AutoFlows: Inferring Message Flows From System Communication Traces

Authors
Department
Affiliation
City, Country
emails

***Abstract*—This paper proposes a novel method for automatically inferring message flow specifications from the communication traces of a system-on-chip (SoC) design that captures messages exchanged among the components during a system execution. The inferred message flows characterize the communication and coordination of components in a system design for realizing various system functions, and they are essential for SoC validation and debugging. The proposed method relieves the burden of manual development and maintenance of such specifications on human designers. Our method also develops a new accuracy metric, *acceptance ratio*, to evaluate the quality of the mined specifications instead of the specification size often used in the previous work, enabling more accurate specifications to be mined. The effectiveness of the proposed method is evaluated on both synthetic traces and traces generated from executing several system models in GEM5. In both cases, the proposed method achieves superior accuracies compared to a previous approach.**

***Index Terms*—system-on-chip, specification mining, model inference, system-level models,**

I. Introduction

Modern System-on-Chip (SoC) designs consist of numerous functional blocks, each handling a distinct task and communicating through advanced protocols to enhance system functionality. These blocks operate concurrently, leading to simultaneous or interleaved system transactions. However, this concurrency and complex protocol interaction often result in runtime errors during debugging phases. An example of a standard SoC design, including components like CPUs, caches, and Network-on-Chip interconnects, is depicted in Fig. 1.

Having a precise, efficient, and comprehensive specification model is crucial for the validation, verification, and debugging of system designs. Creating such a model for a system design can lead to the ability to simulate the system behavior and characteristics to foresee any design defects prior to fabricating the design. However, such models in practice are usually incomplete, inconsistent, ambiguous, or may even include errors. Therefore, the lack of such comprehensive models prevents the systematic analysis and validation of complex system designs.

To address the above challenges, this paper introduces a new method for automatically inferring accurate message flow specifications from system communication traces. The proposed method constructs a causality graph from input traces to understand message relationships. Then, it selects a subset of message sequences in the causality graph as a potential specification model, which is evaluated on the input traces.

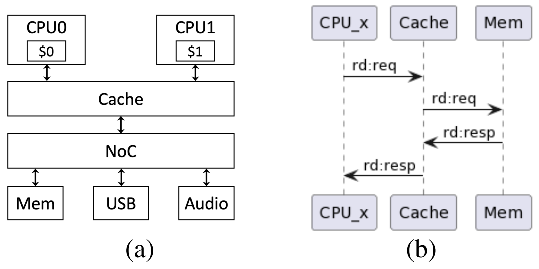


Fig. 1: (a) A simplified SoC architecture example, (b) Message sequence diagram for the CPU downstream read flows. This diagram is parameterized with x being 0 or 1. Only communications involving CPU, Cache, and Mem blocks are included for a clear illustration.

If the chosen model's accuracy is unsatisfactory, it will be refined and re-evaluated. This cycle continues until the model achieves satisfactory accuracy. This paper makes the following contributions.

- A novel mining method that directly extracts specification flows from input traces to represent advanced system-level protocols in complex system designs.
- A new evaluation method that is the first to measure the accuracy of the inferred models instead of using the model sizes as the quality measurement of the results.
- A new refinement algorithm that can improve the accuracy of the inferred models iteratively until a certain level of accuracy is achieved.

Extensive experimental evaluation shows that this method is able to infer models with very high accuracies for diverse traces including synthetic traces and traces generated by executing more realistic system models developed in GEM5 [1].

This paper is organized as follows. Section II reviews some of the previous related works in specification mining and addresses the issues of each one. In section III necessary backgrounds are provided. Sections IV and V represent the proposed method and experimental results. And finally, section VI provides the conclusion and future works.

II. Related Works

Specification mining aims to derive patterns from various artifacts. The *Synoptic* model-based approach extracts invariants from sequential execution logs, recording concurrency partially, and generates a Finite State Machine (FSM) consistent with these invariants [2]. Another tool, *Perracotta*, analyzes logs of software execution and mines temporal API rules. It employs a chaining technique to identify long sequential patterns [3]. The study known as *BaySpec* extracts LTL (Linear Temporal Logic) formulas from Bayesian networks that are trained using software traces. This approach requires traces to be clearly partitioned with respect to different functions [4]. To extract information from hardware traces, the techniques described in [5]–[9] focus on extracting assertions. These approaches mine assertions either from gate-level representations [5] or RTL (Register-Transfer Level) models [6]–[10]. The research discussed in [11] outlines a method for mining assertions by utilizing episode mining from simulation traces of transaction-level models.

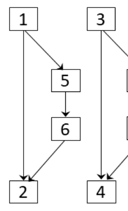
Many of these approaches have limitations in detecting longer patterns, making them unable to identify intricate communication patterns that involve multiple components. In contrast, a recent study in [12] addresses a comparable problem, but it focuses solely on mining sequential patterns. The study in [13] presents a method for validating SoC security properties, focusing on communication fabrics. It involves constructing Control Flow Graphs (CFGs) for each Intellectual Property (IP) component and analyzing the interconnections between these CFGs to assess security risks. The research underscores the importance of system-level interactions and identifies communication fabrics as key areas for security validation. Understanding the communication model at the fabric level is crucial for various validation processes.

Our research focuses on model synthesis methods, aiming to create models from system execution traces that faithfully represent input traces. Heule et al. [14] improved the deterministic finite automata inference, initially proposed by Lang [15], by treating it as a graph coloring problem and using a Boolean satisfiability solver. *Trace2Model*, a recent development *Trace2Model*, learns non-deterministic finite automata models from software traces using C bounded model checking. Similar research is also noted in [16]. These studies together explore diverse model synthesis techniques, emphasizing accurate system behavior representation from execution traces

The aforementioned methods fail to account for the concurrent nature of communication traces in SoC designs. Instead, they heavily rely on identifying temporal dependencies from the traces to infer models. However, it is important to note that temporal dependencies do not always align with the actual dependencies that our research seeks to uncover. The *model synthesis* method in [17], addresses the problem of the previous approaches and uses a constraint-solving approach to infer finite automata models from SoC communication traces. However, it aims to infer models of minimized sizes, not considering the accuracy of the inferred models with respect to the input traces. This approach, assuming that each message leads to at most one subsequent message, is restrictive for more complex communication scenarios.

III. Background

This section provides the necessary background for the proposed method similar to that in [17]. Message flows are a formalism to represent communication protocols across



```
1 (cpu0:cache:rd:req)
2 (cache:cpu0:rd:resp)
3 (cpu1:cache:rd:req)
4 (cache:cpu1:rd:resp)
5 (cache:mem:rd:req)
6 (mem:cache:rd:resp)
```

(a) (b)

```
Initial messages = {1,3}
Terminal messages = {2,4}
```

(c)

Fig. 2: (a) Graph representation of CPU downstream flows given in Fig. 1(b) where the nodes are labeled with messages as defined given (b). (c) gives the flows' initial and terminal messages.

multiple components in a system. Fig. 1(b) depicts a message flow example for a downstream memory read, focusing on interactions between CPU, Cache, and memory while excluding cache coherence. This paper represents these flows as directed acyclic graphs, as seen in Fig. 2. Each message, as exemplified in Fig. 2(b), is structured as a quadruple $(\texttt{src} : \texttt{dest} : \texttt{cmd} : \texttt{type})$, indicating the message's originating and receiving components, the operation at the destination, and whether it is a request or a response respectively. In Fig.2(b), the message $(\texttt{cpu0} : \texttt{cache} : \texttt{rd} : \texttt{req})$ is a read request from cpu0 to cache. Message flows begin with an initial message to start a flow instance and end with terminal messages signaling completion, encompassing various sequences for different execution scenarios. Fig.2(a) shows two execution methods for each CPU's memory read flow. The flow from cpu0 includes sequences (1, 2) for a cache hit and (1, 5, 6, 2) for a cache miss.

Message flows can execute concurrently via *interleaving*, creating traces $\rho = (m_0, m_1, \ldots, m_n)$, with each m_i being a message. In trace ρ, if $i < j$, then the ordering is denoted as $m_i <_\rho m_j$. For example, in Fig. 2, if CPU0 and CPU1 run their memory read flows three and two times respectively, one possible execution trace is

$$(3, 4, 1, 1, 5, 6, 2, 5, 6, 2, 1, 2, 3, 4). \tag{1}$$

Fig. 2(a) shows that a message flow represents sequences of causality relations among messages, a subset of the broader concept of *structural causality*. This is based on the idea that each message in a system trace is a component's output responding to a preceding input message.

Definition 3.1: Message m_j is causal to m_i, denoted as $causal(m_i, m_j)$, if $m_i.\texttt{dest} = m_j.\texttt{src}$.

The causality in flow specifications described above is *structural*, distinct from functional or true causality, which includes but is not limited to structural causality. Message flow mining aims to discern all true causality relations from structural ones using system execution traces.

IV. Method

Algorithm 1 outlines the method used in this paper, requiring a set of traces, a message definition file, and an accuracy

```
Algorithm 1: Proposed Method
  input : A set of traces $T$, Message definition File $F$,
          Accuracy threshold $\mathcal{A}$, Pruning threshold $\theta$
  output: Message flow model mined from $T$
1 $CG = \texttt{ConstructCausalityGraph}(T, F)$;
2 $Model, PG = \texttt{ModelSelection}(CG, \theta)$;
3 $AR, UM, UE = \texttt{ModelEvaluation}(T, Model)$;
4 if $AR \geq \mathcal{A}$ then
5     return $Model$;
6 else
7     return
        $\texttt{ModelRefinement}(Model, T, UM, UE, PG, \mathcal{A})$;
```

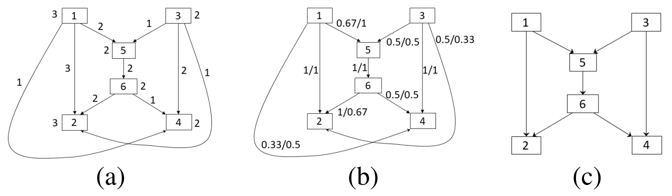


Fig. 3: Causality graph from trace (1): (a) nodes and edges with support information, (b) edges marked with forward/backward confidences, (c) pruned version of the graph.

threshold. It generates message flows from the traces that meet the accuracy threshold. It involves 4 main steps: (1) constructing a causality graph from the traces, (2) creating a message flow model by selecting sequences from the graph, (3) evaluating the model's accuracy against the traces, and (4) refining the model if its accuracy is inadequate. The following subsections give detailed explanations for each of these steps.

A. Causality graph Construction

The key of message flow mining is to discern true causalities from structural ones among messages in traces. The initial step in our method is building causality graphs to encapsulate all structural causalities between messages.

1) Construction: To build causality graphs, the first step is extracting unique messages from input traces T, where a unique message differs in at least one attribute from those already collected. We assume that users identify the initial and terminal messages of interest. Let M be the set of all unique messages from T, with initial and terminal messages labeled as i and t, respectively, where $i \subset M$ and $t \subset M$.

A causality graph is a directed acyclic graph (DAG) with multiple roots and terminals. Each node is labeled with a unique message from set M, roots with initial messages from i, and terminals with terminal messages from t. Edges denote structural causality relations. For each initial message m_i in i, a causality graph is built by adding a root for m_i, followed by nodes for all messages m where $causal(m_i, m)$ is true, proceeding recursively to terminal messages. Edges forming cycles are omitted in this process.

2) Trace Statistics Collection: Once the causality graph is built, the next step is to identify message sequences from initial to terminal nodes. In this paper, the terms 'message sequence' and 'path' are used interchangeably. The causality graph, based on input traces, contains all potential paths from initial to terminal nodes. Finding all paths linearly is impractical due to their vast number. To address this, we use trace statistics derived from analyzing the input trace. Definitions 4.1 and 4.2 specify node support and edge support respectively.

Definition 4.1: $NodeSupport(m)$ is the count of instances of a message m in the input trace.

Definition 4.2: $EdgeSupport(h, t)$ for an edge from nodes h to t, labeled with messages m_h and m_t, is the count of m_t instances each paired with an m_h instance, where $m_h <_\rho m_t$ in the trace and m_h is not matched with other m_t instances.

After determining node and edge supports, edge confidence measures are defined below. These measures are used to show the causality strength between messages.

Definition 4.3: Forward-confidence of edge $h \to t$:

$$ForwardConfidence(h, t) = \frac{EdgeSupport(h, t)}{NodeSupport(h)} \quad (2)$$

Definition 4.4: Backward-confidence of edge $h \to t$:

$$BackrwardConfidence(h, t) = \frac{EdgeSupport(h, t)}{NodeSupport(t)} \quad (3)$$

Given a set of traces T, the final forward and backward confidences for T are the averages of forward and backward confidences computed for each individual trace $\rho \in T$.

Fig. 3 displays the causality graph for trace (1), where Fig. 3(a) shows node and edge supports, and Fig. 3(b) illustrates forward and backward confidences. This information aids in model selection in the next step.

B. Model Selection

In this work, a *model* comprises message sequences as paths in a causality graph. We first prune the graph by excluding edges with low confidence, then create a base model by selecting paths in the pruned graph that meet specific criteria.

1) Pruning the causality graph: Once the causality graph and trace statistics are gathered, we apply a user-defined threshold to the edge confidence data. Edges with higher forward or backward confidence are more likely valid, so those with lower confidence are removed, resulting in a pruned causality graph. Increasing this threshold further reduces the graph's size in terms of edges and paths.

For a better understanding consider the causality graph in Fig. 3. From trace (1), all edges except $(1, 4)$ and $(3, 2)$ have confidences above 0.4. Setting a 0.4 threshold, edges with lower confidences are removed. Edges like $(3, 5)$ and $(6, 4)$ with low confidence might be excluded from the final model. The model refinement process using this data is explained in detail later. The pruned graph is shown in Fig. 3(c).

It is practical to focus on longer message sequences because they illustrate more complex communication scenarios that involve multiple components, making them significant for validation purposes. However, causality graphs might include excessively long message sequences that could be invalid. By managing the length of these sequences, it becomes possible to filter out numerous invalid sequences, thereby simplifying the process of model selection.

Algorithm 2: ModelSelection

```
input  : Causality Graph $CG$, Pruning threshold $\theta$
output: Model $\mathcal{M}$, Pruned Causality Graph $PG$
1  $PG = PruneCausalityGraph(CG, \theta)$;
2  $\mathcal{M} = \emptyset$;
3  Let $UM$ be all messages in $PG$;
4  while $UM \neq \emptyset$ do
5      foreach $initialMessage$ in $PG$ do
6          Select the longest path $LP$ starting from
             $initialMessage$ that covers most messages in
             $UM$;
7          Add $LP$ to $\mathcal{M}$;
8          Let $CM$ be the messages covered by $\mathcal{M}$;
9          $UM = UM - CM$;
10 return $\mathcal{M}$, $PG$;
```

2) Selecting the base model: Model selection aims to find a minimal model meeting the *coverage requirement*, ensuring every message in the causality graph is represented. Following Occam's razor [18], simpler models offer better generalizability and are easier for users to understand. Hence, the objective is to choose the smallest model satisfying the coverage requirement. In this work, a model's size is determined by the number of message sequences from the pruned causality graph in the inferred model. To generate the base model, we categorize message sequences in the pruned causality graph by their initial message and length. For each initial message, we add the longest path containing at least one uncovered message to the base model and remove it from the available sequences. Longer sequences are preferred as they represent more complex communication scenarios. This process is repeated until all messages are covered. Details of this path selection algorithm are in Algorithm 2.

C. Model Evaluation

Previous research often evaluated model quality by size [17], but this does not reliably indicate their effectiveness in explaining input traces. To tackle this, our paper introduces the *acceptance ratio*, a fractional value measuring the proportion of messages in trace set T accepted by model M relative to the total length of traces in T. This ratio serves as an indicator of a model's accuracy.

The proposed evaluation algorithm transforms model paths into a finite state automata (FSA), $\mathcal{M} = \{Q, q_0, \Sigma, F, \Delta\}$, consisting of a finite set of states Q with $q0$ as the initial state, a set of symbols Σ, a subset F of Q for accepting states, and a transition function Δ mapping from $Q \times \Sigma$ to Q.

The evaluation iterates over all messages in the input trace. It starts a new FSA flow instance with each initial message. For non-initial messages, it attempts to fit them into active flow instances; if unsuccessful, they are added to unaccepted messages. Algorithm 3 represents the proposed evaluation method. Unused edges during evaluation and the count of unaccepted messages are returned to be used in the model refinement phase.

Algorithm 3: ModelEvaluation

```
input  : A set of trace $T$, Generated model $\mathcal{M}$
output: Acceptance Ratio $AR$, Unused Messages
          $UM$, Unused Edges $UE$
1  Convert $\mathcal{M}$ to an FSA $M = \{Q, q_0, \Sigma, F, \Delta\}$;
2  $UnAccepted = \emptyset$;
3  $SumAR = 0$;
4  $X = \emptyset$;
5  $UnusedEdges = \Delta$;
6  foreach $\rho$ in $T$ do
7      $Accepted = \emptyset$;
8      foreach $i \in [0, |\rho|]$ do
9          $m \leftarrow \rho[i]$;
10         if $m$ is a start message then
11             find $q_1$ s.t. $\Delta(q_0, m, q_1)$ holds;
12             Create and add a model instance $(M, q_1)$ to
                 $X$;
13             Add $m$ to $Accepted$;
14         else if $\exists (M, q) \in X,\ \Delta(q, m, q')$ holds then
15             $X \leftarrow X \cup \{(M, q')\} - (M, q)$;
16             Add $m$ to $Accepted$;
17             Remove $\Delta$ from $UnusedEdges$ if present;
18         else
19             Add $m$ to $UnAccepted$
20     $SumAR = SumAR + |Accepted|/|\rho|$;
21 return $AR = SumAR/|T|$, $UnAccepted$,
     $UnusedEdges$;
```

D. Model Refinement

In this phase, our goal is to improve the base model to attain a higher acceptance ratio. We assign scores to paths in the pruned causality graph, and based on these scores and evaluation results, we modify the model by adding or removing paths to increase the acceptance ratio. Subsequent subsections detail this model refinement process.

1) Path scoring: As previously noted, each edge in the causality graph has trace statistics like forward and backward confidences. Paths in the model, composed of multiple edges, are assigned *forward score* and *backward score*, calculated as the average of confidences of their edges. Higher scores indicate a greater likelihood of a path being valid. The final score of each path is computed using equation (4).

$$PathScore = \frac{(ForwardScore + BackwardScore)}{PathLength} \quad (4)$$

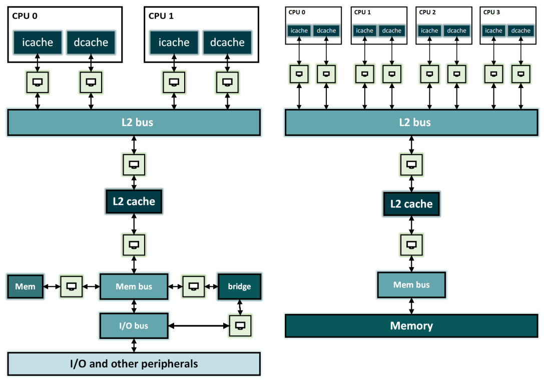
2) Refinement: The refinement algorithm improves the model using evaluation outputs and path scores. If the acceptance ratio meets the accuracy threshold, the model is finalized. Otherwise, it removes unused paths, adds paths with unaccepted messages and high path scores, and re-evaluates. The process repeats until the acceptance ratio meets the threshold or all paths are tested, as detailed in Algorithm 4.

V. Experimental Results

The proposed mining method is tested on synthetic traces and traces from system models in the GEM5 simulator [1]. This section covers the experimental results and discussions.

Algorithm 4: ModelRefinement

```
input : Model $\mathcal{M}$, A set of trace $T$, Unaccepted
        Messages $UM$, Unaccepted Edges $UE$,
        Pruned Causality Graph $PG$, Accuracy
        threshold $\mathcal{A}$
output: Refined Model $\mathcal{M}$
1  $SortedPaths = ComputeScores(PG)$;
2  $AR = 0$;
3  repeat
4      foreach $edge$ in $UE$ do
5          remove $edge$ from $\mathcal{M}$;
6      $maxUnaccepted = max(UM)$;
7      foreach $path$ in $SortedPaths$ do
8          if $maxUnaccepted$ in $path$ then
9              Add $path$ to $\mathcal{M}$;
10             Remove $path$ from $SortedPaths$;
11             break;
12     $AR, UM, UE = \texttt{ModelEvaluation}(T, \mathcal{M})$;
13 until $AR \geq \mathcal{A}$ or $SortedPaths = \emptyset$;
14 return $\mathcal{M}$
```



(a) (b)

Fig. 4: (a) GEM5 Full-System (FS) design. (b) GEM5 System Emulation (SE) design. ▣: Communication Monitor

A. Synthetic Traces

In preliminary tests, our method is applied to synthetic traces generated from 10 message flows, mimicking real SoC designs, involving memory operations and accesses by CPUs, caches, and peripherals. We created three trace sets: `small-20`, `large-10`, and `large-20`. `small` and `large` refer to the number of message flows, and the numbers indicate flow execution instances. `small-20` includes only CPU-initiated flows (4 out of 10) executed 20 times, while `large-10` and `large-20` encompass all flows, executed 10 and 20 times respectively.

B. GEM5 Traces

For a comprehensive evaluation, we applied our method to GEM5-generated traces. The GEM5 simulator operates in Full System (FS) mode as in Fig. 4(a) or System-call Emulation (SE) as in Fig. 4(b). SE mode simulates program-executed system calls and requires static thread-to-core mapping, limiting multi-threaded simulations. FS mode, simulating an operating environment with interrupts, I/O devices, etc., offers more accurate simulations and diverse workloads. The SE setup includes four x86 CPUs with private L1 caches, a shared L2 cache, and 2GB DDR4 memory. The FS setup has two CPUs, DDR3 memory, and additional components like I/O devices. SE simulation produces trace `threads` from a multithreaded program and trace `snoop` from Peterson's algorithm. The FS trace comes from Ubuntu 18 Linux OS boot-up.

TABLE I: Synthetic Results

	Large-20 (10900)			Large-10 (4360)			Small-20 (3680)		
	RT	Size	Ratio	RT	Size	Ratio	RT	Size	Ratio
Proposed Method	224	264	99.52%	36	195	99.47%	4	96	99.37%
Model Synthesis [17]	20	103	57.12%	21	107	63.69%	21	61	69.97%

C. Comparison

We implemented our method in Python and compared it with the Model Synthesis (MS) method [17]. To ensure fairness, we used our evaluation method on both MS results and ours for accuracy comparison and compared model sizes based on the number of transitions $|\Delta|$ in the FSAs. Results for synthetic and GEM5 traces are shown in Tables I and II, with trace names followed by their message counts.

In our method, the pruning threshold considers forward and backward confidences and the model size. We chose a 0.9 threshold for synthetic and 0.5 for GEM5 traces based on our observations. No accuracy threshold was set for synthetic and GEM5 snoop traces, instead testing all sequences in the pruned graph. For the larger GEM5 threads and full system traces, a 98% accuracy threshold was used, based on GEM5 snoop data. The method significantly improved acceptance ratios, by 29.4% to 42.4% for synthetic traces and 2.67% to 33.28% for GEM5 traces. Notably, model synthesis failed to generate a model for GEM5 threads and full-system traces without code modification and constraint relaxation.

Fig. 5(a) shows a message sequence with length 10 from the GEM5 threads trace, illustrating a write operation during a cache miss, as inferred by our method. Fig. 5(b) presents a GEM5-documented write miss sequence, more abstract than Fig. 5(a) and lacking the memory to cache response. This comparison highlights how detailed mined sequences offer a deeper understanding of system designs. Additionally, Fig. 5(c) shows sequences for memory read and cache clean eviction scenarios, with the latter not included in GEM5 documentation but triggered in scenarios like reads and writes.

Although the models from our method are larger than those from model synthesis, they extract longer message sequences. Fig. 6 compares counts of different length message sequences mined by both methods in GEM5 traces, showcasing our model's ability to reveal complex message sequences. The proposed method yields significantly more instances of 10-length message sequences than model synthesis. However, for shorter sequences, counts are sometimes lower with our method, implying these cases are encompassed by longer sequences, reflecting our method's higher accuracy compared to model synthesis.

D. Discussions

There is a trade-off between model accuracy and size, and between accuracy and refinement runtime. Lowering the accuracy threshold speeds up refinement and reduces model

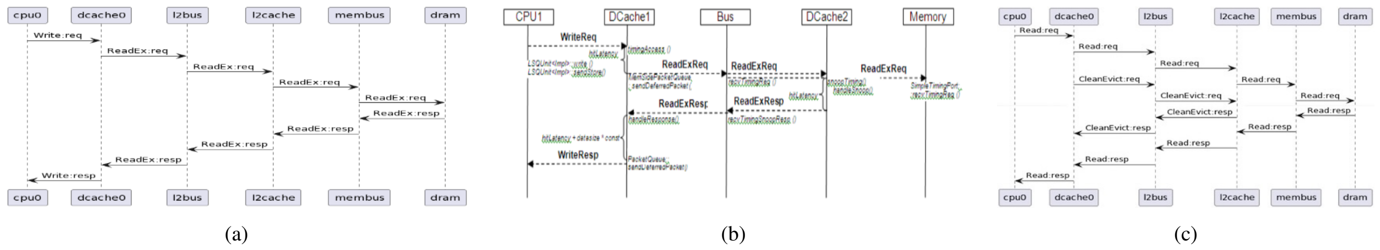


Fig. 5: (a) A flow mined from the GEM5 threads trace. (b) GEM5 documentation's memory write miss sequence. (c) Two flows mined from the GEM5 threads trace depicting a clean eviction scenario, *not in GEM5 documentation*.

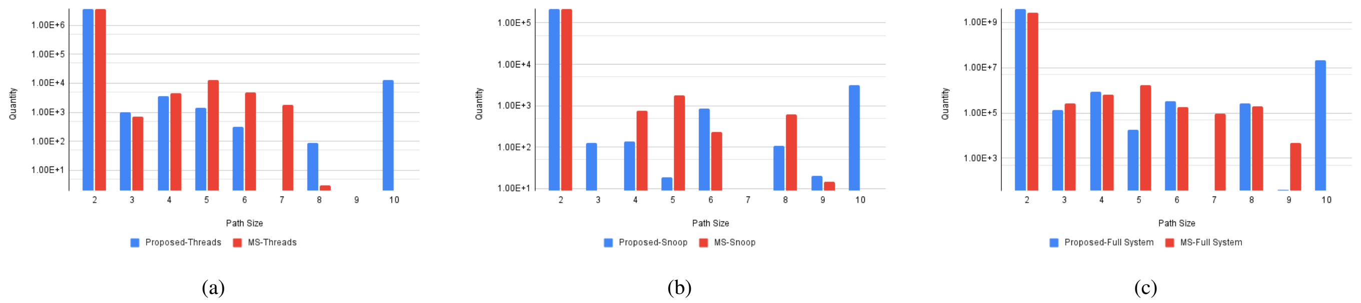


Fig. 6: Counts of instances of the mined message sequences of different lengths using the proposed method and model synthesis method that are found in the GEM5 (a) `threads`, (b) `snoop`, and (c) `full system` traces.

TABLE II: GEM5 Results

	Threads (7649395)			Snoop (485497)			FullSystem (8.4×10^9)		
	RT (s)	Size	Ratio	RT (s)	Size	Ratio	RT (s)	Size	Ratio
Proposed Method	63	146	99.16%	78	126	98.8%	<5 days	132	98.47%
Model Synthesis [17]	100	128	96.49%	19	117	94.52%	<6 hours	100	65.19%

size, while a higher threshold increases both runtime and size. The significant runtime of our method for the FS trace is due to the evaluation phase. Following each model refinement, an evaluation is conducted, contributing to the extended runtime. User vision can enhance the model's efficiency. For instance, incorporating specific paths into the model can boost accuracy, thereby decreasing the number of refinement iterations and finally reducing runtime. Based on our experiments, beyond a certain accuracy level, further improvements significantly increase model size and runtime. Thus, user insight is vital to find the optimal accuracy threshold.

VI. Conclusion

This paper presents a method and a new evaluation metric to accurately infer message flow specifications from complex system traces, outperforming existing methods. We introduced an evaluation approach for model accuracy. Our method excels in handling large, intricate traces and provides insights into system protocols and component interactions, aiding in system design and efficiency improvements. It should be noted that this mining approach isn't entirely automated; it requires user input for enhanced performance, serving as a tool for designers. Future work includes optimizing runtime with trace slicing, enhancing accuracy with techniques like window slicing, and exploring deep learning for identifying valid paths using high-accuracy sample models.

References

[1] N. Binkert et al. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, aug 2011.

[2] I. Beschastnikh, Y. Brun, S. Schneider, M. Sloan, and M. D. Ernst. Leveraging existing instrumentation to automatically infer invariant-constrained models. In *19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, 2011.

[3] J. Yang, D. Evans, D. Bhardwaj, T. Bhat, and M. Das. Perracotta: Mining temporal api rules from imperfect traces. In *28th International Conference on Software Engineering*, pages 282–291, 2006.

[4] A. Mrowca, M. Nocker, S. Steinhorst, and S. Günnemann. Learning temporal specifications from imperfect traces using bayesian inference. In *Proceedings of the 56th Annual Design Automation Conference 2019*.

[5] W. Li, A. Forin, and S. A. Seshia. Scalable specification mining for verification and diagnosis. In *Proceedings of the 47th Design Automation Conference*, pages 755–760, 2010.

[6] S. Hertz, D. Sheridan, and S. Vasudevan. Mining hardware assertions with guidance from static analysis. *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2013.

[7] A. Danese, F. Filini, and G. Pravadelli. A time-window based approach for dynamic assertions mining on control signals. In *IFIP/IEEE International Conference on Very Large Scale Integration*, 2015.

[8] A. Danese, T. Ghasempouri, and G. Pravadelli. Automatic extraction of assertions from execution traces of behavioural models. In *Proceedings of the 2015 Design, Automation, & Test in Europe Conference*.

[9] A. Danese, N. D. Riva, and G. Pravadelli. A-team: Automatic template-based assertion miner. In *Proceedings of the 54th Annual Design Automation Conference 2017*.

[10] P. Chang and L. Wang. Automatic assertion extraction via sequential data mining of simulation traces. In *ASPDAC*, 2010.

[11] L. Liu and S. Vasudevan. Automatic generation of system level assertions from transaction level models. *Journal of Electronic Testing*, 2013.

[12] M. R. Ahmed, H. Zheng, P. Mukherjee, M. C. Ketkar, and J. Yang. Mining message flows from system-on-chip execution traces. In *22nd ISQED*. IEEE, 2021.

[13] X. Meng, K. Raj, S. Ray, and K. Basu. Sevnoc: Security validation of system-on-chip designs with noc fabrics. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[14] M. J. Heule and S. Verwer. Software model synthesis using satisfiability solvers. *Empirical Software Engineering*, 18(4), 2013.

[15] K. J. Lang, B. A. Pearlmutter, and R. A. Price. Results of the abbadingo one dfa learning competition and a new evidence-driven state merging algorithm. In *ICGI*, 1998.

[16] V. Ulyantsev and F. Tsarev. Extended finite-state machine induction using sat-solver. In *ICMLAW*, volume 2, 2011.

[17] H. Zheng, M. R. Ahmed, P. Mukherjee, M. C. Ketkar, and J. Yang. Model synthesis for communication traces of system designs. In *2021 IEEE 39th International Conference on Computer Design (ICCD)*, pages 492–499, CA, USA, Oct 2021.

[18] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Occam's razor. *Information Processing Letters*, 24(6), 1987.

ICCAD comments

ICCAD 2023 Reviews for Submission number 783
Title: A Mining Approach for Inferring Message Flows From System Communication Traces Authors: Bardia Nadimi and Hao Zheng

VII. Reviewer 1

A. *Reviewer's Scores:*

Relevance to ICCAD (1-5): 4
Novelty of technical ideas (1-5): 3
Theoretically sound (1-5): 2
Practically useful (1-5): 4
Thoroughness of the research (1-5): 3
Clarity and language (1-5): 4
Overall Recommendation (1-5): 3

B. *Summary*

This paper presents a method for ”specification mining”, which is the process of extracting specifications from observable system data (e.g., execution traces or log-files). Compared to previous work, the authors claim to have been the first to focus not on the size of the resulting model, but on the accuracy. To this end, the authors define a new way of measuring this accuracy and perform experiments that compare their algorithm to previous work in the field.

C. *Strengths*

The paper has a clear structure, is well written and the information is presented cleary. The proposed approach is convincingly shown to be relevant (in the related work section) and evaluated rather thoroughly.

D. *Weaknesses*

The theoretical background could be more fleshed out in my opinion. Although the content is explained really well, I would have appreciated a more complex mined message flow as an example.

E. *Detailed Comments*

I would have appreciated a more thorough theoretical analysis of the results. Specification mining seems to me like it has close connections to formal language theory (e.g., formal grammals and their corresponding accepting automatons). I would have appreciated at least a mention of these connections.

Furthermore, I think the authors could have provided more clear examples of mined specifications that would not have been easy to describe a-priori (which is the claimed main advantage of specification mining). I understand that the authors chose the given examples (like a memory read that was compared to the actual specification in the gem-5 documentation) so that readers can understand the problem and I agree that these examples are well chosen. However, I would have liked an additional example of a mined message flow that is valid but also complex enough so that it is clear why manual specification could not have been done easily.

I think there is something wrong with the sentence: ” [...] and finding all available paths in the causality graph in a linear time is not achievable due to the enormous size of the available paths.” in Section IV A. Maybe my understanding is wrong but I don't think that the size of the causality graph has any influence on whether computations on it (like extracting valid traces) fall into a specific complexity class. Maybe the authors wanted to make a statement about the feasibility of the computation, which does depend on the absolute size of the graph.

Overall, the paper is sound work that seems to improve upon the state of the art in meaningful ways by exploring a different trade-off than the presented related work. The paper does have some weak spots, that should be addressed.

VIII. Reviewer 2

A. *Reviewer's Scores*

Relevance to ICCAD (1-5): 4
Novelty of technical ideas (1-5): 4
Theoretically sound (1-5): 4
Practically useful (1-5): 2
Thoroughness of the research (1-5): 3
Clarity and language (1-5): 4
Overall Recommendation (1-5): 3

B. *Summary*

This paper presents a method to mine traces of communication among SOC communication to automatically extract a model whose behaviour can cover the generated traces with an acceptable level of accuracy in an iterative process of creating/refining model and evaluating it.

C. *Strengths*

The paper is exceptionally well written. Very easy to read, even for a non-specialist in this field. The idea also seems enticing, and the authors have reviewed the state of the art well and concretely pointed out the differentiation. The experimental results are also OK, though I am not entirely convinced about synthetic traces.

D. *Weaknesses*

My principal problem with this work is how it is used in practice. To create a SOC design, even at the GEM5 level, one needs a specification to start with, even if this specification is almost always non-executable. Once the SOC design team has a specification, and the model, if I understood correctly, this method will mine the traces generated by the simulation and then derive a model of the SOC. This model abstracted as an FSM, will generate a trace that covers the REAL trace generated by the SOC design (SystemC or GEM5 etc). And then what ? This model is obviously neither complete nor an exact match - the coverage at best can asymptotically reach the actual trace. How do I use this mined model?

E. Detailed Comments

This intriguing work is very well written and backed by decent experiments. I enjoyed reading it.

I am not an expert in mining models from communication traces. I have, however, architected a few real-life industrial SOCs. I can't figure out how this will help me improve my verification or validation strategy. The end-user makes many critical decisions, like the confidence of support for nodes and edges. I could not see how the end-user knows what a reasonable threshold is.

IX. Reviewer 3

A. Reviewer's Scores

Relevance to ICCAD (1-5): 3
Novelty of technical ideas (1-5): 3
Theoretically sound (1-5): 4
Practically useful (1-5): 2
Thoroughness of the research (1-5): 3
Clarity and language (1-5): 4
Overall Recommendation (1-5): 3

B. Detailed Comments

The paper addresses the issue of system model reconstruction from monitored message traces. Available approaches try to infer a FSA model of minimal size. This paper aims at also achieving a certain required level of accuracy. This novelty is, basically, the content of section IV C and D.

The paper is quite well written and experiments show the intended improvement in coverage. What the paper does not show is the way in which this really is useful in any way.

X. Reviewer 4

A. Reviewer's Scores

Relevance to ICCAD (1-5): 4 Novelty of technical ideas (1-5): 2 Theoretically sound (1-5): 3 Practically useful (1-5): 2 Thoroughness of the research (1-5): 4 Clarity and language (1-5): 2 Overall Recommendation (1-5): 3

B. Summary

The paper proposes a new method to infer spec model from communication traces of SoC design. The communication traces represent messages exchanged by different components during system execution. They evaluate the method using synthetic traces and real traces generated from GEM5 simulator.

C. Strengths

* This paper is solving an important and hard problem, which is build a model to represent sophisticated system protocols. * The paper designs new metrics, new evaluation method and refinement algorithm to improve their results.

D. Weaknesses

* The writing needs to be improved. Many typos out there. * The model constructed from only communication traces seem not sufficiently complete.

E. Detailed Comments

Please justify the model built by the communication traces are good enough to model very complex systems protocols of SoCs.