Addendum to
Dynamic Partial-Order Reduction
for Model Checking Software

Cormac Flanagan
University of California at Santa Cruz
cormac@cs.ucsc.edu

Patrice Godefroid
Microsoft Research
pg@microsoft.com

On page 6 of our POPL ’2005 paper, we wrote that “sleep sets can be added exactly as described in [10]”. Specifically, sleep sets can be added to the algorithm of Figure 3 as follows:

- line 5 should be replaced with
  
  let \( E = \{ q \in \text{enabled}(\text{pre}(S,i)) \mid q = p \text{ or } \exists j \in \text{dom}(S) : j > i \text{ and } q = \text{proc}(S_j) \text{ and } j \rightarrow_s p \} \setminus \text{Sleep}(\text{pre}(S,i)) \);

- line 7 should be replaced with
  
  \[ \text{if } \forall q \in \{ \text{enabled}(\text{pre}(S,i)) \setminus \text{Sleep}(\text{pre}(S,i)) \} \text{ \text{or} } \exists j \in \text{dom}(S) : j > i \text{ and } q = \text{proc}(S_j) \text{ and } j \rightarrow_s p \} \text{ \text{then} } \text{backtrack}(\text{pre}(S,i)) \text{ \text{else} } \text{PC}(S,j,p) \text{ \text{becomes}}: \]

The postcondition \( \text{PC}(S,j,p) \) then becomes:

\[
\forall q \in \{ \text{enabled}(\text{pre}(S,i)) \mid q = p \text{ or } \exists j \in \text{dom}(S) : j > i \text{ and } q = \text{proc}(S_j) \text{ and } j \rightarrow_s p \} \setminus \text{Sleep}(\text{pre}(S,i))
\]

The definition of \( PC(S,j,p) \) then becomes:

\[
\text{if } S \text{ is a transition sequence from } s_0 \text{ in } A_G
\text{ and } i = \max \{ i \in \text{dom}(S) \mid S_i \text{ is dependent and co-enabled with last}(s_0,p) \text{ and } i \not\rightarrow_s p \} \\
\text{ and } i \leq j
\text{ then}
\text{if } E(S,i,p) \neq \emptyset
\text{ then } \text{backtrack}(\text{pre}(S,i)) \cap E(S,i,p) \neq \emptyset
\text{ else } \text{PC}(S,j,p) = \text{enabled}(\text{pre}(S,i)) \setminus \text{Sleep}(\text{pre}(S,i))
\]

The postcondition \( \text{PC}(S,j,p) \) for \( \text{Explore}(S) \) becomes:

\[
\forall p \forall w : (\forall w_i \in [w] : w_i^1 \notin \text{Sleep}(\text{last}(S))) \Rightarrow \text{PC}(S,w,|S|,p)
\]

where \( \forall w_i \in [w] \) denotes the set of sequences \( w_i \) of transitions equivalent to \( w \) (i.e., transition sequences that are part of the same Mazurkiewicz’s trace – see [10] for details), and where \( w_i^1 \) denotes the first transition of \( w_i \).

In the presence of sleep sets, we use the following definition (similar notions are used in [9], for instance in Theorem 5.2):

**Definition 1.** A set \( T \subseteq T \) of transitions enabled in a state \( s \) is partially persistent in \( s \) iff, for all nonempty sequences \( w \) of transitions

\[
s_1 \rightarrow_1 s_2 \rightarrow_2 s_3 \ldots \rightarrow_{n-1} s_{n-1} \rightarrow_n s_n+1
\]

from \( s \) in \( A_G \) and including only transitions \( t_i \notin T \), \( 1 \leq i \leq n \), and such that \( \forall w_i \in [w] : w_i^1 \notin \text{Sleep}(s) \), \( t_n \) is independent with all the transitions in \( T \).

If \( \text{Sleep}(s) = \emptyset \), this definition coincides with the definition of persistent sets. Note that if \( T = \text{enabled}(s) \setminus \text{Sleep}(s) \), \( T \) is a partially persistent set in \( s \).

With sleep sets, Lemma 1 and Theorem 1 in the appendix remains the same except that “is a persistent set in \( s \)” has to be replaced by “is a partially persistent set in \( s \)” in both. From this modified Theorem 1, it follows from the proof of Theorem 2 in [10] that all deadlocks (terminating states) are visited by the combined algorithm using sleep sets.

For clarity and completeness, we include below those modified versions of Lemma 1 and Theorem 1 extended with sleep sets, as well as their proof.

**Lemma 1.** Whenever a state \( s \) reached after a transition sequence \( S \) is backtracked during the search performed by the algorithm of Figure 3, the set \( T \) of transitions that have been explored from \( s \) is a partially persistent set in \( s \), provided the postcondition \( \text{PC} \) holds for every recursive call \( \text{Explore}(S,t) \) for all \( t \in T \).

**Proof.** Let

\[
s = \text{last}(S)
T = \{ \text{next}(s,p) \mid p \in \text{backtrack}(s) \}
\]

If \( T \) is not \( \text{enabled}(s) \setminus \text{Sleep}(s) \), \( T \) is non-empty and we prove that \( T \) is a partially persistent set in \( s \) by contradiction: assume that there exist \( t_1, \ldots, t_n \notin T \) such that

1. \( S.t_1 \ldots t_n \) is a transition sequence from \( s_0 \) in \( A_G \) and
2. \( \forall w_i \in [t_1 \ldots t_n] : w_i^1 \notin \text{Sleep}(s) \) and
3. \( t_1, \ldots, t_{n-1} \) are all independent with \( T \) and
4. \( t_n \) is dependent with some \( t \in T \).
Let \( w = t_1 \ldots t_{n-1} \). By property of independence, this implies that \( t \) is enabled in the state \( \text{last}(S.w) \) and hence co-enabled with \( t_n \). Without loss of generality, assume that \( t_1 \ldots t_n \) is the shortest such sequence. We thus have that

\[
\forall 1 \leq i < n : i \rightarrow_{s.w} \text{proc}(t_n)
\]

(If this was not true for some \( i \), the same transition sequence without \( i \) would also satisfy our assumptions and be shorter.)

By definition, \( S.w \) is itself a transition sequence from \( s_0 \) in \( A_G \) and we have

\[
\text{next}(\text{last}(S.w), \text{proc}(t_n)) = t_n
\]

If \( \text{proc}(t) = \text{proc}(t_n) \) then

\[
t = \text{next}(\text{last}(S), \text{proc}(t))
= \text{next}(\text{last}(S.w), \text{proc}(t))
= t_n
\]

since \( t \) is independent with all the transitions in \( w \), contradicting that \( t_n \not\in T \). Hence \( \text{proc}(t) \neq \text{proc}(t_n) \).

Since \( t \) is in a different process than \( t_n \) and \( t \) is independent with all the transitions in \( w \), we have

\[
t_n = \text{next}(\text{last}(S.w), \text{proc}(t_n))
= \text{next}(\text{last}(S.w.t), \text{proc}(t_n))
= \text{next}(\text{last}(S.t.w), \text{proc}(t_n))
\]

Since \( t \in T \), \( t \) is executed from \( s \). Since \( \forall w_i \in [w] : w_i \not\in \text{Sleep}(s) \) and since \( t_1, \ldots, t_n \not\in T \) (i.e., none of those transitions are executed from \( s \)), none of the \( w_i \) transitions are in \( \text{Sleep}(\text{last}(S.t)) \) (by construction – see the rules for defining sleep sets in [10]).

Let \( i = |S| + 1 \). Consider the postcondition

\[
\text{PC}(S.t.w, i, \text{proc}(t_n))
\]

for the recursive call \( \text{Explore}(S.t) \). Clearly,

\[
i \not= \text{S.t.w} \text{proc}(t_n)
\]

(since \( t \) is in a different process than \( t_n \) and \( t \) is independent with \( t_1, \ldots, t_{n-1} \)). In addition, we have (by definition of \( E \)):

\[
E(S.t.w, i, \text{proc}(t_n)) \subseteq \{ \text{proc}(t_1), \ldots, \text{proc}(t_{n-1}) \} \cap \text{enabled}(s)
\]

Moreover, we have by construction:

\[
\forall j \in \text{dom}(S.t.w) : j > i \Rightarrow j \rightarrow_{S.t.w} \text{proc}(t_n)
\]

Hence, by the postcondition \( \text{PC} \) for the recursive call \( \text{Explore}(S.t) \), either \( E(S.t.w, i, \text{proc}(t_n)) \) is nonempty and at least one process in \( E(S.t.w, i, \text{proc}(t_n)) \) is in \( \text{backtrack}(s) \), or \( E(S.t.w, i, \text{proc}(t_n)) \) is empty and all the processes in \( \text{enabled}(s) \setminus \text{Sleep}(s) \) are in \( \text{backtrack}(s) \). In either case, at least one transition among \( \{ t_1, \ldots, t_n \} \) is in \( T \). This contradicts the assumption that \( t_1, \ldots, t_n \not\in T \).

\[ \square \]

**Theorem 1.** Whenever a state \( s \) reached after a transition sequence \( S \) is backtracked during the search performed by the algorithm of Figure 3 in an acyclic state space, the postcondition \( \text{PC} \) for \( \text{Explore}(S) \) is satisfied, and the set \( T \) of transitions that have been explored from \( s \) is a partially persistent set in \( s \).

**Proof.** Let

\[
s = \text{last}(S)
T = \{ \text{next}(s, p) \mid p \in \text{backtrack}(s) \}
\]

The proof is by induction on the order in which states are backtracked.

(Base case) Since the state space \( A_G \) is acyclic and since the search is performed in depth-first order, the first backtracked state must be either a deadlock where no transition is enabled, or a state \( s \) where \( \text{enabled}(s) = \text{Sleep}(s) \) (i.e., all transitions enabled in \( s \) are in \( \text{Sleep}(s) \)). Therefore, in either case, the postcondition for that state becomes \( \forall p : \text{PC}(S, |S|, p) \), and is directly established by lines 3–9 of the algorithm of Figure 3.

(Inductive case) We assume that each recursive call to \( \text{Explore}(S.t) \) satisfies its postcondition. That \( T \) is a partially persistent set in \( s \) then follows by Lemma 1. We show that \( \text{Explore}(S) \) ensures its postcondition \( \text{PC} \) for any \( p \) and \( w \) such that \( S.w \) is a transition sequence from \( s_0 \) in \( A_G \) and such that \( \forall w_i \in [w] : w_i \not\in \text{Sleep}(last(S)) \).

1. Suppose some transition in \( w \) is dependent with some transition in \( T \). In this case, we split \( w \) into \( X.t.Y \), where all the transitions in \( X \) are independent with all the transitions in \( T \) and \( t \) is the first transition in \( w \) that is dependent with some transition in \( T \). Since \( T \) is a partially persistent set in \( s \), \( t \) must be in \( T \) (otherwise, \( T \) would not be partially persistent in \( s \)). Thus, \( t \) is independent with all the transitions in \( X \). By property of independence, this implies that the transition sequence \( t.X.Y \) is executable from \( s \). It also implies that \( t \) is one of the \( w_i \) transitions.

   (Case 1.a) If \( t \) is the first transition of the \( w_i \) transitions of \( w \) to be executed in \( s \) and since none of those are in \( \text{Sleep}(last(S)) \), then \( \text{Sleep}(last(S.t)) \) does not contain any of the \( w_i \) transitions either (by the rules defining sleep sets in [10]). By applying the inductive hypothesis to the recursive call \( \text{Explore}(S.t) \) for the sequence \( X.Y \), we know

\[
\forall p : \text{PC}(S.t.X.Y, |S| + 1, p)
\]

which implies (by the definition of \( \text{PC} \)) that

\[
\forall p : \text{PC}(S.t.X.Y, |S|, p)
\]

Since \( t \) is independent with all the transitions in \( X \), we also have that

\[
\forall i \in \text{dom}(S.t.X.Y) : i \rightarrow_{S.t.X.Y} p \iff i \rightarrow_{S.X.t.Y} p
\]

Therefore, by definition,

\[
\text{PC}(S.t.X.Y, |S|, p) \iff \text{PC}(S.X.t.Y, |S|, p)
\]

We can thus conclude that

\[
\forall p : \text{PC}(S.X.t.Y, |S|, p)
\]

(Case 1.b) Otherwise, let \( t' \) be the first transition of the \( w_i \) transitions of \( w \) which is executed in \( s \) before \( t \). We thus have \( w = X.t.W.t'.Z \). Since \( t' \) is one of the \( w_i \) transitions, we know (by definition of \( w_i \)) that \( t' \) is independent of all transitions in \( X.t.W \).
The same reasoning as in the previous case 1.a can be applied to \( \text{Explore}(S.t') \) and the sequence \( X.t.W.Z \). We can thus prove that

\[
PC(S.t'.X.t.W.Z,|S|,p) \iff PC(S.X.t.W.t'.Z,|S|,p)
\]

and conclude again that

\[
\forall p: \; PC(S.w,|S|,p)
\]

2. Suppose that all the transitions in \( w \) are independent with all the transitions in \( T \) and \( p \in \text{backtrack}(s) \). Then

(a) \( \text{next}(s,p) \in T \);
(b) \( \text{next}(s,p) \) is independent with \( w \);
(c) \( p \) is a different process from any transition in \( w \);
(d) \( \text{next}(\text{last}(S.w),p) = \text{next}(\text{last}(S),p) \);
(e) \( \forall i \in \text{dom}(S): \; i \rightarrow_{S.w} p \iff i \rightarrow_{S} p \).

Thus, we have \( PC(S.w,|S|,p) \iff PC(S,|S|,p) \), and the latter is directly established by the lines 3–9 of the algorithm for all \( p \).

3. Suppose that all the transitions in \( w \) are independent with all the transitions in \( T \) and \( p \not\in \text{backtrack}(s) \). Pick any \( t \in T \). We then have that

(a) \( \text{proc}(t) \neq p \);
(b) \( t \) independent with all the transitions in \( w \);
(c) \( \text{next}(\text{last}(S.w),p) = \text{next}(\text{last}(S.t.w),p) \);
(d) \( \forall i \in \text{dom}(S): \; i \rightarrow_{S.w} p \iff i \rightarrow_{S.t.w} p \).

Thus, we have \( PC(S.w,|S|,p) \iff PC(S.t.w,|S|,p) \).

Since none of the \( w^1_i \) transitions are in \( \text{Sleep}(\text{last}(S)) \) and since none of those transitions are executed in \( s \), \( \text{Sleep}(\text{last}(S.t)) \) does not contain any of the \( w^1_i \) transitions either (by the rules defining sleep sets in [10]).

By applying the inductive hypothesis to the recursive call \( \text{Explore}(S.t) \), we know

\[
\forall p: \; PC(S.t.w,|S|+1,p)
\]

which implies (by the definition of \( PC \)) that

\[
\forall p: \; PC(S.t.w,|S|,p)
\]

which in turn implies

\[
\forall p: \; PC(S.w,|S|,p)
\]

as required.

Acknowledgements: We thank Katie Coons for pointing out that combining sleep sets with our POPL’2005 algorithm is not as immediate as originally thought and for helpful comments.