Interpolation Sequences Revisited*

G. Cabodi and S. Nocco and S. Quer
Dipartimento di Automatica ed Informatica
Politecnico di Torino - Torino, Italy
Email: {gianpiero.cabodi, sergio.nocco, stefano.quer}@polito.it

Abstract—This work revisits the formulation of interpolation sequences, in order to better understand their relationships with Bounded Model Checking and with other Unbounded Model Checking approaches relying on standard interpolation.

We first focus on different Bounded Model Checking schemes (bound, exact and exact-assume), pointing out their impact on the interpolation-based strategy. Then, we compare the abstraction ability of interpolation sequences with standard interpolation, highlighting their convergence at potentially different sequential depths. We finally propose a tight integration of interpolation sequences with an abstraction-refinement strategy.

Our contributions are first presented from a theoretical standpoint, then supported by experimental results (on academic and industrial benchmarks) adopting a state-of-the-art academic tool.

I. INTRODUCTION

Craig interpolants (ITPs for short) [1], [2], introduced by McMillan [3] in the Unbounded Model Checking (UMC) field, have shown to be effective on difficult verification instances. Among the attempts that have been made to improve the initial proposed scheme [4], [5], [6], [7], [8], interpolation sequences (ITPSEQs) [9] represent an interesting new idea.

ITPSEQs stress the affinity of ITP-based methods to standard Bounded Model Checking (BMC), as they basically propose a single-loop scheme, generating several interpolants (a sequence) from a single refutation proof. Interpolants are then virtually stored in a matrix-like data structure, which is exploited to evaluate over-approximate reachable state sets. ITPSEQs may have an edge over standard ITPs as their affinity with BMC turns out to be convenient for falsification and for some proved properties.

Starting from theoretical considerations and experimental analysis, we observed the following weaknesses of standard ITP-based verification, compared with BMC:

- ITPs intrinsically rely on SAT calls looking for states violating the property at any depth (bound-k checks). Unsatisfiable instances are thus harder to prove, and, as a consequence, they produce larger refutation proofs (and interpolants).
- ITPs can benefit from other abstraction techniques (e.g., localization abstraction [10]), but they require ad-hoc formulations and implementations.

Given the above observations, ITPSEQs can be attractive for the following reasons:

- ITPSEQs do not require bound-k checks, as they are based on the computationally lighter formulations. To this respect, alternative schemes [11] may deserve further investigation.
- The affinity of ITPSEQs to BMC may be used to tightly integrate ITPSEQs with Counterexample Based Abstraction schemes (CBA) [12], [13]. The convergence speed of ITPSEQs, versus the one of pure ITP-based methods, is another issue that we will discuss. To this respect, we partially contrast one of the conclusions of [9], by showing that standard interpolation can converge at shorter depths than ITPSEQs. Furthermore, to increase the convergence speed, we introduce an intermediate strategy, which we call Serial Interpolation Sequence (SITPSEQ). The resulting engine, albeit not as mature as standard interpolation, and not competitive in a broad sense, shows interesting results on specific properties on both academic and industrial benchmarks.

II. BACKGROUND

A. Bounded Model Checking

Given a sequential system $M$ and an invariant property $p$, SAT-based BMC [14] is an iterative process to check the validity of $p$ up to a given bound. To perform this task, the system transition relation $T$ is unrolled $k$ times

$$T^k(V^{0..k}) = \bigwedge_{i=0}^{k-1} T(V^i, V^{i+1})$$

to implicitly represent all state paths of length $k$. After that, BMC tools may implement different checks, characterized by decreasing complexity:

- $bmc^k_B(V^{0..k}) = S_0(V^0) \land T^k(V^{0..k}) \land \bigvee_{i=1}^{k} \neg p(V^i)$
- $bmc^k_E(V^{0..k}) = S_0(V^0) \land T^k(V^{0..k}) \land \neg p(V^k)$
- $bmc^k_A(V^{0..k}) = S_0(V^0) \land T^k(V^{0..k}) \land \bigwedge_{i=1}^{k-1} p(V^i) \land \neg p(V^k)$

The first one, referred to as bound-$k$, looks for counterexamples falsifying $p$ at any distance from the initial states $S_0$. The second one, called exact-$k$, looks for paths in which a state violating $p$ appears after exactly $k$ steps (but violation is allowed at shorter distances as well). The third one, referred to as exact-assume-$k$ (or simply assume-$k$), looks for paths of length $k$ strictly reaching $\neg p$ only in the last time frame.

If we just look at the SAT solver run time, $bmc^k_A$ usually offers the best performance [11].

---

*This work was supported in part by SRC contract 2009-TJ-1968.
978-3-9810801-7-9/DATE11/©2011 EDAA
B. Craig Interpolants

Definition 1: Let A and B be two inconsistent Boolean formulas, i.e., such that \( A \land B \equiv \bot \). An ITP \( I \) for \( (A, B) \) is a formula such that: 1) \( A \models I \), 2) \( I \land B \equiv \bot \), and 3) \( \text{supp}(I) \subseteq \text{supp}(A) \cup \text{supp}(B) \).

An interpolant \( I = \text{Itp}(A, B) \) can be derived, as an AND/OR circuit, from the refutation proof of \( A \land B \) [3].

Consider now the Boolean formula \( \text{bmc}_{k}^B \), representing the bound-\( k \) BMC check for depth \( k \). If \( \text{bmc}_{k}^B \equiv \bot \), then an ITP \( I_k \) can be computed by partitioning \( \text{bmc}_{k}^B \) into:

\[
A = S_0(V^0) \land T(V^0, V^1) \\
B = \bigwedge_{i=1}^{k-1} T(V^i, V^{i+1}) \land \bigvee_{i=1}^k \neg p(V^i)
\]

(1)

By definition, \( I_k \) represents an over-approximation of the states \( S_1 \) reachable in one step from \( S_0 \), with the property that it does not contain states on paths of length \( k - 1 \) to a failure state. If we replace \( S_0 \) with \( I_1 \) in \( \text{bmc}_{k}^B \), it is possible to find a new ITP \( I_2 \). The process can be iteratively extended to form an over-approximate traversal, where, at the \( j \)-th iteration, \( I_j \) and \( I_{j+1} \) represent the domain and co-domain states of image computations. If a fixed-point is reached, the property is true. Otherwise, if a satisfiable instance is found during the approximate traversal, the value of \( k \) is increased, and the traversal restarted. Figure 1 provides a high-level pseudo-code implementing the ITP-based algorithm.

C. Interpolation Sequences

Definition 2: Let \( \Gamma_{1..n} = \{ A_1, A_2, \ldots, A_n \} \) be a set of formulas such that \( \bigwedge_{i=1}^n A_i \equiv \bot \). An ITPSEQ \( \Gamma_{0..n} \) for \( \Gamma_{1..n} \) is an ordered set \( (I_0, I_1, \ldots, I_n) \) such that: 1) \( I_0 \equiv T \) and \( I_n \equiv \bot \), 2) \( I_i \land A_{i+1} \Rightarrow I_{i+1} \) for every \( 0 \leq i < n \), and 3) \( \text{supp}(I_i) \subseteq \text{supp}(A_i) \cap \text{supp}(A_{i+1}) \) for any \( 0 < i < n \).

Given a refutation proof \( \Pi \) for \( \bigwedge_{i=1}^n A_i \), the computation of a sequence \( I_{0..n} = \text{ITPSEQ}(\Pi_{1..n}) \) can be achieved through the following relation for any \( 0 < j < n \):

\[
I_j = \text{Itp}(\bigwedge_{i=1}^j A_i, \bigwedge_{i=j+1}^n A_i)
\]

(2)

That is, each \( I_j \) is a Craig interpolant extracted from the same proof \( \Pi \) by simply changing the role played by the terms in \( \Gamma_{1..n} \) during the computation.

Given an unsatisfiable formula \( \text{bmc}_{k}^E \), an ITPSEQ \( I_{0..k+1} \) can be computed by partitioning \( \text{bmc}_{k}^E \) into a set

\[
\Gamma_{1..k+1}(bmc_{k}^E) = \{ A_1, \ldots, A_{k+1} \}, \text{ such that:}
\begin{align*}
A_1 &= S_0(V^0) \land T(V^0, V^1) \\
A_i &= T(V^{i-1}, V^i) \text{ for all } 2 \leq i \leq k \\
A_{k+1} &= \neg p(V^k)
\end{align*}
\]

Then, by Definition 1, we have that each \( I_j \) is an over-approximation of \( S_j \), i.e., \( S_j \subseteq I_j \). Furthermore, by Definition 2, all states in \( I_k \) do not violate \( p \) (since \( I_k \land \neg p \equiv \bot \)). Finally, we also have that each \( I_{j+1} \) represents an over-approximation of \( I_j \) for all \( 0 < j < k \), as \( I_j \land T \Rightarrow I_{j+1} \).

By the observations above, if we define \( I_j = \bigwedge_{i=1}^k I_i \), with each \( I_i \) computed by partitioning the BMC formulae generated for smaller depths, it holds that \( I_j \) is an approximated image of \( S_j \) not including states leading to a failure in \( k - j \) steps. This means that the \( \mathcal{I} \) state sets can be exploited in a very similar way to the ITPs used by procedure \text{ItpVerif} in Figure 1.

Figure 2 reports an UMC algorithm based on ITPSEQs.

ItPSEQVerif \( (S_0, T, p) \) for \( (k = 1; k++) \)

\[
\begin{align*}
\text{ex} &= \text{Sat}(\text{bmc}_{k}^E) \\
\text{if } (\text{ex} \neq \emptyset) \text{ return FAIL} \\
R_0 &= S_0 \\
I_{0..k+1} &= \text{ITPSEQ}(\Gamma_{1..k+1}(\text{bmc}_{k}^E)) \\
\text{for } (j = 1; j < k; j++) \\
I_j &= I_{j-1} \lor I_j \\
R_j &= R_{j-1} \lor I_j \\
A &= I_j \land T \\
\text{ex} &= \text{Sat}(A \land B) \\
\end{align*}
\]

Fig. 1. Interpolant-based verification.

Fig. 2. Unbounded Model Checking based on interpolation sequences.

The verification process is pictorially represented in Figure 3. During the first iteration (Figure 3(a)), the formula \( \text{bmc}_{k}^E \) is checked: if it is unsatisfiable, then the interpolation sequence \( I_{1,2}^E \) is computed as \( (I_1 = T, I_2^E = \text{ITP}(S_0 \land T, \neg p), I_2^E = \bot) \). \( I_1^E \) is also assigned to \( I_1 \) and checked against \( S_0 \); if \( I_1 \Rightarrow S_0 \) the process is immediately stopped with a PASS result, otherwise a new iteration is started. Upon an unsatisfiable result for \( \text{bmc}_{k}^E \) (Figure 3(b)), \( I_{0..3}^E \) is generated, being \( I_0 = T \), \( I_1^E = \text{ITP}(S_0 \land T, T, \neg p) \), \( I_2^E = \text{ITP}(S_0 \land T, T, \neg p) \) and \( I_3^E = \bot \). This time, \( I_1 \) is obtained as the conjunction between \( I_1^E \) and the previously generated ITP \( I_1^E \), whereas \( I_2^E \) is directly assigned to \( I_2^E \). Both \( I_1 \) and \( I_2^E \) are checked for inclusion in \( R_0 = S_0 \) and \( R_1 = R_0 \lor I_1 \), respectively. The procedure terminates in case such a relationship is found, otherwise the depth \( k \) is increased and the reasoning is repeated (Figure 3(c)).
III. BMC checks: From bound-k to assume-k

As previously described, standard interpolation relies on bound-k BMC checks, as the B term in ITP computations (see Equation 1) includes the target \( \neg p \) at all time frames:

\[
B^k_B(V^{1..k}) = T^k(V^{1..k}) \land \bigvee_{j=1}^k \neg p(V^j)
\]

This is a strict requirement of the algorithm, as it is necessary to ensure the correctness of function ITPVERIF. However, this also implies that the complexity of the SAT runs (and eventually the size of the related refutation proofs) is higher than with the alternative formulations, i.e., the exact-k and assume-k:

\[
B^k_E(V^{1..k}) = T^k(V^{1..k}) \land \neg p(V^k)
\]

\[
B^k_A(V^{1..k}) = T^k(V^{1..k}) \land \bigwedge_{j=1}^{k-1} p(V^j) \land \neg p(V^k)
\]

This issue has been already partially addressed in previous publications, proposing partitioned ITPs [8], where given a disjoint decomposition of the B term:

\[
B^k_B = \bigvee_{j=1}^k B^j_E = \bigvee_{j=1}^k B^j_A
\]

an interpolant was expressed as the conjunction of (smaller) ITPs, individually computed following an exact or exact-assume strategy:

\[
\text{ITP}(A, B^k_B) = \bigwedge_{j=1}^k \text{ITP}(A, B^j_E) = \bigwedge_{j=1}^k \text{ITP}(A, B^j_A)
\]

The formulation is quite similar to the one adopted for ITPSEQs, where the over-approximate reachability set \( I_j \), i.e., the column-based conjunction in the matrix-like arrangement of ITPs, is given by \( I_j = \bigwedge_{k} I_i^j \), each \( I_i^j \) being the \( i \)-th term of an ITPSEQ generated by the exact BMC problem at depth \( i \).

In our implementation of ITPSEQs, we explicitly resort to an assume-k formulation, by performing SAT calls with the \( \text{bmc}^k_A \) propositional formula. We experimentally compare the two strategies in Section VI, showing that the assume-k approach almost always outperforms the exact-k scheme.

IV. SERIAL INTERPOLATION SEQUENCES

Our experimental practice shows that ITPSEQs can offer different trade-offs among abstraction levels, sequential depth at convergence, and overall size of interpolants, with respect to standard interpolation. Hence, we consider ITPSEQ as an additional engine within a potential portfolio of available MC techniques. We thus propose a hybrid method, where a chain of (standard) ITP computations works as an ITPSEQ. We call this method Serial Interpolation Sequence (STPSEQ). In the sequel, we first motivate the approach, by discussing the performance of interpolant-based approaches in terms of sequential depth. Then we introduce the concepts of serial and parallel interpolation sequences.

A. Abstraction Operators and Sequential Diameter(s)

Let us now discuss of the potential reduction of traversal depths, that is always obtained as a by-product of abstraction techniques. As a term of comparison, in order to evaluate the traversal depths, we will use circuit diameters (forward and/or backward), since they represent an exact measure (though often not available) of sequential behaviors. The diameter of a sequential system is the longest shortest path between any two states in the state transition graph. Forward \( d_F \) and backward \( d_B \) diameters, referred to initial and target states, are preferred once those states are given.

Most of the existing theory on ITP-based model checking just considers the backward diameter \( d_B \) of the system under verification, that represents an upper bound [3] for the number \( k_{fp} \) of outer iterations performed by the standard interpolation algorithm (see Figure 1). But this is just a partial performance estimation, as the overall performance also depends on the number \( j_{fp} \) of inner iterations. Interpolation depth is thus often measured not just in terms of outer iterations \( k_{fp} \), but rather of couples \( (k_{fp}, j_{fp}) \), representing the numbers of outer/inner iterations at the fixed-point. Alternatively, the sum \( k_{fp} + j_{fp} \) is also used, that corresponds to a "virtual" bound (maximum length considered for initial-to-target paths).

In contrast to backward diameter, the exact forward diameter \( d_F \) is not an upper bound to forward over-approximate traversals (number of inner iterations), nor \( d_F + d_B \) is an upper limit for the maximum BMC bound. So not only are we taking terms of comparisons (circuit diameters) that are typically not known, but even in the case of known (or estimated) values, we cannot assume them as an upper bound for ITP-based traversal depths. These observations are fully in line with other abstraction/over-approximation techniques (including BDD-based ones), where nothing can be said about the diameter(s) of an abstract state graph w.r.t. the concrete one. However, it is true that over-approximate traversals usually converge at smaller depths than circuit diameters, which thus represent a good (practical) term of comparison to evaluate the convergence power of an abstraction-based approach.

B. Sequential Depth of Interpolation Sequences

Let us now focus on ITPSEQs, where \( k_{fp} \) and \( j_{fp} \) represent the BMC bound and the approximate forward depth, respectively, providing the fixed-point (i.e., when \( I_{j_{fp}} \Rightarrow R_{j_{fp} - 1} \)). Because of the way the \( I \) sets are built, we have that \( k_{fp} - j_{fp} \leq d_B \), whereas no relation can be established between \( j_{fp} \) and \( d_F \).

As already pointed out, we are interested in practical considerations, so let us explicitly state that we expect \( j_{fp} < d_F \), because of the over-approximation, and, as a consequence, \( k_{fp} < d_F + d_B \). Should we expect \( j_{fp} < d_F \), or rather \( j_{fp} \approx d_F \)? As no theoretical upper bound can be established, we draw our discussion on a heuristic basis. Let us first observe that the abstraction process in ITP-based approaches is fully driven by SAT-solvers. Then, on the one hand, the depth of a traversal visiting an abstract space is unpredictable. On the other hand, the over-approximation freedom for ITP \((A, B)\), i.e., the freedom left for visiting unreachable states, may range over the sub-space including \( A \) and not intersecting \( B \). The smaller is this sub-space, the more constrained (and tightened to actually reachable states) is the over-approximation. Or, in other words, a high abstraction level implies a large degree of freedom for over-approximation.
But the over-approximation level is just part of the problem, and often not a good indication for the traversal convergence/depth. Another key issue is finding a (least) fixed-point, i.e., a chain of over-approximated state sets \( (R_j = \bigvee_{i=0}^{j-1} R_i) \) such that \( R_j \land T \Rightarrow R_j \).

On both of the above mentioned issues, standard ITPs and ITPSEQs show a clearly different nature: (1) State sets in standard ITPs are generated from a cumulative application of (ITP-based) over-approximations, as \( I_1 \) is used to generate its approximated image \( I_2 \), which is in turn used for computing \( I_3 \), etc. This can be captured by a recurrent formulation:

\[
I_j = \text{ITP}(I_{j-1} \land T, B^{k-j})
\]

with \( I_0 = S_0 \). In our experience, the cumulative effect of ITP operators tends to enhance the abstraction level, and to produce shorter traversal depths than \( d_F \). (2) Over-approximate state sets in ITPSEQs, derive from conjunctions of ITPs, that are directly derived from BMC problems:

\[
I_j^k = \text{ITP}(S_0 \land T^j, B^{k-j})
\]

The \( A \) term in the ITPSEQ is a forward circuit unrolling, and its projection over variables \( V^j \) (the lower bound for the ITP) is the exact set of states forward reachable in \( j \) steps (\( S_j \)).

Thus, abstractions in ITPSEQs are potentially tighter, as they derive from individual refutation proofs, to be further combined by column-based conjunctions. On the other hand, standard interpolation cumulatively combines all the contributions: an ITP, generated from a SAT refutation proof, is used as one of the inputs for the next ITP computation.

Our experience basically confirms the above guesses. Whenever we considered problems where the traversal depth is in the range from a few tens to hundreds, standard interpolation tends to converge at shorter depths than interpolation sequences.

### C. Parallel and Serial Interpolation Sequences

As described in Section II-C, all components of an ITPSEQ are computed from the same refutation proof. We call this type of computation parallel, as the ITPs can be generated from the proof in any order (virtually in parallel). We introduce an alternative way to generate an ITPSEQ, that we call serial, as every term is obtained by taking into account the previous one, with a technique more similar to standard interpolation.

**Definition 3:** Given a set of formulas \( \Gamma_{1..n} \) a Serial ITPSEQ (SITPSEQ) for \( \Gamma_{1..n} \) is an ITPSEQ where, for any \( 0 < j < n \), the \( I_j \) term is computed as:

\[
I_j = \text{ITP}(I_{j-1} \land A_j, \bigwedge_{i=j+1}^{n} A_i)
\]

The idea is directly derived from standard interpolation, but it is applied to an ITPSEQ scheme, where the \( B \) terms of successive interpolants are not kept unchanged, but reduced at each iteration. The ITPSEQ properties are guaranteed by construction. The overall SAT time required to obtain the sequence is obviously increased, as the sequence generation needs to loop through a number of SAT calls and standard ITP computations. The potential advantage lies in the cumulative interpolation effect, with a possible saturation of the over-approximation.

More in general, several intermediate solutions are possible, both static and dynamic. Figure 4 reports the pseudo-code of a function able to build an ITPSEQ ranging from being totally parallel to completely serial, depending on the \( \alpha \) parameter (\( 0 \leq \alpha \leq 1 \)). More precisely, the first \( n_\alpha \) terms of the sequence are generated according to Equation 3, whereas the remaining ones are obtained through a parallel computation, as specified by Equation 2. For our experiments we chose \( \alpha = 0.5 \).

**V. INTERPOLATION SEQUENCES AND COUNTEREXAMPLE BASED ABSTRACTION**

Recent works [10] describe alternative ways to compute ITP-like over-approximations without resorting to SAT refutation proofs. Although the techniques were inspired by existing abstraction methods, they required a tight integration and an ad-hoc formulation for ITP-based Model Checking.

Due to their strong similarity with BMC, ITPSEQs appear much more suitable for integration with existing abstraction techniques, which work on a BMC basis. We address in this section abstraction-refinement, following the Counterexample Based Abstraction (CBA) scheme [13], as the integration with our method is straightforward. Furthermore, its abstraction strategy (i.e., refining an initially coarse abstraction), is dual to the one adopted for interpolation (i.e., letting the SAT solver to remove unnecessary details from the model). Although Proof Based Abstraction (PBA) [13] is also possible, we prefer the CBA strategy, because PBA is closer to standard interpolation, as they both start from SAT refutation proofs.

CBA approaches can be useful for both BMC and UMC. In the first case, any refinement done at bound \( k \) is introduced just for BMC checks at larger depths, i.e., BMC checks at lower bounds are not repeated. On the other hand, a full proof is typically restarted in UMC approaches whenever a newly refined circuit is available.

In our case we do not repeat the proofs, as the goal of refinements is only to provide unsatisfiable BMC instances at increasing bounds. The advantage in case of failure is obvious. On the other hand, in case of unsatisfiable checks, we expect smaller refutation proofs, and hence ITPs. As a consequence, we obtain a higher over-approximation for the computed state sets, which is not a guarantee (but it is often a good premise) for better convergence.

The related pseudo-code is provided in Figure 5. The modifications with respect to the original pseudo-code of Figure 2 can be clearly identified in the CBA functions REFINE and EXTEND. Our implementation of CBA is based on an iterative search of abstract counterexamples for each given bound \( k \).
Each abstract counterexample is checked on the concrete model $T$, exploiting function EXTEND. If the counterexample is validated, a failure is returned. In the opposite case, the abstract model $T_A$ is refined in a new model by function REFINE. When no more (abstract) counterexamples are found, the $T_A$ model is adopted in function SITPSEQ to perform the serial ITPSEQ computation.

**ITPSEQCBAVERIF** ($S_0$, $T$, $p$, $a_\alpha$)  
$T_A = \text{ABSTRACT}(T)$  
(for $k = 1$ ; $k++$)  
cex = $\text{SAT}(bm c_{k}^p(S_0, T_A, p))$  
while (cex $\neq \emptyset$)  
if (EXTEND (cex, $T$, $T_A$, $k$) $\neq \emptyset$) return FAIL  
$T_A = \text{REFINE}(cex, T, T_A, k)$  
cex = $\text{SAT}(bm c_{k}^p(S_0, T_A, p))$

$I_0, k+1 = \text{SITPSEQ}(T_{1..k+1}(bm c_{k}^p(S_0, T_A, p)), a_\alpha)$  
for ($j = 1$; $j < k$; $j++$)  
$I_j = I_j \land I_j$  
if ($I_j \Rightarrow R_{j-1}$) return PASS  
$R_j = R_{j-1} \lor I_j$  
$I_k = I_k$

if ($I_k \Rightarrow R_{j-1}$) return PASS

Fig. 5. Computation of serial and parallel interpolation sequence with CBA.

**VI. EXPERIMENTAL RESULTS**

We present a set of experimental data, on a selected sub-set of 100 publicly available and industrial circuits, oriented to provide both a comparison with standard interpolation, and an overall evaluation of the techniques proposed in the paper.

Our prototype ran on a 2.5 GHz Quad-core AMD workstation, with 8 GB of main memory, with 1800 seconds time limit, and 2 GBytes memory limit. We ran our implementations of: (a) standard ITPs, (b) standard ITPSEQs, (c) serial ITPSEQs (with $a_\alpha = 0.5$), and (d) serial ITPSEQs with CBA.

Figure 6 plots CPU times for individual runs, sorted with different orders for each approach, in order to provide monotonic curves. It basically shows that standard interpolation is more powerful than ITPSEQs, at least for proved properties, and that our improved ITPSEQ methods (serial sequences and integration with CBA) have the edge over standard ITPSEQs.

Table I provides more detailed data for the benchmarks completed by at least one of the ITPSEQ techniques. The table is divided into two main parts. The upper part includes mid-size problems, which are reported in order to provide a broader comparison among all techniques. The lower part focuses on industrial circuits, and it shows more challenging cases, often completed by fewer techniques. For each property, the table reports the design name followed by the number of primary inputs (#PI), and register count (#FF). After that, the BDDs section reports the exact diameters and the CPU time for both forward ($d_F$, $\text{Time}_F$) and backward ($d_B$, $\text{Time}_B$) BDD-based verification. The diameter is not reported in case an overflow or a failure occur. For each one of the subsequent parts (ITP, ITPSEQ, SITPSEQ, and ITPSEQCBA), Table I reports execution times, and depth measures (see Section IV-B). $k_{fp}$ represents the BMC bound, and $j_{fp}$ indicates the depth of the forward traversal (or the index of the cut) at fixed-points (a 0 value is reported in case of failure).

A careful analysis of $k_{fp}$ and $j_{fp}$ confirms our guesses of Section IV, i.e., that cumulative abstraction can provide convergence at shorter depths. The values of $k_{fp}$ and $j_{fp}$ are quite similar, with very few exceptions, for the three ITPSEQ approaches, on most of the experiments solved by all methods (typically completed at lower BMC bounds). We can also perceive the advantage of serial sequences and CBA in the verification instances left unsolved by standard ITPSEQs. The BMC bound ($k_{fp}$, in round brackets) over flow is typically larger than the one of serial sequences at fixed-point, indicating that the potential convergence of standard ITPSEQs (with unlimited time and memory bounds) would have been at larger depths.

Table I finally shows that ITPSEQ approaches get better results than standard ITP in some large industrial cases (> 500 latches), with ITPSEQCBA being the only approach able to complete all the presented industrial benchmarks.

A last very interesting observation comes from Figure 7. In this graph we present a scattered plot comparing execution times of standard ITPSEQs, in two versions, based on assume- $k$ and exact- $k$ checks, respectively. The graph clearly shows the advantage of the assume- $k$ formulation, confirming the discussion presented in Section III.

**VII. CONCLUSIONS**

This work revisits the formulation of Unbounded Model Checking (UMC) based on ITPSEQs, in order to better understand its relationships with Bounded Model Checking (BMC) and with other UMC approaches based on standard interpolation. Our contributions includes theoretical insight on interpolation sequences, and a new perspective for performance improvements. Our experimental results provide an interesting support for our formulations, and represent an encouraging platform for further tuning and a broader experimental work.

**REFERENCES**

TABLE I

<table>
<thead>
<tr>
<th>Model</th>
<th>Name</th>
<th>Pi</th>
<th>FF</th>
<th>BDDs</th>
<th>Time</th>
<th>TTP</th>
<th>TTPSEQ</th>
<th>TTPSEQCBΔ</th>
</tr>
</thead>
<tbody>
<tr>
<td>b0f0ambag3f3</td>
<td>8</td>
<td>25</td>
<td>14</td>
<td>1</td>
<td>19</td>
<td>710</td>
<td>19</td>
<td>11</td>
</tr>
<tr>
<td>eikS2030</td>
<td>18</td>
<td>58</td>
<td>11</td>
<td>1</td>
<td>422</td>
<td>22</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>eikS2522</td>
<td>18</td>
<td>62</td>
<td>11</td>
<td>1</td>
<td>422</td>
<td>17</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>eikS9535</td>
<td>16</td>
<td>105</td>
<td>11</td>
<td>2</td>
<td>33</td>
<td>37</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>intel006</td>
<td>345</td>
<td>350</td>
<td>9</td>
<td>22</td>
<td>2</td>
<td>15</td>
<td>ovf (36)</td>
<td>1348</td>
</tr>
<tr>
<td>intel049</td>
<td>136</td>
<td>141</td>
<td>10</td>
<td>15</td>
<td>24</td>
<td>21</td>
<td>ovf (27)</td>
<td>747</td>
</tr>
<tr>
<td>necltp3001</td>
<td>32</td>
<td>2826</td>
<td>9</td>
<td>22</td>
<td>2</td>
<td>15</td>
<td>ovf</td>
<td>1144</td>
</tr>
<tr>
<td>necltp3001</td>
<td>32</td>
<td>2826</td>
<td>9</td>
<td>22</td>
<td>2</td>
<td>15</td>
<td>ovf</td>
<td>1670</td>
</tr>
<tr>
<td>nusmvguidancep6</td>
<td>84</td>
<td>86</td>
<td>72</td>
<td>8</td>
<td>28</td>
<td>24</td>
<td>ovf (33)</td>
<td>240</td>
</tr>
<tr>
<td>nusmvguidancep8</td>
<td>84</td>
<td>86</td>
<td>72</td>
<td>12</td>
<td>31</td>
<td>9</td>
<td>ovf (28)</td>
<td>696</td>
</tr>
<tr>
<td>nusmvreactop4</td>
<td>74</td>
<td>76</td>
<td>272</td>
<td>9</td>
<td>13</td>
<td>25</td>
<td>ovf</td>
<td>427</td>
</tr>
<tr>
<td>nusmvca9op4</td>
<td>146</td>
<td>169</td>
<td>9</td>
<td>22</td>
<td>2</td>
<td>15</td>
<td>ovf</td>
<td>54</td>
</tr>
<tr>
<td>pdtvisscherence3</td>
<td>6</td>
<td>29</td>
<td>28</td>
<td>2</td>
<td>6</td>
<td>1</td>
<td>ovf</td>
<td>11</td>
</tr>
<tr>
<td>pdtvisscherence4</td>
<td>6</td>
<td>29</td>
<td>28</td>
<td>2</td>
<td>15</td>
<td>2</td>
<td>ovf</td>
<td>23</td>
</tr>
<tr>
<td>pdtvisscherence5</td>
<td>6</td>
<td>29</td>
<td>28</td>
<td>2</td>
<td>15</td>
<td>2</td>
<td>ovf</td>
<td>23</td>
</tr>
<tr>
<td>pdtvisscn2p2</td>
<td>16</td>
<td>75</td>
<td>16</td>
<td>3</td>
<td>10</td>
<td>10</td>
<td>ovf</td>
<td>40</td>
</tr>
<tr>
<td>pdtvisscn2p2</td>
<td>146</td>
<td>171</td>
<td>9</td>
<td>22</td>
<td>2</td>
<td>15</td>
<td>ovf</td>
<td>54</td>
</tr>
<tr>
<td>pdtvisscenh</td>
<td>68</td>
<td>163</td>
<td>9</td>
<td>22</td>
<td>2</td>
<td>15</td>
<td>ovf</td>
<td>11</td>
</tr>
<tr>
<td>pdtvisscenhf4</td>
<td>3</td>
<td>43</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>ovf</td>
<td>23</td>
</tr>
<tr>
<td>pdtvissfeis</td>
<td>68</td>
<td>296</td>
<td>9</td>
<td>22</td>
<td>2</td>
<td>15</td>
<td>ovf</td>
<td>23</td>
</tr>
<tr>
<td>pdtvissfeisg0</td>
<td>2</td>
<td>27</td>
<td>118</td>
<td>3</td>
<td>12</td>
<td>1</td>
<td>ovf</td>
<td>11</td>
</tr>
<tr>
<td>proconv1negn</td>
<td>61</td>
<td>86</td>
<td>9</td>
<td>22</td>
<td>2</td>
<td>15</td>
<td>ovf</td>
<td>11</td>
</tr>
<tr>
<td>proconv5</td>
<td>63</td>
<td>84</td>
<td>9</td>
<td>22</td>
<td>2</td>
<td>15</td>
<td>ovf</td>
<td>11</td>
</tr>
<tr>
<td>texasPImainp01</td>
<td>14</td>
<td>47</td>
<td>16</td>
<td>1</td>
<td>36</td>
<td>4</td>
<td>ovf</td>
<td>28</td>
</tr>
<tr>
<td>vischerence6</td>
<td>6</td>
<td>29</td>
<td>28</td>
<td>2</td>
<td>14</td>
<td>2</td>
<td>ovf</td>
<td>36</td>
</tr>
<tr>
<td>viscenen</td>
<td>22</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>ovf</td>
<td>10</td>
</tr>
</tbody>
</table>

Fig. 7. Performance comparison between interpolation sequences adopting exact-k or assume-k checks.


