Reduction of Verification Conditions for Concurrent System using Mutually Atomic Transactions

Malay K. Ganai[1] and Sudipta Kundu[2]

[1]NEC Labs America, Princeton, NJ, USA
[2]University of California, San Diego, CA, USA

Abstract. We present a new symbolic method based on partial order reduction to reduce verification problem size and state space of a multi-threaded concurrent system with shared variables and locks. We combine our method with a previous token-based approach that generates verification conditions directly without a scheduler. For a bounded unrolling of threads, the previous approach adds concurrency constraints between all pairs of global accesses. We introduce the notion of Mutually Atomic Transactions (MAT), i.e., two transactions are mutually atomic when there exists exactly one conflicting shared-access pair between them. We propose to reduce the verification conditions by adding concurrency constraints *only* between MATs. Such an approach removes *all* redundant interleavings, thereby, achieves state reduction as well. We guarantee that our MAT-based reduction is both adequate (preserves all the necessary interleavings) and optimal (no redundant interleaving), for a bounded depth analysis. Our experimental results show the efficacy of our approach in reducing the state space and the verification problem sizes by orders of magnitude, and thereby, improving the overall performance, compared with the state-of-the-art approaches.

1 Introduction

Verification of multi-threaded programs is hard due to complex and un-expected interleaving between the threads [1]. In practice, the verification efforts often use *incomplete* methods, or *imprecise* models, or sometimes both, to address the scalability of the problem. The verification model is typically obtained by composing individual thread models using interleaving semantics, and model checkers are applied to systematically explore the global state space. To combat the state explosion problem, most methods employ partial-order reduction techniques to restrict the state-traversal to only a representative subset of all interleavings, thereby, avoiding exploring the redundant interleaving among independent transitions [2–4]. Explicit model checkers [5–9] explore the states and transitions of concurrent system by explicit enumeration, while symbolic model checkers [10–17] uses symbolic methods. We focus on symbolic approaches based on SMT (Satifiability Modulo Theory) to generate efficient verification conditions. Based on how verifications models are built, symbolic approaches can be broadly classified into: *synchronous* (i.e., with scheduler) and *asynchronous* (i.e., without scheduler) modeling.

Synchronous modeling: In this category of symbolic approaches [10–12], a synchronous model of concurrent programs is constructed with a scheduler. The scheduler is then constrained—by adding guard strengthening—to explore only a subset of interleaving. To guarantee correctness (i.e., cover all necessary interleavings), the scheduler must allow context-switch between accesses that are conflicting (i.e. dependent). One determines statically (i.e., conservatively) which pair-wise locations require context switches, using persistent [4]/ample [18] set computations. One can further use lock-set and/or lock-acquisition history analysis [11, 19–21], and conditional dependency [16, 22] to reduce the set of interleavings need to be explored (i.e., remove redundant interleavings). Even with these state reduction methods, the scalability problem remains. To overcome that, researchers have employed sound abstraction [7] with bounded number of context switches [23] (i.e., under-approximation), while some others have used finite-state model abstractions [13], combined with proof-guided method to discover the context switches [14].

Asynchronous Modeling: In this category, the symbolic approaches such as TCBMC [15] and token-based [17] generate verification conditions directly without constructing a synchronous model of concurrent programs, i.e., without using a scheduler. These verification conditions are then solved by satisfiability solvers. To our knowledge so far, the state-reduction based on partial-order has *hardly* been exploited in the asynchronous modeling approaches [15, 17]. We will focus primarily in that direction.

Our Approach: We present a new SMT-based method—combining partial-order reduction with the previous token-based approach [17]—to reduce verification problem size and state-space for multi-threaded concurrent system with shared variables and locks. For a bounded unrolling of threads, the previous approach adds concurrency constraints between all pairs of global accesses, thereby allowing redundant interleavings. Our goal is to reduce the verification conditions by removing *all* redundant interleavings (i.e., guarantee optimality) but keeping the necessary ones (i.e., guarantee adequacy). We first introduce the notion of *Mutually Atomic Transactions* (MAT), i.e., two transactions are mutually atomic when there exists exactly one conflicting shared-access pair between them. We then propose an algorithm to identify an optimal and adequate set of MATs. For each MAT in the set, we add concurrency constraints *only* between the first and last accesses of the transactions, and not in-between. Our MAT-based approach achieves reduction both in state-space as well as in the size of verification conditions. We guarantee that our MAT-based reduction is both adequate (preserves all the necessary interleavings) and optimal (no redundant interleaving), for a bounded depth analysis. We implemented our approach in a SMT-based prototype framework, and demonstrated the efficacy of our approach against the state-of-the-art SMT-based approaches based on asynchronous modeling [17], and synchronous modeling [16], respectively.

Outline: We provide an informal overview of our MAT-based reduction approach in Section 2, followed by formal definitions and notations in Section 3. In Section 4, we present a flow diagram of our new SMT-based method. We give an algorithm for identifying an adequate and optimal set of MATs in Section 5, followed by a presentation of adequacy and optimality theorems in Section 6. We present our experimental results in Section 7, and conclusions in Section 8.

2 An Overview

We motivate our readers with a following example, which we use to guide the rest of our discussion. Consider a two-threaded concurrent system comprising threads M_1 and M_2 with local variables a_i and b_i, respectively, and shared (global) variables x, y, z. This is shown in Figure 1(a), as a concurrent control flow graph (CCFG) with a fork-join structure. Each shared statement associated with a node is *atomic*, i.e., it cannot be interrupted. Further, each node is associated with at most one shared access. A node with a shared write/read access of variable x is identified as $W(x)/R(x)$. We use the notation ? to denote a non-deterministic input to a variable.

Given such a concurrent system, the goal of the token-based approach [17] is to generate verification conditions that capture necessary interleaving for some bounded unrolling of the threads, aimed at detecting reachability properties such as data races and assertion violations. These verification conditions together with the property constraints are encoded and solved by an SMT solver. A satisfiable result is typically accompanied by a trace—comprising data input valuations, and a total-ordered thread interleaving—that is witness to the reachability property. On the other hand, an unsatisfiable result is followed by these steps (a)—(c): (a) increase unroll depths of the threads, (b) generate verification conditions for increased depths, and (c) invoke SMT solver on these conditions. Typically, the search process (i.e., to find witnesses) is terminated when a resource—such as time, memory or bound depth—reaches its limit. For effective implementation, these verifications constraints are added on-the-fly, lazily and incrementally at each unrolled depth. Though the approach captures all necessary interleaving, it however does not prevent redundant interleavings.

In this work, our goal is to remove *all* the redundant interleavings but keep the necessary ones for a given unroll bound. We focus on reducing the verification conditions, as generated in the token-passing modeling approach [17]. To understand how we remove redundancy, we first present a brief overview of such a modeling approach.

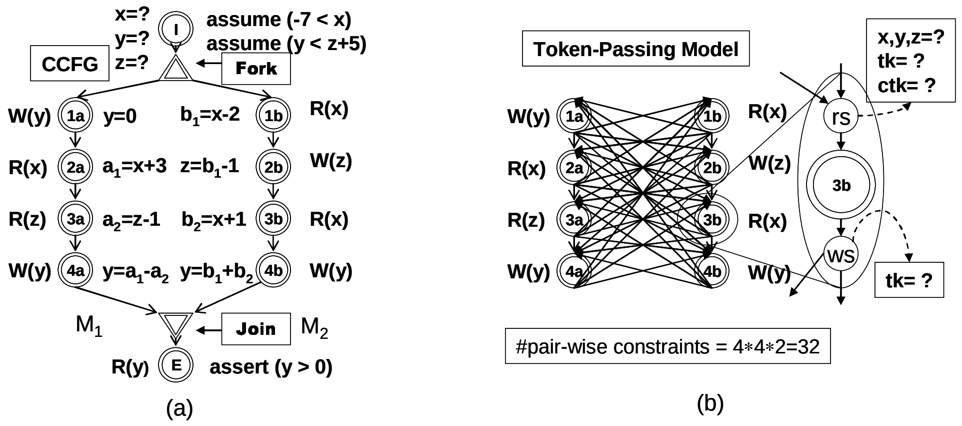


Fig. 1. (a) Concurrent system, shown as thread CFGs, with threads M_1, M_2 and local variables a_i, b_i respectively, communicating with shared variable x, y, z, and (b) Token-passing Model [17].

2.1 Token-passing Model

The main idea of token-passing model (TPM) is to introduce a single Boolean token `tk` and a clock vector `ctk` in a model, and then manipulate the passing of the token to capture all necessary interleavings in the given system. The clock vector records the number of times the token `tk` is passed and is synchronized when the token is passed. Unlike a synchronous model, TPM does not have a scheduler in the model. The verification model is obtained two phases.

In the *first* phase, the goal is obtain abstract and decoupled thread models. Each thread is decoupled from the other threads by localizing all the shared variables. For the example shown in Figure 1(a), M_1 and M_2 are decoupled by renaming (i.e., localizing) shared variable such as x to x_1 and x_2, respectively. Each model is then abstracted by allowing renamed (i.e., localized) variables to take non-deterministic values at every shared access. To achieve that, each shared access node (in every thread) is instrumented with two control states as follows: (a) an atomic *pre-access control state*, referred to as *read_sync* block, is inserted before each shared access, and (b) an atomic *post-access control state*, referred to as *write_sync* block, is inserted after each shared access. In *read_sync* block, all localized shared variables obtain non-deterministic values.

As an example, we show the token-passing model in the Figure 1(b). For clarity of presentation, we did not show renaming of the shared variables, but for all our purpose we consider them to be local to the thread, i.e., x of thread M_i and x of M_j are *not* the same variable. In such a model, atomic control states `rs` and `ws` are inserted pre and post of shared accesses in decoupled model, respectively. As highlighted for a control state $3b$, we add the following statements in the corresponding `rs` node, i.e., `x=?,y=?,z=?,tk=?,ctk=?`. Similarly, we add `tk=?` in `ws` node. (`?` denotes the non-deterministic values.)

Note, the transition (update) relation for each localized shared variable depends on other local variables, thereby, making the model independent (i.e., decoupled). However, due to non-deterministic read values, the model have additional behaviors, hence, it is an abstract model.

In the *second* phase, the goal is to remove the imprecision caused due to abstraction. In this phase, the constraints are added to restrict the introduced non-determinism and to capture the necessary interleavings. More specifically, for each pair of shared access state (in different threads), *token-passing constraints* are added from the *write_sync* node of a shared access to the *read_sync* node of the other shared access. Intuitively, these token-passing constraints allow passing of the token from one thread to another, giving a total order in the shared accesses. Furthermore, these constraints allow to synchronize the values of the localized shared variables from one thread to another. Together, the token-passing constraints captures *all and only* the necessary interleavings that are sequentially consistent [24] as stated in the following theorem.

Theorem 1 (Ganai, 2008 [17]). *The token-based model is both complete, i.e., it allows only sequentially consistent traces, and sound, i.e., captures all necessary interleaving, for a bounded unrolling of threads. Further, the size of pair-wise constraints added grow quadratically (in the worse case) with the unrolling depth.*

In Figure 1(b), we show a token-passing constraint as a directed edge from a *write_sync* `ws` node of one thread to a *read_sync* `rs` node of another. Note, these constraints are added for all pairs of `ws` and `rs` nodes. A synchronization constraint from M_1 to M_2 will include $x_2 = x_1 \wedge y_2 = y_1 \wedge z_2 = z_1 \wedge tk_2 = 1 \wedge tk_1 = 0 \wedge ctk_2 = ctk_1$, where token-passing is enforced by assertion/de-assertion of corresponding token variable. (Recall, v_i is localized variable in M_i corresponding to shared variable v). As shown, one adds $4 * 4 * 2 = 32$ such token-passing constraints for this example.

Improvement Scope: Though the above approach captures all and only necessary interleavings, it also allows interleavings that may be redundant (i.e. equivalent). For example, the interleaving $\sigma_1 \equiv 1b \cdot 2b \cdot 1a \cdot 3b \cdot 4b \cdot 2a \cdot 3a \cdot 4a$, and $\sigma_2 \equiv 1a \cdot 2a \cdot 1b \cdot 2b \cdot 3a \cdot 3b \cdot 4b \cdot 4a$, are equivalent as in these interleavings the conflicting pairs $(2b, 3a)$, $(1a, 4b)$, $(4b, 4a)$ are in the same happens-before order, besides the thread program order pairs. (Note, "$\cdot$" denotes concatenation). The previous-approach [17] will explore both the interleavings.

In the following sections, we build our approach on such a token-passing model to identify pair-wise constraints that can be safely removed, without affecting soundness and completeness, and guaranteeing optimality by removing all redundant interleavings. For the example in Figure 1, our approach removes 24 such pair-wise constraints (as shown in Figure 4), and yet covers all the necessary interleavings with no redundancy. To illustrate, our approach allows σ_1, and not any other equivalent (to σ_1) interleavings such as σ_2. Note, the choice of a representative interleaving will depend on a given thread prioritization, as discussed later.

2.2 Mutually Atomic Transactions

Our partial-order reduction approach is based on the concept of mutually atomic transactions, MAT for short. Intuitively, let a transaction be a sequence of statements in a thread, then we say two transactions tr_i and tr_j of threads M_i and M_j, respectively, are mutually atomic transactions if and only if there exists exactly one conflicting shared-access pair between them, and the statements containing the shared-access pair is the last one in each of the transactions. (We will present a more formal definition later).

Now we illustrate the concept of MAT using an example as shown in Figure 2. From the control state pair $(1a, 1b)$, there are two reachable control states with conflicting accesses, i.e., $(3a, 2b)$ and $(1a, 4b)$. Corresponding to that we have two MATs $m = (tr_1 = 1a \cdots 3a, tr_2 = 1b \cdots 2b)$ (Figure 2(a)) and $m' = (tr'_1 = 1a, tr'_2 = 1b \cdots 4b)$ (Figure 2(b)), respectively. Similarly, from $(1a, 2b)$ we have $m'' = (tr''_1 = 1a, tr''_2 = 2b \cdots 4b)$ (Figure 2(c)). In general, there could be multiple possible MATs for our examples.

In a more general setting with conditional branching, we identify MATs by exploring beyond conditional branches, as illustrated in the Figure 2(d), with a conditional branch denoted as a diamond node, and control states A_i, B_i, C_i denoted as dark ovals. Starting from (A_1, A_2), we have following control path segments, $tr_{11} = A_1 \cdots B_1$, $tr_{12} = A_1 \cdots C_1$, $tr_{21} = A_2 \cdots B_2$, and $tr_{22} = A_2 \cdots C_2$ (shown as ovals). For each of the four combinations of tr_{1i}, tr_{2j}, we define MAT separately.

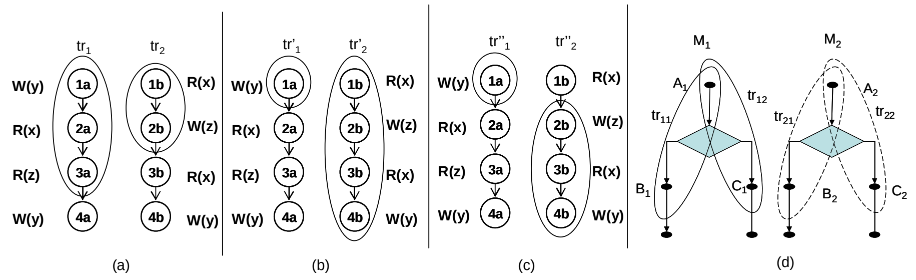


Fig. 2. (a) m=(tr_1, tr_2), (b) m'=(tr'_1, tr'_2), (c) m''=(tr''_1, tr''_2) (d) MATs for branches.

Given a MAT (tr_i, tr_j), we can have only two equivalent classes of interleavings [25]. One represented by $tr_i \cdot tr_j$, i.e., tr_i executing before tr_j and other by $tr_j \cdot tr_i$, i.e., tr_j executing before tr_i. (Note, "$\cdot$" represent concatenations.) For a given MAT $m = (tr_1, tr_2)$ shown in Figure 2(a), the interleavings $\sigma_1 \equiv 1a \cdot 2a \cdot 3a \cdot 1b \cdot 2b$ and $\sigma_2 \equiv 1b \cdot 2b \cdot 1a \cdot 2a \cdot 3a$ represent the two equivalent classes, respectively. In other words, given a MAT, the associated transactions can be considered *atomic pair-wise*, and one can avoid interleaving them *in-between*. In general, transactions associated with different MATs may not be atomic. For example, tr_1 is not atomic with tr''_2 (Figure 2(a),(c)).

Intuitively, it would be desirable to have a set of MATs such that, by adding token-passing constraints only between MATs, we will not only miss any necessary interleaving but also remove all the redundant interleaving. In Section 5, we describe such an algorithm *GenMAT* to compute an optimal and adequate set of MATs. For our example one such set is $\{(1a \cdots 3a, 1b \cdots 2b), (4a, 1b \cdots 4b), (1a, 3b \cdots 4b), (4a, 3b \cdots 4b), (2a \cdots 4a, 3b \cdots 4b)\}$. Based on the set, we add only 8 token-passing constraints (Figure 4), compared to 32 (Figure 1(b)).

At this point we would like to highlight the salient features of our approaches *vis-a-vis* previous works. A previous approach [9] on partial-order reduction used in a explicit model checking framework does not guarantee optimality. Though such guarantee is provided in a recent symbolic approach (using synchronous modeling) [16], our approach goes further in reducing problem sizes, besides an optimal reduction in the state space. Our approach obtains state space reduction by removing constraints (i.e., adding fewer token-passing constraints), while the approach [16] obtains it by adding more constraints (i.e., constraining the scheduler). In our experiments, we observed that our approach is order-of-magnitude more memory efficient compared to the approaches [16,17]. Our approach is orthogonal to the approaches that exploit transaction-based reductions [11,19,20]. Nevertheless, we can exploit those to identify unreachable conflicting pairs, and further reduce the necessary token-passing constraints.
Contributions Highlights:

- We are first to exploit partial order reduction techniques in a SMT-based bounded model checking using token-passing modeling approach. We developed a novel approach—based on MAT—to reduce verification conditions, both in size and state space for concurrent systems.
- We prove that our MAT-based reduction is both adequate (preserves all and only the necessary interleavings) and optimal (no redundant interleaving, as determined statically), for a bounded depth analysis.
- Our approach outperforms other approaches [16, 17] by orders of magnitude, both in performance and size of the verification problems.

3 Formal Definitions

With the brief informal overview, we present our approach in a more formal setting. We consider a multi-threaded system $\mathcal{CS}$ comprising a finite number of deterministic bounded-stack threads communicating with shared variables, some of which are used as synchronization objects such as locks. Let $M_i (1 \leq i \leq N)$ be a thread model represented by a control and data flow graph of the sequential program it executes. Let T_i represent the set of 4-tuple transitions (c, g, u, c') of thread M_i, where c, c' represent the control states, g is Boolean-valued enabling condition (or *guard*) on program variables, u is an update function on program variables. Let $\mathcal{T} = \bigcup_i T_i$ be the set of all transitions. Let V_i be set of local variables in T_i and $\mathcal{V}$ be set of (global) shared variables. Let $\mathcal{S}$ be the set of global states of the system, and a state $s \in \mathcal{S}$ is valuation of all local and global variables of the system. A global transition system for $\mathcal{CS}$ is an interleaved composition of the individual thread models, M_i. Each transition consists of global firing of a local transition $t_i = (a_i, g_i, u_i, b_i) \in \mathcal{T}$. If enabling predicate g_i evaluates to true in s, we say that t_i is *enabled* in s.

3.1 Notation

We define the notion of a run of a multi-threaded program as an observation of events such as global accesses, thread creations and thread termination. If the events are ordered, we call it a *total order run*. We define a set A_i of shared accesses corresponding to a read $R_i(x)$ and a write $W_i(x)$ of a thread M_i where $x \in \mathcal{V}$. For $a_i \in A_i$, we use $var(a_i)$ to denote the accessed shared variable. We use $\vdash_i$ to denote the beginning and $\dashv_i$ to denote the termination of thread M_i, respectively. The alphabets of events of thread M_i is a set $\Sigma_i = A_i \cup \{\vdash_i, \dashv_i\}$. We use $\Sigma = \cup_i \Sigma_i$ to denote a set of all events. A word σ defined over the alphabet set Σ, i.e., $\sigma \in \Sigma^*$ is a string of alphabet from Σ, with $\sigma[i]$ denoting the i^{th} access in σ, and $\sigma[i,j]$ denoting the access substring from i^{th} to j^{th} position, i.e., $\sigma[i] \cdots \sigma[j]$ ($\cdot$ denotes concatenation). $|\sigma|$ denotes the length of the word σ. We use $\pi(\sigma)$ to denote a permutation of alphabets in the word σ. We use $\sigma \mid_i$ to denote the projection of σ on thread M_i, i.e., inclusion of the actions of M_i only.

Transaction: A transaction is a word $tr_i \in \Sigma_i^*$ that may be *atomic* (i.e., uninterrupted by other thread) with respect to some other transactions. If it is atomic with respect to all other thread transactions, we refer it as *independent transaction*.

Schedule: Informally, we define a schedule as a total order run of a multi-threaded program where the accesses of the threads are interleaved. Formally, a schedule is a word $\sigma \in \Sigma^*$ such that $\sigma \mid_i$ is a prefix of the word $\vdash_i \cdot A_i^* \cdot \dashv_i$.

Happens-before Relation ($\prec, \preceq$): Given a schedule σ, we say e happens-before e', denoted as $e \prec_\sigma e'$ if $i < j$ where $\sigma[i] = e$ and $\sigma[j] = e'$. We drop the subscript if it is obvious from the context. Also, if the relation is not strict, we use the notation $\preceq$. If $e, e' \in \Sigma_i$ and e precedes e' in σ, we say that they are in a *thread program order*, denoted as $e \prec_{po} e'$.

Sequentially consistent: A schedule σ is sequentially consistent [24] iff (a) $\sigma \mid_i$ is in thread program order, (b) each shared read access gets the last data written at the same address location in the total order, and (c) synchronization semantics is maintained, i.e., the same locks are not acquired in the run without a corresponding release in between. We only consider schedules (and their permutations) that are sequentially consistent.

Conflicting Access: We define a pair $a_i \in A_i, a_j \in A_j$, $i \neq j$ conflicting, if they are accesses on the same shared variable (i.e., $var(a_i) = var(a_j)$) and one of them is write access. We use $\mathcal{C}_{ij}$ to denote the set of tuples (a_i, a_j) of such conflicting accesses. We use Sh_{ij} to denote a set of shared variables—between M_i and M_j threads—with at least one conflicting access, i.e., $Sh_{ij} = \{var(a_i) | (a_i, a_j) \in \mathcal{C}_{ij}\}$. We define $Sh_i = \bigcup_{i \neq j} Sh_{ij}$, i.e., a set of variables shared between M_i and M_k, $k \neq i$ with at least one conflicting access. In general, $Sh_{ij} \subseteq (Sh_i \cap Sh_j)$.

Dependency Relation (D): A relation $D \subseteq \Sigma \times \Sigma$ is a dependency relation iff for all $(e, e') \in D$, one of the following holds: (1) $e, e' \in \Sigma_i$ and $e \prec_{po} e'$, (2) $(e, e') \in \mathcal{C}_{ij}$, (3) $e = \dashv_i$, $e' = \dashv_j$ for $i \neq j$. Note, the last condition is required when the order of thread termination is important. If $(e, e') \notin D$, we say the events e, e' are *independent*. The dependency relation in general, is hard to obtain; however, one can obtain such relation conservatively using static analysis [4], which may result in a larger dependency set than required. For our reduction analysis, we assume such a relation is provided.

Equivalency Relation ($\simeq$): We say two schedules $\sigma_1 = w \cdot e \cdot e' \cdot v$ and $\sigma_2 = w \cdot e' \cdot e \cdot v$ are equivalent (Mazurkiewicz's trace theory [25]), denoted as $\sigma_1 \simeq \sigma_2$, if $(e, e') \notin D$. An equivalent class of schedules can be obtained by iteratively swapping the consecutive independent events in a given schedule. Final values of both local and shared variables remains unchanged when two equivalent schedules are executed.

A *partial order* is a relation $R \subseteq \Sigma \times \Sigma$ on a set Σ, that is reflexive, antisymmetric, and transitive. A partial order is also a *total order* if, for all $e, e' \in \Sigma$, either $(e, e') \in R$, or $(e', e) \in R$. *Partial order*-based reduction (POR) methods [4] avoid exploring all possible interleavings of shared access events. Note, if $(e, e') \in D$, all equivalent schedules agree on either $e \prec e'$ or $e' \prec e$, but not both.

Definition 1 (MAT). *We say two transactions tr_i and tr_j of threads M_i and M_j, respectively, are mutually atomic iff except for the last pair, all other event pairs in the corresponding transactions are independent. Formally, a Mutually Atomic Transactions (MAT) is a pair of transactions, i.e., $(tr_i, tr_j), i \neq j$ iff $\forall k\ 1 \leq k \leq |tr_i|, \forall h\ 1 \leq h \leq |tr_j|$, $(tr_i[k], tr_j[h]) \notin D$ $(k \neq |tr_i|$ and $h \neq |tr_j|)$, and $tr_i[|tr_i|], tr_j[|tr_j|]) \in D$.*

Given a MAT (tr_i, tr_j), an interesting observation (as noted earlier) is that a word $w = tr_i \cdot tr_j$ is equivalent to any word $\pi(w)$ obtained by swapping any consecutive events $tr_i[k]$ and $tr_j[h]$ such that $k \neq |tr_i|$ and $h \neq |tr_j|$. Similarly, the word $w' = tr_j \cdot tr_i$ is equivalent to any word $\pi(w')$ obtained as above. Note, $w \not\simeq w'$. Therefore, for a given MAT, there are only two equivalent classes, represented by w and w'. In other words, given a MAT, the associated transactions are *atomic pair-wise*.

4 Token-passing Model using MAT

We exploit the pair-wise atomicity of MATs in a token-based model as follows: Let $c(e)$ represent the control state of the thread where the corresponding event e occurs. For the given MAT $(tr_i = f_i \cdots l_i, tr_j = f_j \cdots l_j)$, we *only* add token-passing constraints from $c(l_j)$ to $c(f_i)$, and $c(l_i)$ to $c(f_j)$, respectively. Recall, such constraints are added between the corresponding pre and post- access blocks as discussed in Section 2.1.

Adequacy of MATs Given a schedule $\sigma = w_1^1 \cdots w_N^1 \cdots w_1^n \cdot w_N^n$, $w_i^k \in \Sigma_i^*$, $1 \leq k \leq n$, $1 \leq i \leq N$. We define a set of ordered pairs CSP as follows: $CSP(\sigma) = \{(l_i^k, f_{i'}^{k'}) | 1 \leq i, i' \leq N, 1 \leq k, k' \leq n\}$ where f_i^k and l_i^k denote the first and last accesses of w_i^k; and $w_{i'}^{k'}$ is a non-empty word adjacent right of w_i^k. Note, $CSP(\sigma)$ captures the necessary interleaving pairs to obtain the schedule, i.e., if we add token passing constraints between every pair of control states $(a, b) \in CSP(\sigma)$, we allow the schedule σ. For a given MAT $\alpha = (f_i \cdots l_i, f_j \cdots l_j)$, we define a set of interleaving ordered pairs, $TP(\alpha) = \{(l_i, f_j)), (l_j, f_i))\}$. Given a set of $\mathcal{MAT}_{ij}$, we define $TP(\mathcal{MAT}_{ij}) = \bigcup_{\alpha \in \mathcal{MAT}_{ij}} TP(\alpha)$, and denote it as TP_{ij}. We say a token-passing pairs set TP is *adequate* iff for every schedule σ in the multi-threaded system, $CSP(\sigma) \subseteq TP$. A set $\mathcal{MAT}$ is *adequate* iff TP is adequate. *Note, the size of TP is upper bounded by quadratic number of pair-wise accesses.*

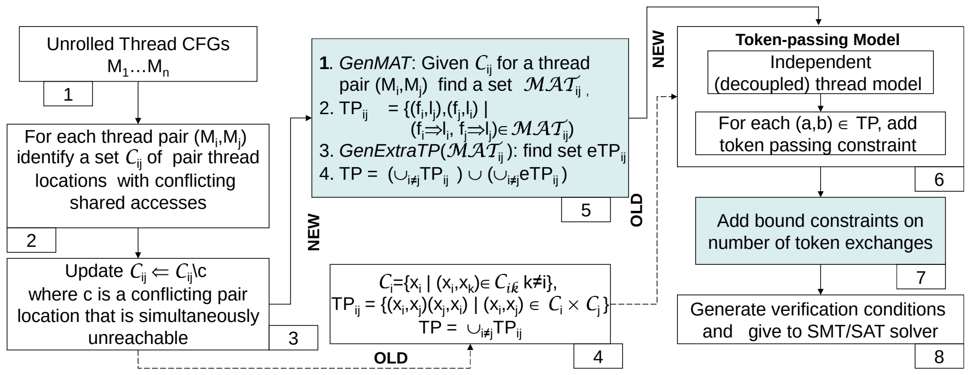

Unrolled Thread CFGs $M_1 \ldots M_n$
1
For each thread pair (M_i,M_j) identify a set C_{ij} of pair thread locations with conflicting shared accesses
2
Update $C_{ij} \Leftarrow C_{ij} \backslash c$ where c is a conflicting pair location that is simultaneously unreachable
3
NEW
1. *GenMAT*: Given C_{ij} for a thread pair (M_i,M_j) find a set $\mathcal{MAT}_{ij}$,
2. $TP_{ij} = \{(f_i,l_j),(f_j,l_i) \mid (f_i \Rightarrow l_i, f_j \Rightarrow l_j) \in \mathcal{MAT}_{ij})$
3. *GenExtraTP*($\mathcal{MAT}_{ij}$): find set eTP_{ij}
4. $TP = (\cup_{i \neq j} TP_{ij}) \cup (\cup_{i \neq j} eTP_{ij})$
5
$C_i = \{x_i \mid (x_i,x_k) \in C_{ik}, k \neq i\}$,
$TP_{ij} = \{(x_i,x_j)(x_j,x_i) \mid (x_i,x_j) \in C_i \times C_j\}$
$TP = \cup_{i \neq j} TP_{ij}$
4
OLD
NEW
OLD
Token-passing Model
Independent (decoupled) thread model
For each $(a,b) \in$ TP, add token passing constraint
6
Add bound constraints on number of token exchanges
7
Generate verification conditions and give to SMT/SAT solver
8

Fig. 3. Reducing verification conditions in a token-passing model using MAT.

We use procedure $GenMAT$ (ref. Section 5) to obtain a set of $\mathcal{MAT}_{ij}$. If $Sh_{ij} \subsetneq Sh_i \cup Sh_j$, we use procedure $GenExtraTP$ (ref. Section 6) to generate an extra token-passing pairs set eTP_{ij} from $\mathcal{MAT}_{ij}$. We then construct the adequate set TP as $(\bigcup_{i \neq j} TP_{ij}) \cup (\bigcup_{i \neq j} eTP_{ij})$. We give an overview of using MATs in a token-passing model to selectively add token-passing constraints as shown in Figure 3.

Step 1,2: Given a set of unrolled threads $M_1 \cdots M_N$, we obtain a set of conflicting pair of control locations $\mathcal{C}_{ij}$ for each thread pair M_i, M_j.

Step 3: From the set $\mathcal{C}_{ij}$, we remove the pairs that are unreachable simultaneously due to i) happens-before relation such as before and after fork/join, ii) mutual exclusion, iii) lock acquisition pattern [11].

Step 4: (Corresponds to previous scheme [17], denoted as **OLD**). An ordered set of token-passing pairs TP is obtained by considering every pair of control states in $\mathcal{C}_i \times \mathcal{C}_j$, where $\mathcal{C}_i$ and $\mathcal{C}_j$ consist of control states of thread M_i and M_j that have some conflicting access, respectively.

Step 5: (Corresponds to our proposed scheme, denoted as **NEW**). For each thread pairs M_i and M_j, and corresponding set $\mathcal{C}_{ij}$, we identify a set $\mathcal{MAT}_{ij}$ using $GenMAT$. We obtain the set $TP_{ij} = TP(\mathcal{MAT}_{ij})$. Given a set $\mathcal{MAT}_{ij}$, we identify a set eTP_{ij} using $GenExtraTP$. We construct $TP = (\bigcup_{i \neq j} TP_{ij}) \cup (\bigcup_{i \neq j} eTP_{ij})$.

Step 6: We now build token-passing model by first generating decoupled (unrolled) thread models. For each ordered pair $(a, b) \in TP$, we add token passing constraints between (a, b), denoting token may be passed from a to b.

Step 7: Optionally, we add constraints $CB_i^l \leq \texttt{ctk} \leq CB_i^u$ to bound the number of times a token could be passed to a specific thread model M_i, with CB_i^l and CB_i^u corresponding to user-provided lower and upper context-bounds, respectively.

Step 8: We generate verification conditions (discussed in Section 2.1) comprising transition relation of each thread model, token-passing constraints, context-bounding constraints (optionally), and environmental assumptions and negated property constraints. These constraints are expressed in a quantifier-free formula and passed to a SMT/SAT solver for a satisfiability check.

5 Generating MATs

Notation Shortcuts: Before we get into details, we make some notation abuse for ease of readability. When there is no ambiguity, we use e_i to also indicate $c(e_i)$, the control state of thread M_i where the access event e_i belongs. Further, we use $+e_i$ to denote the event immediately after e_i in program order, i.e., $c(+e_i) = next(c(e_i))$. Similarly, we use $-e_i$ to denote event immediately preceding e_i, i.e., $c(e_i) = next(c(-e_i))$. We sometimes refer tuple (a, b) as a pair.

We provide a simple procedure, $GenMAT$ (Algorithm 1) for generating $\mathcal{MAT}_{ij}$, given a pair of unrolled threads M_i and M_j and dependency relation D. For ease of explanation, we assume the threads are unrolled for some bounded depth, and there is no conditional branching. We first initialize a queue Q with control state pair $(\vdash_i, \vdash_j)$ representing the beginning of the threads, respectively. For any pair (f_i, f_j) in the Q, representing the current control pair locations, we can obtain a MAT $m' = (tr'_i, tr'_j)$ as follows: we start tr'_i and tr'_j from f_i and f_j respectively, and end in l'_i and l'_j respectively, such that $(l'_i, l'_j) \in D$, and there is no other conflicting pair in-between. There may be many MAT-candidates m'. Let $\mathcal{M}_c$ denote a set of such choices. The algorithm selects $m \in \mathcal{M}_c$ uniquely by assigning thread priorities and using the following selection rule. If a thread M_j is given higher priority over M_i, the algorithm prefers $m = (tr_i = f_i \cdots l_i, tr_j = f_j \cdots l_j)$ over $m' = (tr'_i = f_i \cdots l'_i, tr'_j = f_i \cdots l'_j)$ if $l_j \prec_{po} l'_j$. Note, the choice of M_j over M_i is arbitrary, but is required for the optimality result. We presented MAT selection (lines 7–9) in a declarative style for better understanding. However, algorithm finds the unique MAT using the selection rule, without constructing the set $\mathcal{M}_c$. We show later that $GenMat$ can always find such a unique MAT with the chosen priority (lines 7—9).

We update $\mathcal{MAT}_{ij}$ with m. If $(l_i \neq \dashv_i)$ and $(l_j \neq \dashv_j)$, we update Q with three pairs, i.e., $(+l_i, +l_j), (+l_i, f_j), (f_i, +l_i)$; otherwise, we insert selectively as shown in the algorithm (lines 11—15).

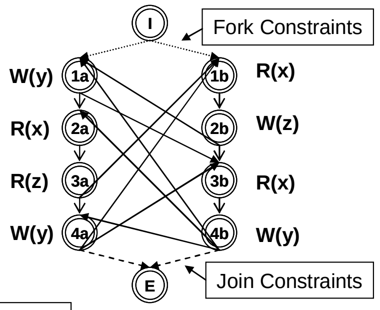
Example: We present a run of $GenMAT$ in Figure 4 for the example in Figure 1(a). We gave M_2 higher priority over M_1. The table columns provide each iteration step (#I), the pair $p \in Q \backslash Q'$ selected, the chosen $\mathcal{MAT}_{12}$, and the new pairs added in $Q \backslash Q'$ (shown in bold). We add token-passing constraints (shown as directed edges) in the figure (on the right) between every ordered pair in the set $TP(\mathcal{MAT}_{12})$. Total number of pair-wise constraints we add is 8, much less compared with all pair-wise constraints (in Figure 1). The fork/join constraints, shown as dotted edges, provide happens-before ordering between the accesses. In the first iteration of the run, out of the two MAT candidates $m = (1a \cdots 3a, 1b \cdots 2b)$ and $m' = (1a, 1b \cdots 4b)$ (also shown in Figure 2(a)-(b)) $GenMAT$ selects m, as M_2 is given higher priority over M_1 and $2b \prec_{po} 4b$.

In the following section, we show the adequacy and optimality of the pair-wise constraints so obtained.

Theorem 1 *The algorithm* $GenMAT$ *terminates.*

Proof. For bounded depth, number of pair-wise accesses are bounded. As each control state pair is picked only once (line 6), the procedure terminates. □.

#I	p∈Q\Q'	$\mathcal{MAT}_{12}$	Q\Q'
			(1a,1b)
1	(1a,1b)	(1a⇒3a,1b ⇒2b)	**(4a,1b)(1a,3b) (4a,3b)**
2	(4a,1b)	(4a,1b⇒4b)	(1a,3b)(4a,3b)
3	(1a,3b)	(1a,3b⇒4b)	(4a,3b)**(2a,3b)**
4	(4a,3b)	(4a,3b⇒4b)	(2a,3b)
5	(2a,3b)	(2a⇒4a,3b ⇒4b)	



Token Passing pair-set (TP($\mathcal{MAT}_{12}$))=
{(2b,1a)(3a,1b)(4a,1b)(4b,4a),(1a,3b)(4b,1a) (4a,3b)(4b,2a)}

#pair-wise constraints = 8

Fig. 4. Run of $GenMAT$ on example in Figure 1(a).

6 MAT-based Reduction: Optimality and Adequacy

For ease of understanding, we first present optimality and adequacy results for a two-threaded system i.e., M_i and M_j with $i, j \in \{1, 2\}$. For two-threaded system, $Sh_{ij} = (Sh_i \cup Sh_j)$, and as noted earlier, $eTP_{ij} = \emptyset$. We ignore it for now; we discuss the general case later as the proof arguments are similar.

Theorem 2 (Two-threaded Optimality) *For two-threaded system with bounded unrolling, the set $TP = TP(\mathcal{MAT}_{ij})$ is optimal i.e., it does not allow two equivalent schedules.*

Algorithm 1 $GenMAT$: Obtain a set of MATs

```
1:  input: Unrolled Thread Models: $M_i, M_j$; Dependency Relation $D$
2:  output: $\mathcal{MAT}_{ij}$.
3:  $\mathcal{MAT}_{ij} := \emptyset$; $Q := \{(\vdash_i, \vdash_j)\}$; $Q' := \emptyset$ {Initialize Queue};
4:  while $Q \neq Q'$ do
5:      Select $(f_i, f_j) \in Q \backslash Q'$
6:      $Q := Q \backslash \{(f_i, f_j)\}$; $Q' := Q' \cup \{(f_i, f_j)\}$
7:      MAT-candidates set, $\mathcal{M}_c = \{m' \mid m' = (tr'_i = f_i \cdots l'_i, tr'_j = f_j \cdots l'_j)\ is\ MAT\}$,
8:      Select a MAT $m = (tr_i = f_i \cdots l_i, tr_i = f_j \cdots l_j) \in \mathcal{M}_c$ such that
9:          $\forall_{m' \in \mathcal{M}_c, m' \neq m}\ l_j \prec_{po} l'_j$, (i.e., $M_j$ has higher priority).
10:     $\mathcal{MAT}_{ij} := \mathcal{MAT}_{ij} \cup \{m\}$
11:     if $(l_i = \dashv_i \wedge l_j = \dashv_j)$ then continue;
12:     elseif $(l_i = \dashv_i)$ then $q := \{(f_i, +l_j)\}$;
13:     elseif $l_j = \dashv_j)$ then $q := \{(+l_i, f_j)\}$;
14:     else $q := \{(+l_i, +l_j), (+l_i, f_j), (f_i, +l_j)\}$;
15:     $Q := Q \cup q$;
16: end while
17: return $\mathcal{MAT}_{ij}$
```

Lemma 1. *If $(a_i, a_j) \in TP(\mathcal{MAT}_{ij})$, then $\exists m = (a'_i \cdots a_i, a_j \cdots a'_j) \in \mathcal{MAT}_{ij}$ where $\vdash_i \preceq_{po} a'_i \preceq_{po} a_i$ and $a_j \preceq_{po} a'_j \preceq_{po} \dashv_j$.*

Lemma 2. *From a given pair $(f_i, f_j) \in Q$, given possible MAT candidates $m_1 = (f_i \cdots e_i, f_j \cdots e_j)$ or $m_2 = (f_i \cdots e'_i, f_j \cdots e'_j)$, $GenMAT$ selects only one of them, i.e., either $m_1 \in \mathcal{MAT}_{ij}$ or $m_2 \in \mathcal{MAT}_{ij}$, but not both. Further, if the thread M_i is given higher priority than M_j, m_1 is selected if $(e_i \prec_{po} e'_i)$, otherwise m_2 is selected.*

Optimality Proof. We show the optimality by arguing the contrapositive holds, i.e., if two schedules allowed by $TP(\mathcal{MAT}_{ij})$ are equivalent, then they are same. We explain our proof steps using the Figure 5(a). Consider two equivalent schedules, i.e., $\sigma_1 \simeq \sigma_2$. We assume that the necessary interleaving pairs for the two schedules be captured by the MAT set, i.e., $CSP(\sigma_1) \subseteq TP(\mathcal{MAT}_{ij})$, and $CSP(\sigma_2) \subseteq TP(\mathcal{MAT}_{ij})$. We show $\sigma_1 = \sigma_2$ by contradiction.

Assume $\sigma_1 \neq \sigma_2$, i.e., $CSP(\sigma_1) \neq CSP(\sigma_2)$. Wlog, let $\sigma_1 = w_i^1 \cdot w_j^1 \cdots w_i^k \cdot w_j^k \cdots w_i^n \cdot w_j^n$ and $\sigma_2 = v_i^1 \cdot v_j^1 \cdots v_i^k \cdot v_j^k \cdots v_i^n \cdot v_j^n$, a sequence of words, $w_i^k, v_i^k \in \Sigma_i^*$, $w_j^k, v_j^k \in \Sigma_j^*$, $1 \leq k \leq n$. (Note, if the words do not align, we pick the schedule with fewer words, say σ_1, and prefix it with empty words corresponding to each thread.) Starting from the end, let the difference first show up at the k^{th} word, i.e., $w_j^k \neq v_j^k$, and $\forall t\ k < t \leq n$, $w_i^t = v_i^t$, $w_j^t = v_j^t$.

Let $w_j^k = f_j^k \cdots l_j^k$ and $v_j^k = f_j^{k'} \cdots l_j^k$. Note, both words end with the same access event because the interleaving pairs matches till that point. Wlog, we assume $f_j^{k'} \prec_{po} f_j^k$. Similarly, we have $w_i^k = f_i^k \cdots l_i^k$, and $v_i^k = f_i^{k'} \cdots l_i^k$. Note, l_i^k immediately precedes (in program order) w_i^{k+1}, i.e., $l_i^k = -w_i^{k+1}[1]$ (Recall, $w[1]$ denotes the first event in word w).

If $k = 1$, we get a trivial contradiction as $w_i^2 = v_i^2$ implies $w_i^1 = v_i^1$. Therefore, we only need to consider $k > 1$. Further, as $w_j^{k+1} = v_j^{k+1}$, we have $|v_j^k| \neq 0$ implies $|w_j^k| \neq 0$ (Note, $|v_j^k| \neq 0$ implies $\vdash_j \preceq_{po} -v_j^{k+1}[1]$, which implies $\vdash_j \preceq_{po} -w_j^{k+1}[1]$, which implies $|w_j^k| \neq 0$). Similarly, as $w_i^{k+1} = v_i^{k+1}$, $|v_i^k| \neq 0$ implies $|w_i^k| \neq 0$. As σ_1 is a schedule prefixed with empty words, $|w_j^k| \neq 0$ implies $|v_j^k| \neq 0$, and $|w_i^k| \neq 0$ implies $|v_i^k| \neq 0$. Thus, we only need to consider $|v_j^k| \neq 0$ where $k > 1$.

Claim 1: $\exists p_j^k\ f_j^{k'} \prec_{po} f_j^k \preceq_{po} p_j^k \preceq_{po} l_j^k$, s.t. $(l_i^k, p_j^k) \in D$.

As $(l_i^k, f_j^{k'}) \in TP(\mathcal{MAT}_{ij})$, $\exists m_1 = (b_i^k \cdots l_i^k, f_j^{k'} \cdots p_j^k) \in \mathcal{MAT}_{ij}$ with $(l_i^k, p_j^k) \in D$, $b_i^k \preceq_{po} l_i^k$ and $f_j^{k'} \preceq_{po} p_j^k$ (as per lemma 1).

Further, $f_j^k \preceq_{po} p_j^k$. If not, i.e., $p_j^k \prec_{po} f_j^k$, then $p_j^k \prec_{\sigma_1} l_i^k$, and $l_i^k \prec_{\sigma_2} p_j^k$. Since, $(l_i^k, p_j^k) \in D$, we get $\sigma_1 \not\simeq \sigma_2$ (contradicting our assumption).

Claim 2: $\forall f\ f_j^{k'} \preceq_{po} f \preceq l_j^{k-1} (= -f_j^k)$, and $\forall e\ f_i^k \preceq_{po} e \preceq_{po} l_i^k$ s.t. $(f, e) \notin D$.

For such f and e, we have $f \prec_{\sigma_1} e$ and $e \prec_{\sigma_2} f$. Since $\sigma_1 \simeq \sigma_2$, the claim $(f, e) \notin D$ follows.

Claim 3: $\exists f'\ l_i^k \prec_{po} f'$ s.t. $(f', l_j^{k-1}) \in D$.

Since $(l_j^{k-1}, f_i^k) \in TP(\mathcal{MAT}_{ij})$, $\exists m_2 = (f_i^k \cdots f', f \cdots l_j^{k-1}) \in \mathcal{MAT}_{ij}$ (as per lemma 1). Thus, $(f', l_j^{k-1}) \in D$. Using claim 2, we have $l_i^k \prec_{po} f'$ as well.

Claim 4: $\neg((l_j^{k-1}, f_i^k) \in TP(\mathcal{MAT}_{ij}) \wedge (l_i^k, f_j^{k'}) \in TP(\mathcal{MAT}_{ij}))$, i.e., $\neg(m_1 \in \mathcal{MAT}_{ij} \wedge m_2 \in \mathcal{MAT}_{ij})$.

Note, $b_i^k, f_i^k \preceq_{po} l_i^k \prec_{po} f'$, and $f, f_j^{k'} \preceq_{po} l_j^{k-1} \prec_{po} p_j^k$. Applying lemma 2, following holds: with M_i higher priority, m_2 will not be chosen as $l_i^k \prec_{po} f'$, and with M_j higher priority, m_1 will not be chosen as $l_j^{k-1} \prec_{po} p_j^k$. Thus, the claim follows. □

Theorem 3 (Two-threaded Adequacy) *For two-threaded system with bounded unrolling, the set $TP = TP(\mathcal{MAT}_{ij})$ is adequate.*

Proof Sketch: Equivalently, we claim that the token-passing constraints added between every pairs in $TP(\mathcal{MAT}_{ij})$ adequately captures all sequentially consistent schedules. In the first step, we construct a procedure *GenEqv* (algorithm 2) to obtain an equivalent schedule $\sigma' \simeq \sigma$, which is also a unique representative of the equivalence class. Second, we use that equivalent schedule σ' to show $CSP(\sigma') \subseteq TP(\mathcal{MAT}_{ij})$.

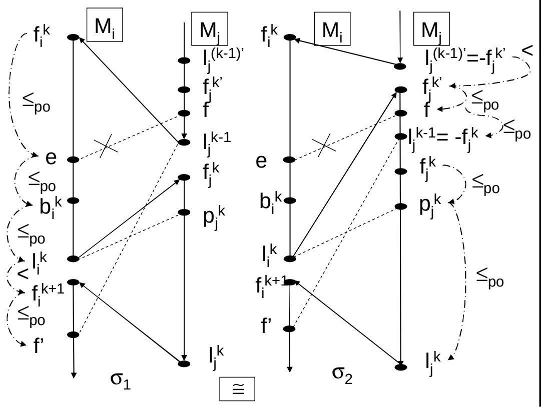
Example: Given a schedule σ shown in Figure 6(a), we obtain equivalent schedules σ' and σ_e by right moving (shown as dotted edges) the last access of a word that does not conflict with the adjacent right word. Note, we use $(1a : W(y))$ to denote a write access on variable y at control state $1a$, and similarly, the rest are denoted.

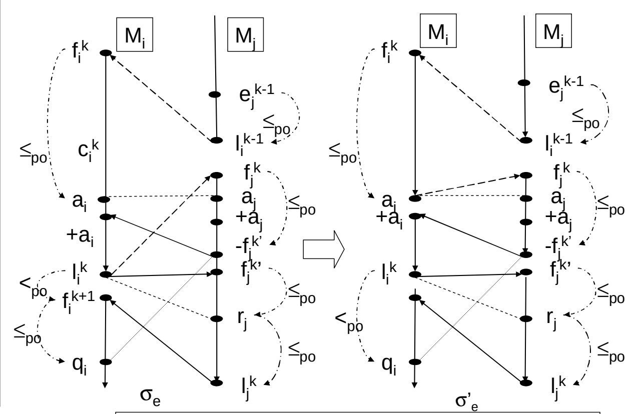
We give the proof details later, but first present required lemmas whose proofs follow from the construction of *GenEqv* and $GenMAT$ (refer Appendix A for details).

Lemma 3. $\sigma_e = GenEqv(\sigma) \simeq \sigma$. *Further, for $\sigma_e = v_i^1 \cdot v_j^1 \cdot \cdot v_i^n \cdot v_j^n$, the last event of a non-empty word is dependent on some access in the right adjacent non-empty word.*

Lemma 4. *For some $k < n$, if $|v_i^k| = 0$, $|v_i^{k+1}| \neq 0$, then for $1 \leq t \leq k$, $|v_i^t| = 0$, and for $k < h \leq n$, $|v_i^h| \neq 0$. In other words, σ_e can have prefix of empty words, but remaining ones are non-empty.*

Lemma 5. *Procedure GenEqv always terminates.*





Optimality Claims
1. $(l_i^k, p_j^k) \in D$
2. $(f, e) \notin D$
3. $(f', l_j^{(k-1)}) \in D$
4. $\neg(l_j^{k-1}, f_i^k) \in TP \wedge (l_i^k, f_i^{k'}) \in TP)$

Conflict pair	┈┈┈
Program order (<,≤)	-·-→
Not established token passing order	- - -→
Established token passing order	──→

Adequacy Claims	
1. $f_j^k <_{po} f_j^{k'}$ 2. $(l_i^k, r_j) \in D$ 3. $(q_i, f) \in D$ 4. $(-f_j^{k'}, f_i^k) \in TP$	Case 1: $\exists (a_i, a_j) \in D$ $\Rightarrow (-f_j^{k'}, +a'_i) \in TP$ Case 2 : $\forall (a_i, a_j) \notin D$

(a) (b)

Fig. 5. (a) Optimality claim: $\sigma_1 \simeq \sigma_2 \Rightarrow \sigma_1 = \sigma_2$ (b) Adequacy claim: CSP(σ_1) $\subseteq TP(\mathcal{MAT}_{ij})$.

Lemma 6. *If $(f_i \cdots e_i, f_j \cdots e_j) \in \mathcal{MAT}_{ij}$, then (1) $(e_i, e_j) \in D$, (2) $(f_i, f_j) \in Q$, (2) if $e_i \neq \dashv_i$, $(+e_i, f_j) \in Q$ (4) if $e_j \neq \dashv_j$, $(f_i, +e_j) \in Q$ (5) if $e_i \neq \dashv_i \wedge (e_j \neq \dashv_j)$, $(+e_i, +e_j) \in Q$ (6) $(e_i, f_j), (e_j, f_i) \in TP(\mathcal{MAT}_{ij})$.*

Lemma 7. *For every pair $(f_i, f_j) \in Q$, there is a MAT candidate, $m = (f_i \cdots e_i, f_j \cdots e_j)$, where $(e_i, e_j) \in D$, and $f_i \preceq_{po} e_i$, and $f_j \preceq_{po} e_j$.*

Lemma 8. *Given a pair $(f_i, f_j) \in Q$, and a reachable pair $(e_i, e_j) \in D$ there exists a MAT $(f'_i \cdots e_i, f'_j \cdots e_j) \in \mathcal{MAT}_{ij}$, where $f_i \preceq_{po} f'_i \preceq_{po} e_i$, $f_j \preceq_{po} f'_j \preceq_{po} e_j$ and $(f'_i, f'_j) \in Q$.*

Lemma 9. *Given a pair $(f_i, f_j) \in Q$, and a reachable pair $(e_i, e_j) \in D$, then (a) $(e_i, f_j) \in TP(\mathcal{MAT}_{ij})$ if M_i is given higher priority. (b) $(e_j, f_i) \in TP(\mathcal{MAT}_{ij})$ if M_j is given higher priority.*

Adequacy Proof. We explain our proof using the Figure 5(b). We first obtain an equivalent schedule $\sigma_e = GenEqv(\sigma)$ such that $\sigma_e = v_i^1 \cdot v_j^1 \cdots v_i^n \cdot v_j^n$. We remove the

Algorithm 2 $GenEqv$: Obtain an equivalent schedule

```
1: input: A sequence of words, $\sigma = w_i^1 \cdot w_j^1 \cdot \cdot w_i^n \cdot w_j^n$, $w_i^k \in \Sigma_i^*$, $w_j^k \in \Sigma_j^*$, $1 \leq k \leq n$.
2: output: $\sigma_e (\simeq \sigma)$.
3: repeat
4:     Do one of the following, A or B
5:     A Right move the last access event $l_i^k$ of the word $w_i^k$ ($k \leq n$) to the beginning of the
       word $w_i^{k+1}$ if $l_i^k$ is either independent of all events in $w_j^k$ or $|w_j^k| = 0$
6:     B Right move the last access event $l_j^k$ of $w_j^k$ ($k < n$) to the beginning of the word $w_j^{k+1}$
       if $l_j^k$ is independent of all events in $w_i^{k+1}$ or $|w_i^{k+1}| = 0$.
7: until neither A nor B occurred
8: return $\sigma_e$
```

prefix empty words, and assume that $|v_i^1| \neq 0$. Using lemma 4, we have $v_i^k \neq 0$, $v_j^k \neq 0$ for $1 \leq k \leq n$. Assume that all interleaving pairs up to k^{th} word are captured in the set $TP(\mathcal{MAT}_{ij})$, i.e., $\forall t\ k < t \leq n$, we have $(l_i^t, f_j^t) \in TP(\mathcal{MAT}_{12})$ and $\forall h$ $k \leq h < n$, we have $(l_j^h, f_i^{h+1}) \in TP(\mathcal{MAT}_{12})$. Let $(l_i^k, f_j^k) \notin TP(\mathcal{MAT}_{ij})$, but $(l_i^k, f_j^{k'}) \in TP(\mathcal{MAT}_{ij})$, $f_j^k \neq f_j^{k'}$.

Claim 1: $f_j^k \prec_{po} f_j^{k'}$. Also, thread M_j is given higher priority over M_i in selecting MAT.

From the lemma 3, we have $(a_i^{k'}, -f_j^k) \in D$, were $f_i^k \preceq_{po} a_i^{k'} \preceq_{po} l_i^k$. From the lemma 6-8, we have $\exists a_i\ a_i \preceq_{po} a_i^{k'}$ s.t. $(a_i, f_j^k) \in Q$ If we give M_i higher priority, we have $(l_i^k, f_j^k) \in TP(\mathcal{MAT}_{ij})$, with $(a_i, f_j^k) \in Q$, and $(l_i^k, r_j^k) \in D$ (using lemma 6, 9). This contradicts our assumption $(l_i^k, f_j^k) \notin TP(\mathcal{MAT}_{ij})$. Thus, we can not give M_i higher priority. Since $f_j^k \neq f_j^{k'}$, the claim $f_j^k \prec_{po} f_j^{k'}$ follows.

Claim 2: $\exists r_j\ \forall a_j\ f_j^k \preceq_{po} a_j \prec_{po} r_j \preceq_{po} l_j^k$ s.t. $(l_i^k, r_j) \in D$ and $(l_i^k, a_j) \notin D$.

From the lemma 3, we clearly have $f_j^k \preceq_{po} r_j$. From claim 1, we have $f_j^k \prec_{po} r_j$.

Claim 3: $\exists f\ f_j^k \preceq_{po} f \prec_{po} f_j^{k'}$, $\exists q_i\ l_i^k \prec_{po} q_i$, s.t. $(q_i, f) \in D$.

If not true, then we move to the $(k-1)^{th}$ word, as the schedules are equivalent up to k^{th} word.

Claim 4: $(-f_j^{k'}, f_i^k) \in TP(\mathcal{MAT}_{ij})$.

We have $f_i^k \prec_{po} q_i$ (from claim 3), and $f_j^k \preceq_{po} -f_j^{k'}$ (from claim 1). Using lemma 3, $\exists e_j^{k-1}\ e_j^{k-1} \prec_{po} f_j^k$ s.t. $(-f_i^k, e_j^{k-1})) \in D$. From $(f_i^k, +e_j^{k-1}) \in Q$ (lemma 6), with M_j higher priority (claim 1), the claim $(-f_j^{k'}, f_i^k) \in TP(\mathcal{MAT}_{ij})$ holds (using lemma 9).

Case Scenario 1: $f_i^k \cdots l_i^k$ conflicts with $f_j^k \cdots - f_j^{k'}$.

Since $(l_i^k, -f_j^{k'}) \notin D$ (claim 2), there $\exists (a_i^k, a_j^k) \in D$, such that $+a_i^k \cdots l_i^k$ does not conflict with $+a_j^k \cdots - f_j^{k'}$, where $f_i^k \prec_{po} a_i^k \prec_{po} l_i^k$ and $f_j^k \preceq_{po} a_j^k \prec_{po} -f_j^{k'}$. Note, we have $(+a_i^k, +a_j^k) \in Q$ (lemma 6). Since $(q_i, -f_j^{k'}) \in D$ (claim 3), and with M_j higher priority (claim 1), $(-f_j^{k'}, +a_i^k) \in TP(\mathcal{MAT}_{ij})$ (lemma 9).

We rearrange the schedule σ_e (as shown in Figure 5(a)) to obtain an equivalent $\sigma_e' = v_i^1 \cdot v_j^1 \cdots v_j^{k-1} \cdot v_i' \cdot v_j' \cdot v_i'' \cdot v_j'' \cdot v_j^{k+1} \cdot v_j^{k+1} \cdots v_i^n \cdot v_j^n$ where $v_i' = f_i^k \cdots a_i^k$ and $v_j' = f_j^k \cdots - f_j^{k'}$, and $v_i'' = +a_i^k \cdots l_i^k$, and $v_j'' = f_j^{k'} \cdots l_j^k$. Note, $(-f_j^{k'}, +a_i^k) \in TP(\mathcal{MAT}_{ij})$ (by above argument) and $(l_i^k, f_j^{k'}) \in TP(\mathcal{MAT}_{ij})$ (by assumption). For the subsequence of σ_e', $v_j' \cdot v_i'' \cdot v_j'' \cdot v_j^{k+1} \cdot v_j^{(k+1)} \cdots v_i^n \cdot v_j^n$, we have established that all the interleaving pairs are captured in the set $TP(\mathcal{MAT}_{ij})$. We then obtain a prefix subsequence $\sigma_{es}' = v_i^1 \cdot v_j^1 \cdots v_j^{k-1} \cdot v_i' \cdot v_j' \cdot v_i''$. Note, the last access of v_i', i.e., may not have conflict with v_j', but v_i' and v_j' are in conflict (case assumption). Therefore, we obtain $\sigma_{ese}' = GenEqv(\sigma_{es}')$, and then reapply our above arguments on σ_e'.

Case Scenario 2: $f_i^k \cdots l_i^k$ does not conflict with $f_j^k \cdots - f_j^{k'}$.

We have $(-f_j^{k'}, f_j^k) \in TP(\mathcal{MAT}_{ij})$ (claim 4) and $(l_i^k, f_j^{k'}) \in TP(\mathcal{MAT}_{ij})$ (by assumption). We rearrange the schedule σ_e to obtain an equivalent $\sigma_e' = v_i^1 \cdot v_j^1 \cdots v_j^{k-1} \cdot v_j' \cdot v_i' \cdot v_i^{k+1} \cdot v_j^{k+1} \cdots v_i^n \cdot v_j^n$ where $v_j' = f_j^k \cdots - f_j^{k'}$, and $v_i' = f_i^k \cdots l_i^k$. We apply our above arguments on the subsequence of $\sigma_{es}' = v_i^1 \cdot v_j^1 \cdots v_j^{k-1}$. $\square$

Example: We show a run of the adequacy proof in Figure 6(b) on a schedule σ shown in Figure 6(a). We first apply procedure *GenEqv* to obtain σ_e. The solid edges show the token passing pairs in $TP(\mathcal{MAT}_{12})$. Starting from left on σ_e, we find that the control state pair $(4b, 2a) \in TP(\mathcal{MAT}_{12})$, but $(1a, 1b) \notin TP(\mathcal{MAT}_{12})$. As $1a : W(y)$ not in conflict with $1b : R(x) \cdots - 4b : W(y)$, we apply the Case Scenario 2, and rearrange the schedule as shown in σ'_e by right moving access $1a : W(y)$ after $2b : W(z)$. Note, $(2b, 1a) \in TP(\mathcal{MAT}_{12})$, as $(2b : W(z), 3a : R(z)) \in D$.

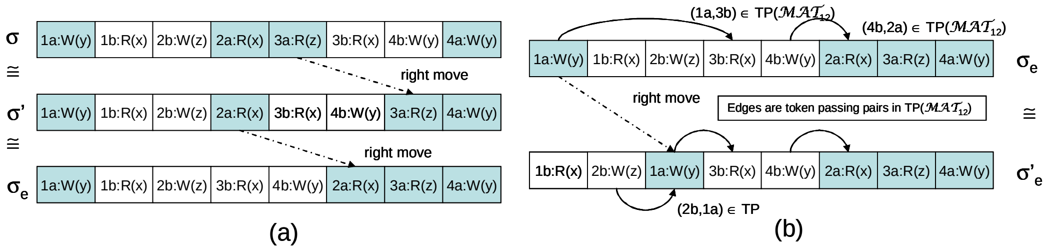


Fig. 6. (a) Equivalent sched. $\sigma, \sigma', \sigma_e$ at each step of GenEqv. (b) Run of adequacy proof.

6.1 Optimality and Adequacy for Multi-threaded System

For a thread pair M_i, M_j if $Sh_{ij} \subsetneq (Sh_i \cup Sh_j)$ holds, then the set $\bigcup_{i \neq j} TP_{ij}$ is not adequate. This can happen for a schedule if a token passes from M_i or M_j to a conflicting access in another thread $k \neq i, j$ on a shared variable $v \in (Sh_i \cup Sh_j) \setminus Sh_{ij}$. We illustrate it with the following example.

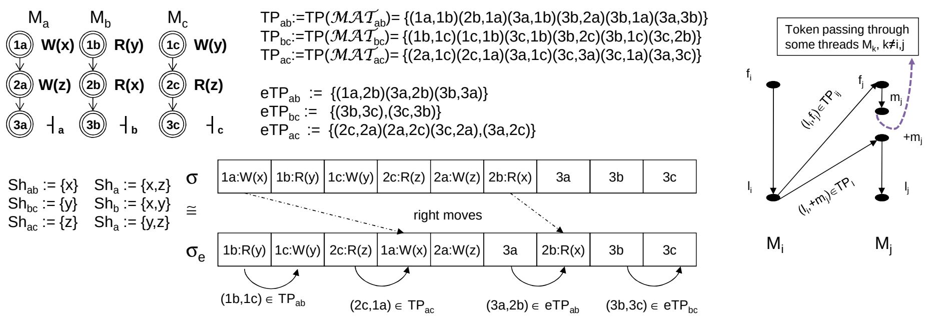
Example: Consider a three-threaded system with threads M_a, M_b and M_c communicating with shared variables x, y, and z as shown in Figure 7(a), and the corresponding pair-wise token-passing sets TP_{ab}, TP_{bc}, and TP_{ac} computed using $GenMAT$ procedure. Consider a schedule σ as shown in the figure. One can obtain an equivalent schedule σ_e by performing right moves. (The procedure $GenEqv$ can be modified to obtain σ_e for general case.) One can verify that the schedule σ_e can not be captured by the computed sets due to missing token-passing pairs such as $(3a, 2b)$. This non-adequacy arise from the following observation: As $y \notin Sh_{ab}$, the procedure $GenMAT$ ignores any interference on such variables by the thread M_c, while considering threads M_a and M_b. Therefore, the token passing pair $(3a, 2b)$ is not added in TP_{ab} while considering the MAT $(2a \Rightarrow 3a, 1b \Rightarrow 3b)$, although $(1b, 1c)$ is added in TP_{bc} as $y \in Sh_{bc}$.

To overcome that scenario, we propose the following construction $GenExtraTP$ that uses $\mathcal{MAT}_{ij}$ to generate eTP_{ij} by adding token-passing pairs for such cases.

$$\begin{aligned} eTP_{ij} = \{(l_i, +m_j), (l_j, +m_i) | (f_i \Rightarrow l_i, f_j \Rightarrow l_j) \in \mathcal{MAT}_{ij}, \\ (f_i \preceq m_i \prec l_i) \wedge \exists_{k \neq j} c_k.(m_i, c_k) \in TP_{ik} \wedge \neg\exists c_j.(m_i, c_j) \in TP_{ij}, \\ (f_j \preceq m_j \prec l_j) \wedge \exists_{k \neq i} c_k.(m_j, c_k) \in TP_{jk} \wedge \neg\exists c_i.(m_j, c_i) \in TP_{ij}\} \end{aligned}$$

For the example, we need additional 9 token-passing pairs with a total of 27 such pairs for adequacy, as compared to 54 (=3*18) in all pair-wise approach [17]. Following result shows that the set is optimal as well.

Theorem 4 (Optimality and Adequacy) *For a multi-threaded system, the set* $(\bigcup_{i \neq j} TP_{ij}) \cup (\bigcup_{i \neq j} eTP_{ij})$ *is both adequate and optimal.*



M_a M_b M_c
1a W(x) 1b R(y) 1c W(y)
2a W(z) 2b R(x) 2c R(z)
3a 3b 3c
TP_{ab}:=TP($\mathcal{MAT}_{ab}$)= {(1a,1b)(2b,1a)(3a,1b)(3b,2a)(3b,1a)(3a,3b)}
TP_{bc}:=TP($\mathcal{MAT}_{bc}$)= {(1b,1c)(1c,1b)(3c,1b)(3b,2c)(3b,1c)(3c,2b)}
TP_{ac}:=TP($\mathcal{MAT}_{ac}$)= {(2a,1c)(2c,1a)(3a,1c)(3c,3a)(3c,1a)(3a,3c)}
eTP_{ab} := {(1a,2b)(3a,2b)(3b,3a)}
eTP_{bc} := {(3b,3c),(3c,3b)}
eTP_{ac} := {(2c,2a)(2a,2c)(3c,2a),(3a,2c)}
Sh_{ab} := {x} Sh_a := {x,z}
Sh_{bc} := {y} Sh_b := {x,y}
Sh_{ac} := {z} Sh_a := {y,z}
σ 1a:W(x) 1b:R(y) 1c:W(y) 2c:R(z) 2a:W(z) 2b:R(x) 3a 3b 3c
≅ right moves
σ_e 1b:R(y) 1c:W(y) 2c:R(z) 1a:W(x) 2a:W(z) 3a 2b:R(x) 3b 3c
(1b,1c) ∈ TP_{ab}
(2c,1a) ∈ TP_{ac}
(3a,2b) ∈ eTP_{ab}
(3b,3c) ∈ eTP_{bc}
Token passing through some threads M_k, k≠i,j
f_i f_j m_j $+m_j$ l_i l_j
$(l_i,f_j)\in TP_{ij}$
$(l_i,+m_j)\in TP_{ij}$
M_i M_j
(a) (b)

Fig. 7. (a) Adequacy for a 3-thread example, (b) Example used in the proof.

Proof sketch: The proof arguments are similar to that used in proving Theorems 2 and 3. We provide a proof sketch here.
Adequacy. Consider $(\bigcup_{i\neq j} eTP_{ij}) = \emptyset$. We claim that for every c_i such that $(m_j, c_i) \in TP_{ij}$, there exists l_i such that $c_i \preceq_{po} l_i$, and $(l_i, +m_j) \in TP_{ij}$.

Consider $(\bigcup_{i\neq j} eTP_{ij}) \neq \emptyset$, i.e., $\exists_{i\neq j} Sh_{ij} \subsetneq (Sh_i \cup Sh_j)$. Consider a MAT $(f_i \Rightarrow l_i, f_j \Rightarrow l_j) \in \mathcal{MAT}_{ij}$ (shown in Figure 7(b)) with $(m_j, c_k) \in TP_{ik}$ for some $k \neq j$ and $f_j \preceq m_j \prec l_j$. By construction of $GenExtraTP$ procedure, $(l_i, +m_j) \in eTP_{ij}$. In other words, if a token leaves at m_j, it comes back from thread M_i at $+m_j$.

We then proceed the proof as follows: Given any schedule σ, we first obtain an equivalent schedule σ_e by right moving the last access until a fix point (similar to $GenEqv$ procedure). Then using the argument similar to proving Theorem 3, we show that $CSP(\sigma_e) \in (\bigcup_{i\neq j} eTP_{ij}) \cup \bigcup_{i\neq j} TP_{ij})$.

Optimality. Given two equivalent schedules σ_1 and σ_2, if $CSP(\sigma_1), CSP(\sigma_2) \subseteq (\bigcup_{i\neq j} TP_{ij})$, we show the optimality by applying Theorem 2 on consecutive words in the schedules. Otherwise, w.l.o.g assume $(l_i, +m_j) \in CSP(\sigma_1)$ with $(l_i, +m_j) \in eTP_{ij}$ (Figure 7(b)). We claim that the token passing path from m_j to l_i (through some other thread(s)) necessarily contains the pair $(m_j, a_k) \in D$ with $m_j \prec a_k$. We then show that $(l_i, +m_j) \in CSP(\sigma_2)$. Thereby, we show that $\sigma_1 = \sigma_2$. □

7 Experiments and Results

We implemented our approach in a token-based modeling framework (similar to [17]) (Figure 3), and used the SMT solver Yices-1.0.13 [26]. We conducted our experiments on a linux box with Intel dual core CPU at 2.0 GHz with 1GB RAM running Ubuntu Linux 8.04, using a 1800 secs time limit. We also integrated context-bounding [23] by bounding the clock vector variable `ctk`. (Recall, such a variable is used to record the number of times the token is exchanged, i.e., number of context-switches).

In our experiments, we automatically checked several three-threaded benchmarks of varied complexity with respect to the number of shared variable accesses. The property constraints correspond to assertion violations. All benchmarks are checked at a depth D equal to the longest path in the program (as it is unrolled). We used standard lockset analysis and inferred happens-before relation from fork/join constraints to reduce the size of $\mathcal{C}_{ij}$.

The details of the benchmarks are shown in Table 1. Columns 1–4 includes the name of benchmarks (Column 1), the number of shared variable accesses in each thread $(\#SA)$ (Column 2), the number of shared variables in the program $(\#SV)$ (Column 3), and the number of transitions in the program $(\#T)$ (Column 4). Each benchmark is suffixed with S or U corresponding to the satisfiable (i.e., has a reachable violation) or unsatisfiable instance. For example, benchmark E3S has a reachable violation with three threads with 1, 20, and 20, number of shared accesses, respectively. Also, E3S benchmark has 2 shared variables, and 51 transitions.

The rest of the columns describes the comparison results. In Column 5 (unrolled cfg), we provide total number of pair-wise constraints$(\#P)$. In the MAT analysis columns (6—7), we provide number of pair-wise constraints after MAT analysis $(\#P_M)$, and number of MAT $(\#M)$. Note, we get significant reduction in $(\#P_M)$. In Columns 8—11, we present the results of token-based approach [17] using P constraints, referred to as basic encoding `B`. In these columns, we provide SMT formula size, time taken (in sec) with no context-bound constraint (NCB), time taken with one context-bound per thread ($C1$), and the witness length (D) (if any), respectively. As the formula sizes for NCB and $C1$ are almost the same, and we do not report them separately. In Columns 12—15, we present similar results for our approach using MAT analysis, denoted as `B+M`, i.e., token-based approach using P_M constraints. In Columns 16—18, we compare our results with the state-of-the-art symbolic approach [16] based on synchronous modeling, referred to as `Ext`, and present similar results. Since `Ext` does not support context-bounding, and it is not clear how to add those constraints efficiently, we do not have any reportable data.

Our approach using MAT (`B+M`) outperforms the basic encoding `B`, and `Ext` in both performance and size of verification conditions by 1–2 orders of magnitude. Encoding using MATs and context bounding (`B+M+C1`) can find the SAT instances very quickly, whereas other encoding *cannot* find it within the time limit. Note, due to synchronous modeling, the witness length D tends to be larger for `Ext`, also noted in [17].

Table 1. Comparing MAT-based reduction with prior approaches

Ex	Program Size			Unrolled cfg	MAT anal.		B [17]				B+M				`Ext` [16]		
(S/U)	#SA	#SV	#T	$\#P$	$\#P_M$	#MAT	Size	NCB	C1	D	Size	NCB	C1	D	Size	NCB	D
E1S	(1,4,4)	2	19	48	23	14	32K	00:0	00:0	11	26K	00:0	00:0	11	174K	00:0	14
E2S	(2,8,8)	3	29	192	8	4	98K	00:1	00:1	15	18K	00:0	00:0	15	497K	00:2	24
E3S	(1,20,20)	2	51	880	591	390	487K	22:1	00:9	27	375K	07:3	00:5	27	1.7M	00:8	46
E4S	(2,40,40)	3	93	3520	200	100	1.9M	1550:5	08:5	47	163K	00:2	00:2	47	6.9M	04:2	88
E5U	(2,40,40)	3	93	3520	200	100	1.9M	TO	13:7	-	163K	50:5	00:2	-	6.9M	99:7	-
E6S	(1,100,100)	2	211	20400	14951	9950	12M	TO	497:8	107	8.7M	TO	150:8	107	51M	MO	206
E7S	(2,200,200)	3	413	81600	5000	2500	48M	TO	TO	207	3.3M	TO	16:4	207	256M	MO	408

SA - Shared Accesses in each thread. **SV** - Shared Variables. **T** - Transitions
B: [17] Basic. **M:** With MAT. **Ext:** [16]. **P:** All pair-wise constraints $\mathbf{P_M}$: P after MAT
NCB: No context bound **C1:** One context bound. **D:** Witness depth. **MO:** Memory out.
TO: Time out. Time is in sec:msec.

8 Conclusion

We are first to exploit partial order reduction techniques in a symbolic model checking effort that generates verification conditions directly without an explicit scheduler. We discussed a novel approach to reduce verification problem sizes and state space for concurrent systems using MATs. We show that our approach gives both adequate and optimal set of token-passing constraints for a bounded unrolling of threads. Our experimental results demonstrates the efficacy of our approach. In future, we would like to exploit transaction-based reductions [11, 19, 20] to further reduce necessary token-passing pairs.

References

1. G. Ramalingam. Context sensitive synchronization sensitive analysis is undecidable. In *ACM Transactions on Programming Languages and Systems*, 2000.
2. A. Valmari. Stubborn sets for reduced state space generation. In *Application and theory of petri nets*, 1989.
3. D. Peled. All from one, one for all: on model checking using representatives. In *Proc. of CAV*, 1993.
4. P. Godefroid. *Partial-order Methods for the Verification of Concurrent Systems: An Approach to the State-explosion Problem*. PhD thesis, 1995.
5. G. Holzmann. The model checker spin. *IEEE Transactions on Software Engineering*, 1997.
6. P. Godefroid. Model checking for programming languages using verisoft. In *Proc. of POPL*, 1997.
7. T. Andrews, S. Qadeer, S. K. Rajamani, J. Rehof, and Y. Xie. ZING: Exploiting program structure for model checking concurrent software. In *Proc. of CONCUR*, 2004.
8. C. Flanagan and P. Godefroid. Dynamic partial-order reduction for model checking software. In *Proc. of POPL*, 2005.
9. G. Gueta, C. Flanagan, E. Yahav, and M. Sagiv. Cartesian partial-order reduction. In *Proc. of SPIN Workshop*, 2007.
10. R. Alur, R. K. Brayton, T. A. Henzinger, S. Qadeer, and S. K. Rajamani. Partial-order reduction in symbolic state space exploration. In *Proc. of CAV*, pages 340–351, 1997.
11. V. Kahlon, A. Gupta, and N. Sinha. Symbolic model checking of concurrent programs using partial orders and on-the-fly transactions. In *Proc. of CAV*, 2006.
12. F. Lerda, N. Sinha, and M. Theobald. Symbolic model checking of software. In *Electronic Notes Theoretical Computer Science*, 2003.
13. B. Cook, D. Kroening, and N. Sharygina. Symbolic Model Checking for Asynchronous Boolean Programs. In *Proc. of SPIN Workshop*, 2005.
14. O. Grumberg, F. Lerda, O. Strichman, and M. Theobald. Proof-guided Underapproximation-Widening for Multi-process Sytems. In *Proc. of POPL*, 2005.
15. I. Rabinovitz and O. Grumberg. Bounded model checking of concurrent programs. In *Proc. of CAV*, 2005.
16. C. Wang, Z. Yang, V. Kahlon, and A. Gupta. Peephole Partial Order Reduction. In *Proc. of TACAS*, 2008.
17. M. K. Ganai and A. Gupta. Efficient modeling of concurrent systems in bmc. In *Proc. of SPIN Workshop*, 2008.
18. E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
19. S. D. Stoller and E. Cohen. Optimistic synchronization-based state-space reduction. In *Proc. of TACAS*, 2003.
20. C. Flanagan and S. Qadeer. Transactions for software model checking. In *Proc. of TACAS*, 2003.
21. V. Levin, R. Palmer, S. Qadeer, and S. K. Rajamani. Sound transaction-based reduction without cycle detection. In *Proc. of SPIN Workshop*, 2003.
22. P. Godefroid and D. Pirottin. Refining dependencies improves partial-order verification methods. In *Proc. of CAV*, 1993.
23. S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *Proc. of TACAS*, 2005.
24. L. Lamport. How to make multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers*, 1979.
25. A. Mazurkiewicz. Trace theory. *Advances in Petric nets*, 1986.
26. SRI. Yices: An SMT solver. *http://fm.csl.sri.com/yices*.

`***The appendix should not be considered as a part of the submission.***`

A Appendix: Proofs

Lemma 1 *If $(a_i, a_j) \in TP(\mathcal{MAT}_{ij})$, then $\exists m = (a'_i \cdots a_i, a_j \cdots a'_j) \in \mathcal{MAT}_{ij}$ where $\vdash_i \preceq_{po} a'_i \preceq_{po} a_i$ and $a_j \preceq_{po} a'_j \preceq_{po} \dashv_j$.*

Proof. Follows from definition of $TP(\mathcal{MAT}_{ij})$. □

Lemma 2 *From a given pair $(f_i, f_j) \in Q$, given possible MAT candidates $m_1 = (f_i \cdots e_i, f_j \cdots e_j)$ or $m_2 = (f_i \cdots e'_i, f_j \cdots e'_j)$, $GenMAT$ selects only one of them, i.e., either $m_1 \in \mathcal{MAT}_{ij}$ or $m_2 \in \mathcal{MAT}_{ij}$, but not both. Further, if the thread M_i is given higher priority than M_j, m_1 is selected if $(e_i \prec_{po} e'_i)$, otherwise m_2 is selected.*

Proof. When m_1 and m_2 are such that $e_i \prec_{po} e'_i$ and $e'_j \prec_{po} e_j$, then $GenMAT$ selects m_1 if M_i has higher priority than M_j, otherwise it selects m_2.

Lemma 3 *$\sigma_e = GenEqv(\sigma) \simeq \sigma$. Further, for $\sigma_e = v_i^1 \cdot v_j^1 \cdot \cdot v_i^n \cdot v_j^n$, the last event of a non-empty word is dependent on some access in the right adjacent non-empty word.*

Proof. Follows from construction of the procedure $GenEqv$. □

Lemma 4 *For some $k < n$, if $|v_i^k| = 0$, $|v_i^{k+1}| \neq 0$, then for $1 \leq t \leq k$, $|v_i^t| = 0$, and for $k < h \leq n$, $|v_i^h| \neq 0$. In other words, σ_e can have prefix of empty words, but remaining ones are non-empty.*

Proof. If for $1 \leq h < n$, $|v_i^h| \neq 0$, then $|v_j^h| \neq 0$; otherwise, we would right move the entire word v_i^h to v_i^{h+1}. If $|v_j^h| \neq 0$, then $|v_i^{h+1}| \neq 0$; otherwise, we would right move the entire word v_j^h to v_j^{h+1}. Similarly, if $|v_i^t| = 0$, $|v_i^{t-1}| = 0$ for $1 \leq t \leq k < n$. □

Lemma 5 *Procedure GenEqv always terminates.*

Proof. For a given schedule σ, the number of different sequences are finite. We only need to show that every eligible right move generates a new and different sequence which corresponds to an equivalent schedule. Let l_i^k be the last access event that is moved from w_i^k ($k < n$) to w_i^{k+1}. After the move, w_i^k changes to $w_i^k[1, |w_i^k| - 1]$, and w_i^{k+1} to $l_i^k \cdot w_i^{k+1}$. Thus, we obtain a different sequence from the previous one. Since, we always make a right move, we can not move the same event e twice from the same word. Thus, we always obtain a new sequence on every move. Also, since each right move respects the dependency (conflict) ordering, the corresponding schedule is equivalent. □

Lemma 6 *If $(f_i \cdots e_i, f_j \cdots e_j) \in \mathcal{MAT}_{ij}$, then (1) $(e_i, e_j) \in D$, (2) $(f_i, f_j) \in Q$, (2) if $e_i \neq \dashv_i$, $(+e_i, f_j) \in Q$ (4) if $e_j \neq \dashv_j$, $(f_i, +e_j) \in Q$ (5) if $e_i \neq \dashv_i \wedge (e_j \neq \dashv_j)$, $(+e_i, +e_j) \in Q$ (6) $(e_i, f_j), (e_j, f_i) \in TP(\mathcal{MAT}_{ij})$.*

Proof. By construction of $GenMAT$ and definition of $TP(\mathcal{MAT}_{ij})$. □

Lemma 7 *For every pair* $(f_i, f_j) \in Q$*, there is a MAT candidate,* $m = (f_i \cdots e_i, f_j \cdots e_j)$*, where* $(e_i, e_j) \in D$*, and* $f_i \preceq_{po} e_i$*, and* $f_j \preceq_{po} e_j$*.*

Proof. For every pair $(f_i, f_j) \in Q$ $(\vdash_i \preceq_{po} f_i \preceq_{po} \dashv_i)$ and $(\vdash_i \preceq_{po} f_i \preceq_{po} \dashv_i$ holds. Also, $(\dashv_i, \dashv_j) \in D$. Thus, the claim holds trivially. □

Lemma 8 *Given a pair* $(f_i, f_j) \in Q$*, and a reachable pair* $(e_i, e_j) \in D$ *there exists a MAT* $(f'_i \cdots e_i, f'_j \cdots e_j) \in \mathcal{MAT}_{ij}$*, where* $f_i \preceq_{po} f'_i \preceq_{po} e_i$*,* $f_j \preceq_{po} f'_j \preceq_{po} e_j$ *and* $(f'_i, f'_j) \in Q$*.*

Proof. Assume M_i is given the higher priority. The argument is similar if M_j is given the higher priority. Let f'_i, f'_j represent a pair reachable from f_i, f_j. If $(f'_i \cdots e_i, f'_j \cdots e_j) \in \mathcal{MAT}_{ij}$, we are done; otherwise $\exists e'_i, e'_j \; f'_i \preceq_{po} e'_i \prec_{po} e_i \; f'_j \preceq_{po} e'_j$ s.t. $(f'_i \cdots e'_i, f'_j \cdots e'_j) \in \mathcal{MAT}_{ij}$. (lemma 2). In that case, we have $(+e'_i, +e'_j), (+e'_i, f'_j) \in Q$, (lemma 6). If $e_j \prec_{po} +e'_j$, we reapply the argument from $(+e'_i, f'_j)$, otherwise from $(+e'_i, +e'_j)$. In both cases, (e_i, e_j) is still reachable. Thus, by applying the argument repeatedly, the claim follows. □.

Lemma 9 *Given a pair* $(f_i, f_j) \in Q$*, and a reachable pair* $(e_i, e_j) \in D$*, then (a)* $(e_i, f_j) \in TP(\mathcal{MAT}_{ij})$ *if* M_i *is given higher priority. (b)* $(e_j, f_i) \in TP(\mathcal{MAT}_{ij})$ *if* M_j *is given higher priority.*

Proof. Consider case (a), as case (b) is a similar. If $(f_i \cdots e_i, f_j \cdots e_j) \in \mathcal{MAT}_{ij}$, we are done; otherwise $\exists e'_i, e'_j \; f_i \preceq_{po} e'_i \prec_{po} e_i \; f_j \preceq_{po} e'_j$ s.t. $(f_i \cdots e'_i, f_j \cdots e'_j) \in \mathcal{MAT}_{ij}$. (lemma 2). In that case, we have $(+e'_i, f_j) \in Q$ (lemma 6). if $(+e'_i = e_i)$, we will have $(e_i, f_j \cdots e'_j) \in \mathcal{MAT}_{ij}$ (lemma 2). The claim follows using lemma 6. □.