Reduction Techniques for Distributed Applications

May 18, 2015

In this section, we explain two different reduction techniques to reduce the state space of distributed multithreaded applications. These techniques are exploited in our work. One technique (Section 1) is a slight modification of a reduction technique presented by Godefroid in [1, Section 2]. This reduction technique by Godefroid reduces the state space of a system consisting of a single process that has multiple threads. These threads are assumed to be deterministic. We apply a similar technique to a system with multiple processes (Section 2). We also show that applying such a technique on the state space of a distributed system preserves deadlocks and assertion violations.

Moreover, we propose a partial order reduction technique for distributed systems (Section 3). Our technique relies on the fact that each process has its own local data which is not shared with any other processes in the system. We show that applying our POR algorithm on a system, which is constructed by the former reduction technique, allows for detecting deadlocks. We use the persistent set technique of Godefroid [2] to prove the correctness of our algorithm. Using this technique, we show that in each state, our algorithm explores a sufficient subset of all possible transitions leading out of the state, preserving deadlocks.

The reduction techniques are presented in general settings. In the last section (Section **??**), we specialize these techniques to JPF. We show that JPF applies the reduction technique of Godefroid [1] for model checking single process Java applications which preserves deadlocks and assertion violations. Moreover, we show that applying our POR algorithm within JPF allows for detecting global deadlocks in distributed Java applications.

1 Reduction of Single Process Systems

Consider a system consisting of a single process composed of a set Φ of threads. The system is modeled by a transition system TS. The transition system is a tuple $(S, Act, Lab, \rightarrow, \beta, s_0)$ such that

- S is a set of states,
- Act is a set of actions,
- $\rightarrow \subseteq S \times Lab \times S$ is the transition relation,
- Lab is a set of transition labels,
- $\beta : S \rightarrow 2^{Lab}$ is the blocking function where,

$$\text{if } \alpha \in \beta(s) \text{ then } \nexists s' \in S \text{ where } (s, \alpha, s') \in \rightarrow, \text{ and}$$

- $s_0 \in S$ is the initial state.

Each thread $T \in \Phi$ executes a sequence of *actions*. Each action captures an operation, and it is uniquely identified by two elements: 1) the thread executing the action, 2) the position of the action in the sequence executed by the thread. We define the function $\tau : \mathit{Act} \to \Phi$ which given an action returns its associated thread. Consider the sequence of actions $a_1 a_2$ executed by T. Let both a_1 and a_2 capture the operation `read x`. Although they capture the same operation, since they take different positions in the sequence, they are referred to as two different actions. Consider the two threads T_1 and T_2 where each executes the sequence of actions ab. The action a to be executed by T_1 is considered to be a different action from a executed by T_2. The same applies to the action b.

The set Act of actions is partitioned into the set V of visible actions, and the set I of invisible actions. The set Lab includes actions that can be associated with transitions, where for the transition system TS, $\mathit{Lab} = \mathit{Act}$.

Instead of $(s, \alpha, s') \in \to$ we write $s \xrightarrow{\alpha} s'$. The action α is said to be *enabled* in the state s, if

$$\exists s' \in S : s \xrightarrow{\alpha} s'.$$

When there is a need to identify the thread T which executes the transition (s, α, s'), we use $s \xrightarrow[T]{\alpha} s'$ in which the thread name is specified as a transition label. Note that by defining the function τ this information is redundant, however, for the sake of readability we use the thread as a transition label.

We use $\mathit{enabled}(s)$ to denote the set of all enabled actions in s, that is,

$$\mathit{enabled}(s) = \{\alpha \in \mathit{Lab} \mid \exists s' \in S : s \xrightarrow{\alpha} s'\}.$$

The action $\alpha \in \mathit{Lab}$ is said to be *blocking* at s, if $\alpha \in \beta(s)$.

The set Φ of threads is assumed to be finite. This is expressed by Assumption 1.1.

Assumption 1.1. *The set Φ is finite.*

Each thread is assumed to be deterministic. At any state s, for each thread T, there is at most one action α and state s', where $s \xrightarrow[T]{\alpha} s'$. This is expressed by Assumption 1.2.

Assumption 1.2. *If $s \xrightarrow[T]{\alpha_1} s_1$ and $s \xrightarrow[T]{\alpha_2} s_2$ then $\alpha_1 = \alpha_2$ and $s_1 = s_2$.*

A state s is called a *global state* if,

$$\mathit{enabled}(s) \subseteq V$$

We denote the set of global states by $g(S)$.

We also assume that all actions to be executed from the initial state are visible. That is expressed by Assumption 1.3.

Assumption 1.3. $\mathit{enabled}(s_0) \subseteq V$

As a consequence, $s_0 \in g(S)$.

We assume that invisible actions of one thread do not have any effect on actions performed by other threads.

Assumption 1.4. *If $s_1 \xrightarrow[T_1]{\alpha_1} s_2 \xrightarrow[T_2]{\alpha_2} s_3$ where $T_1 \neq T_2$ and $\alpha_1 \in I$ or $\alpha_2 \in I$ then $s_1 \xrightarrow[T_2]{\alpha_2} s_2' \xrightarrow[T_1]{\alpha_1} s_3$ for some $s_2' \in S$.*

From the transition system TS, we construct a reduced system, denoted as $r(TS)$. The transition system $r(TS)$ describes the behavior of the system by its set of global states and the transitions (sequence of actions) that take the system from one global state to another. A transition of $r(TS)$ is one visible action followed by a finite maximal sequence of invisible actions performed by a single thread. The transition system $r(TS)$ is the tuple $(g(S), Act, Lab, \Rightarrow, \beta, s_0)$ such that $s \xRightarrow{\alpha_1 \ldots \alpha_n} s'$ if,

- $s = s_1 \xrightarrow{\alpha_1} s_2 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} s_n = s'$,
- $\alpha_1 \in V, \alpha_2 \in I, \ldots, \alpha_n \in I$,
- $s' \in g(S)$, and
- $\tau(\alpha_1) = \tau(\alpha_2) \ldots = \tau(\alpha_n)$.

The set $Lab \subseteq VI^*$ of all possible transitions labels includes the sequences of actions that can be associated with the transitions. The transition $t \in Lab$ where $t = \alpha_1 \ldots \alpha_n$ is said to be *blocking* at the state s, if $\alpha_1 \in \beta(s)$. We use $Blocked(s)$ to denote all the transitions that are blocking in s, that is,

$$Blocked(s) = \{\alpha_1 \ldots \alpha_n \in Lab \mid \alpha_1 \in \beta(s)\}.$$

Let $t \in Blocked(s)$ where $t = \alpha_1\alpha_2 \ldots \alpha_n$ and $\alpha_1 \in \beta(s)$. Towards contradiction suppose that $\exists s'$ where $s \overset{t}{\Rightarrow} s'$. This implies that $s = s_1 \xrightarrow{\alpha_1} s_2 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} s_n = s'$ which contradicts that $\alpha_1 \in \beta(s)$. Thus, it can be concluded that if $t \in Blocked(s)$, then

$$\nexists s' \in S \text{ where } s \overset{t}{\Rightarrow} s'.$$

The example illustrated in Figure 1 compares the transition systems TS and $r(TS)$, composed of two threads T_1 and T_2. The code of T_1 includes the sequence of actions $a_1; a_2$, and the code of T_2 includes the sequence of actions $b_1; b_2;$. The state of the system is composed of three variables (v, i, j) where v is shared, i is local to T_1, and j is local to T_2. The initial state of the system is $(-1, -1, -1)$. The actions a_1 and b_1 are visible actions which set v to 0 and 1, respectively. The actions a_2 and b_2 are invisible where a_2 sets i to 1, and b_2 sets j to 1. Figure 1(a) illustrates the transition system TS, whereas Figure 1(b) illustrates the reduced system $r(TS)$.

The state $s \in S$ is *deadlocked* in TS if,

$$enabled(s) = \emptyset$$

In the system $r(TS)$, $t \in Lab$ is said to be *enabled* in the state $s \in g(S)$, if

$$\exists T \in \Phi : \exists s' \in g(S) : s \underset{T}{\overset{t}{\Rightarrow}} s'$$

We use $Enabled(s)$ to denote the set of all enabled actions in s, that is,

$$Enabled(s) = \{t \in Lab \mid \exists T \in \Phi : \exists s' \in g(S) : s \underset{T}{\overset{t}{\Rightarrow}} s'\}$$

The state $s \in g(S)$ is *deadlocked* in $r(TS)$ if,

$$Enabled(s) = \emptyset$$

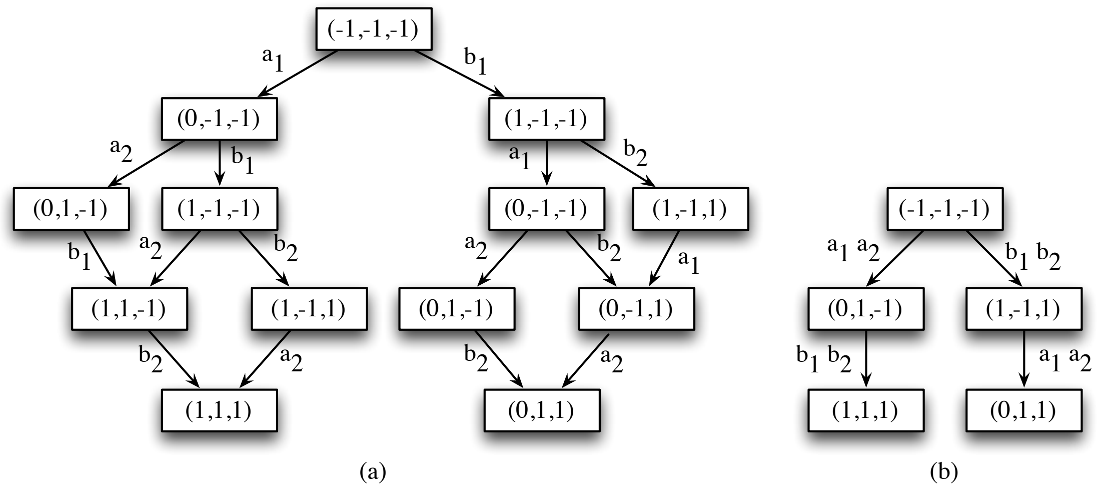

Figure 1: Comparing the transition systems (a) TS and (b) $r(TS)$ for a system composed of two threads with actions $a_1; a_2$ and $b_1; b_2$, where a_1 and b_1 are visible and a_2 and b_2 of the actions are invisible

Note that in our model, states at which the system terminates are also identified as deadlock states. Since these final states can easily be identified, we do not distinguish between final and deadlocked states. This simplifies our model and the proofs that follow, without any loss of generality.

It can be shown that $r(TS)$ preserves deadlocks and assertion violations which are reachable from s_0 in TS. This is captured by the theorems presented next. First, we present Lemma 1.1, which is used to prove these two theorems. This lemma implies that any global state which is reachable from s_0 in TS is also reachable from s_0 in $r(TS)$.

Lemma 1.1. *Let* $s_1 \xrightarrow[T_1]{\alpha_1} s_2 \xrightarrow[T_2]{\alpha_2} \cdots \xrightarrow[T_n]{\alpha_n} s_{n+1}$ *and* $s_1,\ s_{n+1} \in g(S)$. *Let* $\alpha_{r(1)}, \cdots, \alpha_{r(m)}$ *be the subsequence of* $\alpha_1, \cdots, \alpha_n$ *of visible actions. Then* $s_1 = s'_1 \xRightarrow[T_{r(1)}]{t_1} s'_2 \xRightarrow[T_{r(2)}]{t_2} \cdots \xRightarrow[T_{r(m)}]{t_m} s'_{m+1} = s_{n+1}$ *for some* $s'_1, \cdots, s'_{m+1} \in g(S)$ *and* $t_i \in \alpha_{r(i)} I^*$.

Proof. Note that to prove this lemma we use Assumption 1.4. We prove this lemma by induction on the number m of visible actions in $\alpha_1, \ldots, \alpha_n$.

- Base case: $m = 1$. Since $s_1 \in g(S)$, $enabled(s_1) \subseteq V$. As a consequence $\alpha_1 \in V$. Therefore, $\alpha_2, \ldots, \alpha_n \in I$.

  Next, we show that $T_i = T_1$ for all $2 \leq i \leq n$. Towards a contradiction, let j be the smallest index in $[2, n]$ such that $T_j \neq T_1$. Since $\alpha_j \in I$, we can conclude from Assumption 1.4 that $s_1 \xrightarrow[T_j]{\alpha_j} s'_2 \xrightarrow[T_1]{\alpha_1} \cdots \xrightarrow[T_1]{\alpha_{j-1}} s_{j+1}$ for some $s'_2, \ldots, s'_j \in S$. Hence, $\alpha_j \in enabled(s_1)$. But this contradicts that $s_1 \in g(S)$.

  Combining the above, we get that $s_1 \xRightarrow[T_1]{\alpha_1 \ldots \alpha_n} s_{n+1}$.

- Inductive step: let $m > 1$. Since $\alpha_{r(1)}$ is the first visible action, as in the base case, $\alpha_1 = \alpha_{r(1)}$. Note that $\alpha_{r(2)}$ is the second visible action in $\alpha_1, \ldots, \alpha_n$. Let $r(2) = k$. As in the base case, we can show that $T_i = T_1$ for all $2 \leq i < k$. Let ℓ be the smallest index in the interval $[k, n+1]$ such that either $\mathit{enabled}(s_\ell) = \emptyset$ (in which case $\ell = n+1$) or $s_\ell \xrightarrow[T_1]{\alpha} s$ for some $\alpha \in V$ and $s \in S$. Note that such an ℓ exists, since $s_{n+1} \in g(S)$.

Let i be the number of invisible actions performed by T_1 in $\alpha_{k+1}, \ldots, \alpha_{\ell-1}$. Let $\alpha_{f(1)}, \ldots, \alpha_{f(i)}$ be the subsequence of $\alpha_{k+1}, \ldots, \alpha_{\ell-1}$ of invisible actions performed by T_1. Let $\alpha_{g(1)}, \ldots, \alpha_{g(\ell-k-i-1)}$ be the subsequence of $\alpha_{k+1}, \ldots, \alpha_{\ell-1}$ of the remaining actions. We will prove that

$$s_k \xrightarrow[T_1]{\alpha_{f(1)}} \cdots \xrightarrow[T_1]{\alpha_{f(i)}} s'_{k+i} \xrightarrow[T_k]{\alpha_k} s'_{k+i+1} \xrightarrow[T_{g(1)}]{\alpha_{g(1)}} \cdots \xrightarrow[T_{g(\ell-k-i-1)}]{\alpha_{g(\ell-k-i-1)}} s_\ell \tag{1}$$

for some $s'_{k+1}, \ldots, s'_{\ell-1} \in S$ by induction on i.

- The base case, $i = 0$, is trivial.
- Let $i > 0$. Using Assumption 1.4, we can conclude

$$s_k \xrightarrow[T_1]{\alpha_{f(1)}} s'_{k+1} \xrightarrow[T_k]{\alpha_k} \cdots \xrightarrow[T_{f(1)-1}]{\alpha_{f(1)-1}} \cdots \xrightarrow[T_{f(1)+1}]{\alpha_{f(1)+1}} \cdots \xrightarrow[T_{\ell-1}]{\alpha_{\ell-1}} s_\ell$$

  for some $s'_{k+1}, \ldots, s'_{\ell-1} \in S$. By induction,

$$s'_{k+1} \xrightarrow[T_1]{\alpha_{f(2)}} \cdots \xrightarrow[T_1]{\alpha_{f(i)}} s'_{k+i} \xrightarrow[T_k]{\alpha_k} s'_{k+i+1} \xrightarrow[T_{g(1)}]{\alpha_{g(1)}} \cdots \xrightarrow[T_{g(\ell-k-i-1)}]{\alpha_{g(\ell-k-i-1)}} s_\ell$$

  for some $s'_{k+2}, \ldots, s'_{\ell-1} \in S$. Hence, (1) immediately follows.

Next, we will show that $s'_{k+i} \in g(S)$ by showing that $\mathit{enabled}(s'_{k+i}) \subseteq V$. From the choice of ℓ and the construction of the subsequence $\alpha_{f(1)}, \ldots, \alpha_{f(i)}$ we can conclude that if $s'_{k+i} \xrightarrow[T]{\alpha} s$ where $T = T_1$ then $\alpha \in V$. Let $T \neq T_1$. Towards a contradiction, assume that $s'_{k+i} \xrightarrow[T]{\alpha} s$ for some $s \in S$ and $\alpha \in I$. Again using Assumption 1.4, we can conclude that $s_1 \xrightarrow[T]{\alpha} s'$ for some $s' \in S$. This contradicts that $s_1 \in g(S)$. Therefore $s'_{k+i} \in g(S)$.

From the above we can conclude that $s_1 \overset{t_1}{\underset{T_{r(1)}}{\Longrightarrow}} s'_{k+i}$ for some $s'_{k+i} \in g(S)$ and $t_1 \in \alpha_{r(1)} I^*$. By induction, $s'_{k+i} = s'_2 \overset{t_2}{\underset{T_{r(2)}}{\Longrightarrow}} s'_3 \overset{t_3}{\underset{T_{r(3)}}{\Longrightarrow}} \cdots \overset{t_m}{\underset{T_{r(m)}}{\Longrightarrow}} s'_m = s_{n+1}$ for some $s'_2, \cdots, s'_m \in g(S)$ and $t_j \in \alpha_{r(j)} I^*$ for $1 < j \leq m$. Hence, $s_1 = s'_1 \overset{t_1}{\underset{T_{r(1)}}{\Longrightarrow}} s'_2 \overset{t_2}{\underset{T_{r(2)}}{\Longrightarrow}} s'_3 \overset{t_3}{\underset{T_{r(3)}}{\Longrightarrow}} \cdots \overset{t_m}{\underset{T_{r(m)}}{\Longrightarrow}} s'_m = s_{n+1}$ for some $s'_1, \cdots, s'_m \in g(S)$ and $t_j \in \alpha_{r(j)} I^*$ for $1 \leq j \leq m$.

□

Corollary 1.1 can be directly derived from Lemma 1.1.

Corollary 1.1. *If* $s_1 \xrightarrow[T_1]{\alpha_1} s_2 \xrightarrow[T_2]{\alpha_2} \cdots \xrightarrow[T_n]{\alpha_n} s_{n+1}$ *and* s_1, $s_{n+1} \in g(S)$ *then* $s_1 \Rightarrow^* s_{n+1}$.

Theorem 1.1. *All deadlocked states which are reachable from* s_0 *in* TS *are also deadlocked states, reachable from* s_0*, in* $r(TS)$*.*

Proof. Note that to prove this theorem we use Assumption 1.4 and 1.3. Consider that the state s reachable from s_0 in TS is deadlocked. According to the definition of a deadlocked state, we have $\mathit{enabled}(s) = \emptyset$. Therefore, $\mathit{enabled}(s) \subseteq V$ which implies that s is a global state, that is, $s \in g(S)$. According to Corollary 1.1 and Assumption 1.3, since s is reachable from s_0 in TS, it is also reachable from s_0 in $r(TS)$. Since $\mathit{enabled}(s) = \emptyset$, $\mathit{Enabled}(s) = \emptyset$. Therefore, s is a deadlocked state in $r(TS)$. □

Theorem 1.2 captures the converse case. To prove this theorem, we need to assume that each thread cannot do a sequence of infinitely many invisible actions.

Assumption 1.5. *If $s_i \xrightarrow[T]{\alpha_i} s_{i+1}$ for all $i \in \mathbb{N}$ then for all $n \in \mathbb{N}$ there exists $m > n$ such that $\alpha_m \in V$.*

The following lemma is also needed to prove Theorem 1.2. As we will show next, from any state we can reach a global state by doing only invisible actions.

Lemma 1.2. *For all $s_1 \in S$ there exists $s_1 \xrightarrow[T_1]{\alpha_1} s_2 \xrightarrow[T_2]{\alpha_2} \cdots \xrightarrow[T_n]{\alpha_n} s_{n+1}$ where $\alpha_1, \ldots, \alpha_n \in I$ and $s_{n+1} \in g(S)$.*

Proof. Note that to prove this lemma we use Assumption 1.1, 1.4, and 1.5. Consider the following algorithm.

```
$s \leftarrow s_1$
**while** $s \notin g(S)$ **do**
  pick $\alpha \in I$, $s' \in S$ and $T \in \Phi$ such that $s \xrightarrow[T]{\alpha} s'$
  $s \leftarrow s'$
**end while**
```

Note that if $s \notin g(S)$, then $\mathit{enabled}(s) \not\subseteq V$. Hence, we can find $\alpha \in I$, $s' \in S$ and $T \in \Phi$ such that $s \xrightarrow[T]{\alpha} s'$. If the algorithm terminates, then the lemma obviously holds.

Towards a contradiction, assume that the algorithm does not terminate. Then for all $i \in \mathbb{N}$, there exist $s_i \in S$, $\alpha_i \in I$ and $T_i \in \Phi$ such that $s_i \xrightarrow[T_i]{\alpha_i} s_{i+1}$. Since the set Φ is finite, one thread, say T, takes infinitely many transitions. Let $(T_{f(i)})_{i \in \mathbb{N}}$ be the subsequence of the transitions taken by thread T. Let $(T_{g_n(i)})_{i \in \mathbb{N}}$ be the subsequence obtained by removing $T_{f(1)}, \ldots, T_{f(n-1)}$ from $(T_i)_{i \in \mathbb{N}}$.

Next, we show that for each $n \in \mathbb{N}$ there exists a sequence $(s_{n,i})_{i \in \mathbb{N}}$ such that

- $s_{n,1} = s_1$,
- for all $1 \leq i < n$, $s_{n,i} \xrightarrow[T_{f(i)}]{\alpha_{f(i)}} s_{n,i+1}$
- for all $i \geq n$, $s_{n,i} \xrightarrow[T_{g_n(i-n+1)}]{\alpha_{g_n(i-n+1)}} s_{n,i+1}$

by induction on n. Note that this contradicts Assumption 1.5.

- Base case: $n = 1$. We simply take $s_{1,i} = s_i$ for all $i \in \mathbb{N}$.
- Inductive step: let $n > 1$. We define for all $1 \leq i \leq n-1$, $s_{n,i} = s_{n-1,i}$. Let j be such that $s_{n-1,j} \xrightarrow[T_{f(n-1)}]{\alpha_{f(n-1)}} s_{n-1,j+1}$. Then for all $n-1 \leq i < j$, $s_{n-1,i} \xrightarrow[T_{g_{n-1}(i-n+2)}]{\alpha_{g_{n-1}(i-n+2)}} s_{n-1,i+1}$

and $T_{g_{n-1}(i-n+2)} \neq T$. Hence, according to Assumption 1.4, there exist $s_{n,i}$, for $n \leq i \leq j$, such that $s_{n-1,n-1} \xrightarrow[T_{f(n-1)}]{\alpha_{f(n-1)}} s_{n,n}$ and $s_{n,i} \xrightarrow[T_{g_{n-1}(i-n+1)}]{\alpha_{g_{n-1}(i-n+1)}} s_{n,i+1}$ for all $n \leq i \leq j$ and $s_{n,j} \xrightarrow[T_{g_{n-1}(j-n+1)}]{\alpha_{g_{n-1}(j-n+1)}} s_{n-1,j+1}$. For all $i > j$, define $s_{n,i} = s_{n-1,i}$. The sequence $(s_{n,i})_{i \in \mathbb{N}}$ satisfies the properties by construction.

□

Theorem 1.2. *All deadlocked states which are reachable from s_0 in $r(TS)$ are also deadlocked states, reachable from s_0, in TS*

Proof. To prove this theorem, we need Assumption 1.1, 1.4, and 1.5. Suppose that $s \in g(S)$ is a deadlocked state reachable from s_0 in $r(TS)$. It is clear that s is reachable from s_0 in TS through the same sequence of actions. Now we need to show that s is also a deadlocked state in TS.

Since s is a deadlocked state in $r(TS)$, $\mathit{Enabled}(s) = \emptyset$. Towards contradiction, assume that s is not a deadlocked state in TS which implies that $\mathit{enabled}(s) \neq \emptyset$. Hence, $s \xrightarrow[T_0]{\alpha_0} s_1$ for some $T_0 \in \Phi$, $\alpha_0 \in \mathit{Lab}$, and $s_1 \in S$. According to Lemma 1.2, there exists

$$s_1 \xrightarrow{\alpha_1} s_2 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_{n-1}} s_n \text{ where } \alpha_1, \ldots, \alpha_{n-1} \in I \text{ and } s_n \in g(S).$$

Moreover, according to Corollary 1.1, since $s, s_n \in g(S)$, and there exists a path from s to s_n, then $s \Rightarrow^* s_n$. However, this contradicts that s is a deadlocked state in $r(TS)$. Thus $\mathit{enabled}(s) = \emptyset$. □

Consider the system TS which can only perform the following sequence of transitions

$$s_0 \xrightarrow[T_1]{a_1} s_1 \xrightarrow[T_2]{a_2} s_2 \xrightarrow[T_1]{b_1} s_3 \xrightarrow[T_2]{b_2} s_4$$

where $a_1, a_2 \in V$ and $b_1, b_2 \in I$. Note that Assumption 1.4 is not satisfied. The following sequence of transitions in $r(TS)$ reaches a deadlock.

$$s_0 \overset{a_1}{\underset{T_1}{\Longrightarrow}} s_1 \overset{a_2}{\underset{T_2}{\Longrightarrow}} s_2 \not\Rightarrow$$

Hence, without Assumption 1.4, Theorem 1.1 may not hold.

Consider the system TS which can only perform the following sequence of transitions

$$s_0 \xrightarrow[T_1]{a_1} s_1 \xrightarrow[T_2]{a_2} s_1$$

where $a_1 \in V$ and $a_2 \in I$. Note that Assumption 1.5 is not satisfied. Since $\mathit{Enabled}(s_0) = \emptyset$, the state s_0 is deadlocked in $r(TS)$. However, s_0 is not a deadlock in TS, since $\mathit{enabled}(s_0) = \{a_1\}$. Hence, without Assumption 1.5, Theorem 1.1 may not hold.

In our model, an assertion is defined as an action $\mathit{assert}(A)$, where $A \subseteq S$. The set A consists of those states in which the assertion holds. In TS, the assertion A is said to be violated in $s \in S$, if

$$\mathit{assert}(A) \in \mathit{enabled}(s) \wedge s \notin A$$

In $r(TS)$, the assertion A is said to be violated in $s \in g(S)$, if there exists $t = \mathit{assert}(A); a_1; \ldots; a_n$, $t \in \mathit{Lab}$ such that

$$t \in \mathit{Enabled}(s) \wedge s \notin A$$

In our model, it is assumed that the action $assert(A)$ is visible which is expressed by Assumption 1.6.

Assumption 1.6. $assert(A) \in V$

Moreover, we assume that invisible actions cannot affect assertion results. This is expressed by Assumption 1.7

Assumption 1.7. *If $s \xrightarrow[T]{\alpha} s'$ where $\alpha \in I$ and $assert(A) \in enabled(s)$ then $assert(A) \in enabled(s')$ and $s \in A$ iff $s' \in A$*

The $assert(A)$ action can be thought of as an evaluation of variables whose values cannot be changed by invisible actions.

Theorem 1.3. *If there is an assertion, A, violated in a state reachable from s_0 in TS, then there is a state reachable from s_0 in $r(TS)$ at which A is violated.*

Proof. Note that to prove this lemma we use Assumption 1.1, 1.4, 1.6, 1.7, and 1.5. Suppose that s is a reachable state from s_0 in TS where an assertion A is violated. Then there is a thread $T \in \Phi$ where $s \xrightarrow[T]{assert(A)} s'$ for $s \notin A$ and some $s' \in S$. From Lemma 1.2 we can conclude that

$$s = s_1 \xrightarrow[T_1]{\alpha_1} s_2 \xrightarrow[T_2]{\alpha_2} \cdots \xrightarrow[T_n]{\alpha_n} s_{n+1} \tag{2}$$

where $\alpha_1, \ldots, \alpha_n \in I$ and $s_{n+1} \in g(S)$. From Assumption 1.7, since $assert(A) \in enabled(s_1)$ then $assert(A) \in enabled(s_{n+1})$ and since $s \notin A$ then $s_{n+1} \notin A$. From Corollary 1.1, since $s_{n+1} \in g(S)$, we can conclude that s_{n+1}, at which A is violated, is reachable from s_0 in $r(TS)$. □

The converse is true as well. Suppose that $s \in g(S)$ is a state reachable from s_0 in $r(TS)$, at which A is violated. The state s is reachable from s_0 in TS through the same sequence of actions. Since A is violated at s, $s \notin A$ and $t \in Enabled(s)$ for some $t = assert(A); a_1; \ldots; a_n$. Therefore, $assert(A) \in enabled(s)$. Since $s \notin A$, the assertion A is also violated in the state s of TS.

Consider the system TS which can perform the following sequence of transitions.

$$s_0 \xrightarrow[T_1]{a_1} s_1 \xrightarrow[T_1]{assert(\emptyset)} s_2$$

Since $s_1 \notin \emptyset$, the assertion is violated at s_1. Assume that $a_1 \in V$ and $assert(\emptyset) \in I$. Note that Assumption 1.6 is violated. Then the system $r(TS)$ includes the following transition.

$$s_0 \overset{a_1;assert(\emptyset)}{\underset{T_1}{\Longrightarrow}} s_2$$

The only state at which the assertion is violated is s_1, however, $s_1 \notin g(S)$. Hence without Assumption 1.6, Theorem 1.3 may not hold.

Consider the system TS that performs the sequence of transitions

$$s_0 \xrightarrow[T_1]{a_1 \in V} s_1 \xrightarrow[T_1]{a_2 \in I} s_2$$

where $a_1 \in V$ and $a_2 \in I$. Assume that

$$assert(\emptyset) \notin enabled(s_0) \wedge assert(\emptyset) \in enabled(s_1) \wedge assert(\emptyset) \notin enabled(s_2)$$

Note that Assumption 1.7 is violated. The system $r(TS)$ includes the following transition.

$$s_0 \underset{T_1}{\overset{a_1;a_2}{\Longrightarrow}} s_2$$

An assertion is violated at s_1 and there is no state of $r(TS)$ at which the assertion is violated. Hence without Assumption 1.7, Theorem 1.3 may not hold.

Consider the system TS that performs the infinite sequence of transitions

$$s_0 \xrightarrow[T_1]{assert(\emptyset)} s_1 \xrightarrow[T_1]{a_1} s_2 \xrightarrow[T_1]{a_2} \dots$$

where $a_i \in I$. Note that Assumption 1.5 is violated. Since this sequence never reaches a global state, it does not belong to the system $r(TS)$. That implies $Enabled(s_0) = \emptyset$. Therefore, the assertion A is not violated in $r(TS)$. Hence, without Assumption 1.5, Theorem 1.3 may not hold.

2 Reduction of Distributed Systems

Consider a distributed system composed of a set of (multithreaded) processes. Let the set Φ of threads include all threads in the system regardless of which process they belong to. The behavior of such a distributed system can be captured by a single process system composed of the set Φ of threads which can be modeled by the transition system TS. Therefore, we can use the reduced system $r(TS)$ to model the distributed system which allows for detecting deadlocks and assertion violations. Note that the distributed system is required to satisfy Assumption 1.2-1.5, otherwise there is no guarantee that deadlocks and assertion violations are preserved by $r(TS)$.

Let P denote a set of (multithreaded) processes. The set P of processes is assumed to be finite. This is expressed by Assumption 2.1.

Assumption 2.1. *The set P is finite.*

We use Φ_p to denote the finite set of threads that belong to the process $p \in P$. These sets are disjoint. That is expressed by Assumption 2.2.

Assumption 2.2. *If $p_1 \neq p_2$ then $\Phi_{p_1} \cap \Phi_{p_2} = \emptyset$*

When it comes to distributed systems, two different types of deadlocked states can be distinguished: *globally deadlocked states* and *locally deadlocked states*. The definition for deadlocked states, given in Section 1, represents globally deadlocked states. In globally deadlocked states, the entire system is prevented from progressing, whereas in locally deadlocked states one process is prevented from progressing. Next we formalize the definition of locally deadlocked states.

We use $enabled(s, p)$ to denote the set of all enabled actions in $s \in S$ which belong to the process $p \in P$, that is,

$$enabled(s, p) = \{\alpha \in lab \mid \exists T \in \Phi_p : \exists s' \in S : s \xrightarrow[T]{\alpha} s'\}$$

We use a definition similar to $enabled(s, p)$ for the set $Enabled(s, p)$, which denotes the set of all enabled transitions in $s \in g(S)$ which belong to the process $p \in P$, except $Enabled(s, p)$ is defined over the relation $\Rightarrow$ instead of $\rightarrow$.

$$Enabled(s, p) = \{t \in Lab \mid \exists T \in \Phi_p : s' \in g(S) : s \underset{T}{\overset{t}{\Rightarrow}} s'\}$$

A state $s' \in S$ is said to be *reachable* from s if there exists a finite execution fragment

$$s = s_1 \xrightarrow[T_1]{\alpha_1} s_2 \xrightarrow[T_2]{\alpha_2} \cdots \xrightarrow[T_n]{\alpha_n} s_n = s'.$$

Let $\mathit{reach}(s)$ denote the set of all states reachable from s in TS which also includes s. That is, $\mathit{reach}(s)$ is the smallest set $R \subseteq S$ such that

- $s \in R$, and
- if $s' \in R$ and $s' \xrightarrow[T]{\alpha} s''$ for some $\alpha \in \mathit{lab}$ and $T \in \Phi$ then $s'' \in R$.

For TS, we say that the process $p \in P$ is deadlocked in $s \in S$ if,

$$\forall s' \in \mathit{reach}(s) : \mathit{enabled}(s', p) = \emptyset.$$

The state s represents a locally deadlocked state. Once s is reached, the threads of process p cannot execute any actions, regardless of the executions of other processes in the system.

We use a definition similar to $\mathit{reach}(s)$ for the set $\mathit{Reach}(s)$, which denotes the set of all states reachable from s in $r(TS)$, except $\mathit{Reach}(s)$ is defined over the relation $\Rightarrow$ instead of $\rightarrow$. For $r(TS)$, we say that the process $p \in P$ is deadlocked in $s \in g(S)$ if,

$$\forall s' \in \mathit{Reach}(s) : \mathit{Enabled}(s', p) = \emptyset.$$

Theorem 2.1. *For any state reachable from s_0 in TS in which the process p is deadlocked, there exists a state in $r(TS)$ reachable from s_0 in which p is deadlocked.*

Proof. Note that to prove this theorem we use Assumption 1.4, 1.5, and 1.1. Suppose that the process p is deadlocked in the state $s_n \in S$ which is reachable from s_0 in TS. This implies

$$s_0 \xrightarrow[T_1]{\alpha_1} s_1 \xrightarrow[T_2]{\alpha_2} s_2 \cdots \xrightarrow[T_n]{\alpha_n} s_n$$

where for each state $s \in \mathit{reach}(s_n)$, $\mathit{enabled}(s, p) = \emptyset$. According to Lemma 1.2, there exists a state, $s_{n+m} \in g(S)$ reachable from s_n, that is

$$s_0 \xrightarrow[T_1]{\alpha_1} s_1 \xrightarrow[T_2]{\alpha_2} s_2 \cdots \xrightarrow[T_n]{\alpha_n} s_n \xrightarrow[T_{n+1}]{\alpha_{n+1}} s_{n+1} \cdots \xrightarrow[T_{n+m}]{\alpha_{n+m}} s_{n+m}$$

Any state s which is reachable from s_{n+m} in TS where $s_{n+m} \xrightarrow[T_{n+m}]{\alpha_{n+m}} \cdots \xrightarrow[T_{n+m+1}]{\alpha_{n+m+l}} s_{n+m+l} = s$ is also reachable from s_n through the path

$$s_n \xrightarrow[T_n]{\alpha_n} s_n \xrightarrow[T_{n+1}]{\alpha_{n+1}} s_{n+1} \cdots \xrightarrow[T_{n+m}]{\alpha_{n+m}} s_{n+m} \cdots \xrightarrow[T_{n+m+1}]{\alpha_{n+m+l}} s_{n+m+l} = s.$$

This implies,

$$\mathit{reach}(s_{n+m}) \subseteq \mathit{reach}(s_n) \tag{3}$$

Since $s_{n+m} \in g(S)$, according to Corollary 1.1, $s_0 \Rightarrow^* s_{n+m}$. It is obvious that for any state s reachable from s_{n+m} in $r(TS)$, there exists a path (consisting of the same sequence of actions) from s_{n+m} which reaches s in TS. That implies,

$$\mathit{Reach}(s_{n+m}) \subseteq \mathit{reach}(s_{n+m}) \tag{4}$$

From (3) and (4), it can be concluded that $\mathit{Reach}(s_{n+m}) \subseteq \mathit{reach}(s_n)$. Therefore for all $s \in \mathit{Reach}(s_{n+m})$ we have $\mathit{enabled}(s, p) = \emptyset$ which also implies $\mathit{Enabled}(s, p) = \emptyset$. Thus s_{n+m} is a state of $r(TS)$ in which p is deadlocked. □

Thus we have shown that for any state s in TS in which p is deadlocked, there is a state in $r(TS)$ in which p is deadlocked. Now we show the reverse of Theorem 2.1 which is captured by Theorem 2.2.

Theorem 2.2. *For any state reachable from s_0 in $r(TS)$ in which the process p is deadlocked, there exists a state in TS reachable from s_0 in which p is deadlocked.*

Proof. Note that to prove this theorem we use Assumption 1.1, 1.4, and 1.5. Suppose that the process p is deadlocked in the state $s \in g(S)$. This implies that for each state $s' \in Reach(s)$, $Enabled(s', p) = \emptyset$.

Towards contradiction suppose that in TS, p is not deadlocked in s. Therefore, there exists a state s_n where $s_n \in reach(s)$ and $enabled(s_n, p) \neq \emptyset$, that is,

$$s = s_1 \xrightarrow[T_1]{\alpha_1} s_2 \cdots \xrightarrow[T_{n-1}]{\alpha_{n-1}} s_n \xrightarrow[T_n]{\alpha_n} s_{n+1} \text{ where } T_n \in \Phi_p$$

Without loss of generality, assume that s_n is a first such state in the execution fragment $s_1 \rightarrow^* s_n$. As a consequence, $T_i \neq T_n$ for all $1 \leq i < n$. One possibility is that $\alpha_n \in I$. According to Assumption 1.4, this implies that $\alpha_n \in enabled(s)$. But this contradicts that $s \in g(S)$. Another possibility is that $\alpha_n \in V$. According to Lemma 1.2, there exists

$$s_{n+1} \xrightarrow[T_{n+1}]{\alpha_{n+1}} s_{n+2} \xrightarrow[T_{n+2}]{\alpha_{n+2}} \cdots \xrightarrow[T_{n+m-1}]{\alpha_{n+m-1}} s_{n+m} \text{ where } \alpha_{n+1}, \ldots, \alpha_{n+m-1} \in I \text{ and } s_{n+m} \in g(S).$$

Therefore, there exits the fragment $s = s_1 \xrightarrow[T_1]{\alpha_1} s_2 \cdots s_n \xrightarrow[T_n]{\alpha_n} s_{n+1} \xrightarrow[T_{n+1}]{\alpha_{n+1}} \cdots \xrightarrow[T_{n+m-1}]{\alpha_{n+m-1}} s_{n+m}$. Let $\alpha_{r(1)}, \cdots, \alpha_{r(k)}$ be the subsequence of $\alpha_1, \cdots, \alpha_{n+m-1}$ of visible actions and $\alpha_n = \alpha_{r(j)}$ for some j. According to Lemma 1.1, since $s_1, s_{n+m} \in g(S)$ then $s_1 = s'_1 \xRightarrow[T_{r(1)}]{t_1} s'_2 \xRightarrow[T_{r(2)}]{t_2} \cdots \xRightarrow[T_{r(k)}]{t_k} s'_{k+1} = s_{n+m}$ for some $s'_1, \cdots, s'_{k+1} \in g(S)$ and $t_l \in \alpha_{r(l)} I^*$ for $1 \leq l \leq k$. This implies that there exists $s'_i \in g(S)$ and $t_i \in \alpha_n I^*$ in $s_1 \Rightarrow^* s_{n+m}$ where $t_i \in Enabled(s'_i, p)$.

According to the definition of $Reach(s)$, since $s_1 \in Reach(s)$ then $s_2 \in Reach(s)$, and hence, by induction $s_i \in Reach(s)$. However, this contradicts that p is deadlocked at s in the system $r(TS)$. □

3 Partial Order Reduction Algorithm

In this section, we explain our POR technique that reduces the transition system $r(TS)$. Before presenting our technique, we introduce some definitions and notations used throughout the remainder of this section. We partition the set V of visible transitions in TS into the set V_l of *locally visible actions*, which involve interactions between threads of a single process, and the set V_g of *globally visible actions*, which involve interactions between threads that belong to different processes. We assume that these sets are disjoint. This is expressed by Assumption 3.1.

Assumption 3.1. $V_l \cap V_g = \emptyset$

We also assume that only globally visible actions can be blocking. This is expressed by Assumption 3.2.

Assumption 3.2. $\beta(s) \subseteq V_g$

Algorithm 1 Partial Order Reduction Algorithm

```
1: Initialize: $stack \leftarrow \emptyset$; $visited \leftarrow \emptyset$; $p \leftarrow process$;
2: push$(s_0, p)$ onto $stack$;
3: while $stack \neq \emptyset$ do
4:   pop$(s, p)$ from $stack$;
5:   if $s \notin visited$ then
6:     add to $s$ in $visited$;
7:     for all $t \in Branch(s, p)$ do
8:       $< s_{succ}, p > \leftarrow < s', p' >$ where $s \overset{t}{\underset{T}{\Rightarrow}} s'$ and $T \in \Phi_{p'}$;
9:       push$(s_{succ}, p)$ onto $stack$;
10:    end for
11:  end if
12: end while
```

In $r(TS)$, we distinguish between different types of transitions. Let $s \overset{t}{\underset{T}{\Rightarrow}} s'$, where $T \in \Phi_p$. The transition t is said to be a *local transition* of p if $t \in V_l I^*$. We use L_p to denote the set of all local transitions of p. The transition t is said to be a *global transition* if $t \in V_g I^*$. We use G_p to denote the set of all global transitions of p.

We define the predicate $Blocked_G(s, p)$ for a state s and process p, where $Blocked_G(s, p)$ verifies to true *iff* there exist $t \in G_p$ and

$$s = s_1 \overset{t_1}{\underset{T_1}{\Longrightarrow}} s_2 \overset{t_2}{\underset{T_2}{\Longrightarrow}} \cdots \overset{t_{n-1}}{\underset{T_{n-1}}{\Longrightarrow}} s_n$$

such that,

- $T_1, \ldots, T_{n-1} \notin \Phi_p$,
- $t \in Blocked(s)$, and
- $t \in Enabled(s_n, p)$.

We need the predicate $Blocked_G(s, p)$ to identify if at the state s there exists a blocking transition of the process p which is waiting on other processes in order to make progress.

In the remainder of this section, first, we describe our POR algorithm (see Algorithm 1). It adapts the persistent-set selective search of Godefroid presented in Figure 1.4 of [2]. We show that in any state our algorithm explores a persistent set of transitions. Then, using Theorem 4.3 of Godefroid in [2], we conclude that our algorithm can detect deadlocks.

Our POR algorithm starts from the initial state s_0 in $r(TS)$. The set *visited* is used to keep track of visited states. The variable *process* represents one of the processes in the system. In a way, at s_0, $p \in P$ is randomly set to one of the processes in the system. At any other state $s \neq s_0$, where $s \in g(S)$, $p \in P$ represents the process whose thread takes a transition discovering s. In any state reached by the algorithm, it computes the set $Branch(s, p) \subseteq Enabled(s)$, and it continues searching the state space from the transitions in $Branch(s, p)$ only. The set $Branch(s, p)$ for a state $s \in g(S)$ is defined as below.

```
if $Enabled(s, p) \neq \emptyset$ and $Enabled(s, p) \subseteq V_l I^*$ and $\neg Blocked_G(s, p)$ then
  $Branch(s, p) \leftarrow Enabled(s, p)$
```

```
    else
        $Branch(s, p) \leftarrow Enabled(s)$
    end if
```

We say that s is a process-local state of p if,

$$Enabled(s, p) \neq \emptyset \wedge Enabled(s, p) \subseteq L_p \wedge \neg Blocked_G(s, p)$$

If the state s is not a process-local state, then s is referred to as a *system-global state*. In process-local states, the algorithm explores the transitions of only the current process. In system-global states, the algorithm explores the transitions of all processes in the system.

In our approach, we compute an approximation of $Blocked_G(s, p)$, which is denoted by $ApproxBlocked_G(s, p)$. If there exists a transition $t \in G_p$ where $t \in Blocked(s)$, then $ApproxBlocked_G(s, p)$ is evaluated as $true$, otherwise it is evaluated as $false$. Our approach implies that if $ApproxBlocked_G(s, p) = false$, then $Block(s, p) = false$, and for $ApproxBlocked_G(s, p) = true$, $Blocked_G(s, p)$ can be evaluated as $true$ or $false$. By evaluating $ApproxBlocked_G(s, p)$ to $true$, where $Blocked_G(s, p) = false$, the algorithm uses a system-global state where a process-local state can be used. Therefore, $ApproxBlocked_G(s, p)$ is considered a conservative approximation of $Blocked_G(s, p)$.

Below we define the notation of *independence of transitions*, adapted from Definition 3.1 in [2]. Using this notion, we later show that our algorithm preserves certain properties.

Definition 3.1. *Let t_1 and t_2 be two transitions of $r(TS)$. The transitions t_1 and t_2 are said to be independent in $r(TS)$ if for any $s \in g(S)$ the two following properties hold:*

- *if $s \overset{t_1}{\underset{T}{\Longrightarrow}} s'$, then $t_2 \in Enabled(s)$ iff $t_2 \in Enabled(s')$, and*
- *if $t_1, t_2 \in Enabled(s)$, then there exists a unique state s'' where $s \overset{t_1}{\underset{T_1}{\Longrightarrow}} s'_1 \overset{t_2}{\underset{T_2}{\Longrightarrow}} s''$ and $s \overset{t_2}{\underset{T_2}{\Longrightarrow}} s'_2 \overset{t_1}{\underset{T_1}{\Longrightarrow}} s''$.*

In our model, we assume that local transitions of one process are independent from the transitions taken by any other process in the system. This is expressed by Assumption 3.3.

Assumption 3.3. *If $t_1 \in L_{p_1}$ and $t_2 \in L_{p_2} \cup G_{p_2}$ where $p_1 \neq p_2$ then t_1 and t_2 are independent.*

Before we discuss the correctness of the algorithm, below, we provide the definition of persistent set taken from Definition 4.1 in [2].

Definition 3.2. *A set γ of transitions in a state s is persistent in s iff, for all nonempty sequences of transitions*

$$s = s_1 \overset{t_1}{\underset{T_1}{\Longrightarrow}} s_2 \overset{t_2}{\underset{T_2}{\Longrightarrow}} \cdots \overset{t_{n-1}}{\underset{T_{n-1}}{\Longrightarrow}} s_n \overset{t_n}{\underset{T_n}{\Longrightarrow}} s_{n+1}$$

from s in $r(TS)$ and including only transitions $t_i \notin \gamma, 1 \leq i \leq n, t_n$ is independent in s_n with all transitions in γ.

To prove that our algorithm perform a persistent-set selective search, we need show that the set of transitions, explored by Algorithm 1 in any state $s \in g(S)$, is a persistent set in s, and that set is nonempty if there is a transition enabled in s. Theorem 3.1 shows that the set of transitions explored at states are persistent. To prove this theorem we use Assumption 3.4.

Assumption 3.4. *If* $s_1 \xRightarrow[T_1]{t_1} s_2 \xRightarrow[T_2]{t_2} \dots s_n \xRightarrow[T_n]{t_n} s_{n+1}$ *where* $t_n \notin \mathit{Enabled}(s_i)$*,* $T_n \in \Phi_p$*, and* $T_i \notin \Phi_p$ *for* $1 \leq i < n$ *then* $t_n \in \mathit{Blocked}(s_1)$*.*

Theorem 3.1. *For any state* $s \in g(S)$ *and process* $p \in P$*, the set* $\mathit{Branch}(s,p)$ *of transitions explored by Algorithm 1 is a persistent set in* s*.*

Proof. For a system-global state s, $\mathit{Branch}(s,p)$ is set to $\mathit{Enabled}(s)$. Therefore, there is no nonempty sequence of transitions $s = s_1 \xRightarrow[T_1]{t_1} s_2 \xRightarrow[T_2]{t_2} \cdots \xRightarrow[T_n]{t_n} s_{n+1}$ which includes only $t_i \notin \mathit{Branch}(s,p)$ for $1 \leq i \leq n$. Therefore, according to Definition 3.2, $\mathit{Branch}(s,p)$ is a persistent set in s.

Now we show that, at a process-local state s of the process p, the set $\mathit{Branch}(s,p)$ is persistent in s. Towards contradiction, we assume that $\mathit{Branch}(s,p)$ is not a persistent set in s. Then, according to Definition 3.2, there exists a nonempty sequence of transitions

$$s = s_1 \xRightarrow[T_1]{t_1} s_2 \xRightarrow[T_2]{t_2} \cdots \xRightarrow[T_{n-1}]{t_{n-1}} s_n \xRightarrow[T_n]{t_n} s_{n+1} \tag{5}$$

such that:

(a) $t_1, t_2, ..., t_n \notin \mathit{Branch}(s,p)$.

(b) t_n in (5) is dependent with some transition $t \in \mathit{Branch}(s,p)$ in s_n.

Without a loss of generality, suppose (5) is a shortest such a sequence. Now we show that such a sequence cannot exist. Since s is a process-local state,

$$\mathit{Branch}(s,p) = \mathit{Enabled}(s,p) \tag{6}$$

First we show that for all $1 \leq i \leq n$,

$$\mathit{Enabled}(s,p) = \mathit{Enabled}(s_i,p) \tag{7}$$

by induction on i.

- Base case: $i = 1$, which is trivial.

- Inductive step: let $i > 1$. We assume that (7) holds in s_{i-1}, that is,

  $$\mathit{Enabled}(s,p) = \mathit{Enabled}(s_{i-1},p). \tag{8}$$

  Then we show that (7) holds in s_i, that is, $\mathit{Enabled}(s,p) = \mathit{Enabled}(s_i,p)$.

  According to (a), $t_{i-1} \notin \mathit{Branch}(s,p)$, which according to (6), implies that $t_{i-1} \notin \mathit{Enabled}(s,p)$. Therefore, according to (8), $t_{i-1} \notin \mathit{Enabled}(s_{i-1},p)$. That implies $t_{i-1} \in L_{p'} \cup G_{p'}$ where $p' \neq p$. Since s is a process-local state, $\mathit{Enabled}(s,p) \subseteq L_p$, and hence, $\mathit{Enabled}(s_{i-1},p) \subseteq L_p$. Therefore, according to Assumption 3.3, t_{i-1} is independent from any transition $t \in \mathit{Enabled}(s_{i-1},p)$. Therefore, from Definition 3.1, we can conclude that for any $t \in \mathit{Enabled}(s_{i-1},p)$, $t \in \mathit{Enabled}(s_i,p)$. Hence,

  $$\mathit{Enabled}(s_{i-1},p) \subseteq \mathit{Enabled}(s_i,p). \tag{9}$$

  Towards a contradiction, suppose that $\mathit{Enabled}(s_{i-1},p) \neq \mathit{Enabled}(s_i,p)$. Therefore, there exists a transition t', where $t' \in \mathit{Enabled}(s_i,p)$, but $t' \notin \mathit{Enabled}(s_{i-1},p)$.

Therefore, according to Definition 3.1, t' and t_{i-1} are dependent. As shown above, $t_{i-1} \in L_{p'} \cup G_{p'}$ where $p' \neq p$, and since $t' \in \mathit{Enabled}(s_i, p)$, then $t' \in L_p \cup G_p$. Therefore, since t' and t_{i-1} are dependent, from Assumption 3.3, we can conclude that $t_{i-1} \in G_{p'}$ and $t' \in G_p$.

Since $t' \notin \mathit{Enabled}(s_{i-1}, p)$, according to (8) $t' \notin \mathit{Enabled}(s, p)$. Moreover, since $t' \in \mathit{Enabled}(s_i, p)$, by showing that $T_1, \cdots, T_{i-1} \notin \Phi_p$, according to Assumption 3.4, we can conclude $t' \in \mathit{Blocked}(s)$.

It is shown above that $T_{i-1} \notin \Phi_p$. Towards a contradiction, suppose that there exists $T_k \in \Phi_p$ where $1 < k < i-1$. According to (8), since $t_k \in \mathit{Enabled}(s_k, p)$ then $t_k \in \mathit{Enabled}(s, p)$. However, this contradicts (a). Therefore, we can conclude that $t' \in \mathit{Blocked}(s)$. Since $t' \in G_p$, this implies that $\mathit{Blocked}_G(s, p) = \mathit{true}$. This is a contradiction, i.e., since s is a process local state, $\mathit{Blocked}_G(s, p) = \mathit{false}$. Hence, such t' does not exist, and

$$\mathit{Enabled}(s_i, p) \subseteq \mathit{Enabled}(s_{i-1}, p). \tag{10}$$

Therefore, from (9) and (10), we can conclude that (7) holds in the state s_i.

Next we show that $t_n \in \mathit{Enabled}(s_n, p)$. Towards contradiction, suppose $t_n \in L_{p'} \cup G_{p'}$ where $p' \neq p$. Since s is a process-local state, according to Algorithm 1, $\mathit{Branch}(s, p) \subseteq L_p$. Therefore, according to Assumption 3.3, t_n is independent from any $t \in \mathit{Branch}(s, p)$. However, that contradicts (b). Thus, $t_n \in \mathit{Enabled}(s_n, p)$. From (7), this implies $t_n \in \mathit{Enabled}(s, p)$. Hence, from (6), $t_n \in \mathit{Branch}(s, p)$. This contradicts our initial assumption (a). □

Theorem 3.1 shows that our algorithm performs a selective search through $r(TS)$, and in each state s reached by a process p, explores a set $\mathit{Branch}(s, p)$ of enabled transitions that is persistent.

Since the empty set is a persistent set, we need to show that the set $\mathit{Branch}(s, p)$ computed by the algorithm becomes empty if and only if $\mathit{Enabled}(s) = \emptyset$. For $\mathit{Enabled}(s) = \emptyset$, it is clear that $\mathit{Branch}(s, p) = \emptyset$. Towards a contradiction, consider that for the state s where $\mathit{Enabled}(s) \neq \emptyset$, $\mathit{Branch}(s, p) = \emptyset$. One possibility is that s is a local state of the process p. Therefore, $\mathit{Branch}(s, p) = \mathit{Enabled}(s, p)$ which implies that $\mathit{Enabled}(s, p) = \emptyset$. However, this is a contradiction, since for a local state s, our algorithm requires that $\mathit{Enabled}(s, p) \neq \emptyset$. Another possibility is that s is a global state. Therefore, $\mathit{Branch}(s, p) = \mathit{Enabled}(s)$ which implies that $\mathit{Enabled}(s) = \emptyset$. However, that contradicts our initial assumption. Thus, $\mathit{Branch}(s, p) = \emptyset$ iff $\mathit{Enabled}(s) = \emptyset$.

The following theorem captures Theorem 4.3 by Godefroid, presented in [2]. According to this theorem our algorithm preserves deadlocks reachable from s_0 in $r(TS)$. Note that the system, on which the algorithm is applied, is required to satisfy Assumption 1.2-3.3, otherwise there is no guarantee that global deadlocks are detected by Algorithm 1.

Theorem 3.2. *Let $s \in g(S)$ be a globally deadlocked state which is reachable from s_0 in $r(TS)$. Then s is reached from s_0 by Algorithm 1.*

References

[1] Patrice Godefroid. Model checking for programming languages using Verisoft. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 174–186, Paris, France, January 1997. ACM.

[2] Patrice Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems - An Approach to the State-Explosion Problem*, volume 1032 of *Lecture Notes in Computer Science*. Springer, 1996.