. Therefore, a σ-abstractions must reflect the worst case among *all* the (reachable) states. Usually, a tighter bound can be achieved for a specific state. We will show that in some cases these state-specific bounds can still be used to compute the worst overall signature. But first, we formalize the notion of the state-specific abstraction.

Given some system S, signature σ, subset Q of system states, and some function $F : D_\sigma \times \mathbb{R}^+ \mapsto D_\sigma$, we say that F is a (σ, Q)-*abstraction* (or *conditional abstraction*) of S if F is:

- *monotone*: if $T_1 \leq T_2$ and $s_1 \leq s_2$ then $F(s_1,T_1) \leq F(s_2,T_2)$, and
- *future-bounding in Q*: for every execution $x \in S$ and every u,t such that the state of S after the execution $x[0,u]$ is an element of Q:

$$\sigma(x[u,t]) \ \leq \ F(\sigma(x[u,t]^-), t-u) \ .$$

In other words, a (σ, Q)-abstraction must be a conservative predictor of the future only if the system is in some state in Q.

Consider, for example, the σ-abstraction shown in Table 1. If we restrict out intention to the set of states Q where both CONTROL and BUFFER are terminated (i.e. their next executions would start from lines 1 and 6 in Figure 1), we can get a tighter conditional abstractions shown in Table 2. For example, we have previously argued that the number of $frame$ events the CONTROL module can generate is bounded by $pl + req + 1$, because it can generate a $frame$ event only if it receives a $play$ or a $request$ event, plus one extra time at the beginning of the segment, if it is about to execute line 5. Because the conditional abstractions in Table 2 needs to be conservative only if CONTROL is about to execute line 1, we may disregard that initial $frame$ generation, and strengthen the bound to $pl + req$. Table 2 shows similar strengthening for other components as well. This relatively small change in abstraction may have a dramatic impact on the results of the worst case analysis.

Conditional abstractions can be used for the worst case analysis, as stated by the following:

Table 2: Conditional abstraction of the pager in Figure 1.

environment	CONTROL	BUFFER
$F_{pl} = \lfloor \frac{T}{500} \rfloor + 1$	$F_{fr} = \min(pl + rq, \sum_{i=1}^{5} f_i)$	$F_{rq} = \min(fr, \lfloor \frac{tk}{30} \rfloor + 1)$
$F_{ms} = \lfloor \frac{T}{200} \rfloor + 1$	$F_{f_5} = \min(pl + rq, ms)$	$F_{sp} = \min(tk, 50 * fr)$
$F_{tk} = \lfloor \frac{T}{100} \rfloor + 1$	$F_{f_i} = \min(pl + rq, f_{i+1})$ for $i = 1, \ldots, 4$	

Theorem 1 *Let σ, Q, and F, be a signature, a subset of states, and a (σ, Q)-abstraction of some system S. If $x \in S$, $s \in D_\sigma$, and $T \geq 0$ are such that:*

1. $\sigma(x[0,0]) \leq s$,
2. $s = F(s,T)$,
3. *for all $t \in [0, len(x) - T]$ there exists $u \in [t, t+T]$ such that the state of S after execution $x[0,u]$ is in Q,*

then:

$$\sigma(x[t, t+T]) \leq s$$

for all $t \in [0, len(x) - T]$ such that the state of S after execution $x[0,t]$ is in Q.

The proof of Theorem 1 proceeds similarly to the proof of Theorem 1 in [4], which states an analogous result for σ-abstractions (rather than (σ, Q)-abstractions). The proof in [4] is by induction over the time points in which some events occur. The proof of Theorem 1 is also by induction, but it uses only time points when a system is in some state in Q. Condition 3 in Theorem 1 ensures that the induction goes through, because it requires that there is always the next Q state within time distance T.

Busy-period analysis proposed in [4] is particularly well suited for conditional abstractions. Assume that we can characterize states in which the system can be at the beginning of an initialized busy period. Note that these are also the states at the end of a busy period, because the system is idle between the end of one busy period and the beginning of the next one. Let Q denote that set of states, and let F be a conditional abstraction with respect to that set of states. If we can use F to find T such that the workload generated in in initial T time units of some initialized busy period is less than T, then that busy period must end within T and so there must exist another Q point in the interval of length T. Thus, we may apply Theorem 1. This reasoning is summarized in the following:

Theorem 2 *For a given system S let:*

- *σ be a signature of S,*
- *Q be a subset of states of S that contains all states in which S can be at the beginning of some initialized busy period,*
- *F be a (σ, Q)-abstraction of S,*
- *R be a workload function of S.*

If $s \in D_\sigma$, and $T \geq 0$ are such that:

1. $s = F(s,T)$,
2. $T = R(s)$,
3. *$\sigma(x[0,0]) \leq s$ for all executions $x \in S$,*

then T is an upper bound on the length of busy periods in all executions $x \in S$.

Theorem 2 is an improvement over the busy-period analysis from [4], because it allows the use of potentially tighter conditional abstractions instead of looser and more general σ-abstraction. For example, if we assume that at the beginning of an initialized busy period both CONTROL and BUFFER modules are terminated, then we can use the abstraction in Table 2 to bound the busy period. This is a very reasonable assumption, as most operating systems would not allow an idle period as long as there are processes that are not terminated. This change results in a significant improvement: using the abstraction in Table 1 (as required in [4]) gives the bound of 670 time units on the length of a busy period, while using the abstraction in Table 2 (as Theorem 2 allows) tightens that bound to 370 time units.

4 Conclusions

We have proposed an improvement on the worst-case analysis proposed in [4]. The improvement recognizes and exploits the fact that system abstractions need to be valid only in certain system states. We have showed that the response time analysis is particularly well suited for use of such conditional abstractions.

In general, the use of the approach presented here will be effective if the user can recognize a set of system states such that:

- system behavior can be predicted much more accurately if the system is in one of those, rather than in some arbitrary state,
- the system visits that set of state frequently (more precisely, at least once for every interval of interest).

In the future we plan to investigate whether such sets of states can be recognized to improve analysis of other system properties such as power consumption and resource utilization.

References

[1] Rajeev Alur and David L. Dill. Automata for modelling real-time systems. In M.S. Paterson, editor, *ICALP'90 Automata, languages, and programming: 17th international colloquium.* Springer-Verlag, 1990. LNCS vol. 443.

[2] Neil C. Audsley, Alan Burns, M. Richardson, Ken W. Tindell, and Andy J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, pages 284–292, September 1993.

[3] Felice Balarin. Priority assignment for embedded reactive real-time systems. In Frank Mueller and Azer Bestavros, editors, *Languages, Compilers, and Tools for Embedded Systems, ACM*

SIGPLAN Workshop LCTS'98 Montreal, Canada, June 1998, pages 146–155. Springer, 1998.

[4] Felice Balarin. Worst-case analysis of discrete systems. Technical report, Cadence Berkeley Laboratories, May 1999. also submitted to ICCAD-99.

[5] Felice Balarin, Karl Petty, Alberto L. Sangiovanni-Vincentelli, and Pravin Varaiya. Formal verification of the PATHO real-time operating system. In *Proceedings of 33rd Conference on Decision and Control, CDC'94*, December 1994.

[6] Felice Balarin and Alberto Sangiovanni-Vincentelli. Schedule validation for embedded reactive real-time systems. In *Proceedings of the 34th ACM/IEEE Design Automation Conference*, June 1997.

[7] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-realtime environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.

[8] Jennifer McManis and Pravin Varaiya. Suspension automata: A decidable class of hybrid automata. In David L. Dill, editor, *Proceedings of Computer Aided Verification: 6th International Conference, CAV'94, Stanford, CA, June 1994*, pages 105–117. Springer-Verlag, 1994. LNCS vol. 818.