Interpolation Sequences Revisited

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 initially 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

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

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

$$bmc_B^k(V^{0..k}) = S_0(V^0) \wedge T^k(V^{0..k}) \wedge \bigvee_{i=1}^{k} \neg p(V^i)$$
$$bmc_E^k(V^{0..k}) = S_0(V^0) \wedge T^k(V^{0..k}) \wedge \neg p(V^k)$$
$$bmc_A^k(V^{0..k}) = S_0(V^0) \wedge T^k(V^{0..k}) \wedge \bigwedge_{i=1}^{k-1} p(V^i) \wedge \neg p(V^k)$$

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_A^k usually offers the best performance [11].

*This work was supported in part by SRC contract 2009-TJ-1968.

B. Craig Interpolants

Definition 1: *Let A and B be two inconsistent Boolean formulas, i.e., such that $A \wedge B \equiv \bot$. An ITP I for (A, B) is a formula such that: 1) $A \Rightarrow I$, 2) $I \wedge B \equiv \bot$, and 3) $supp(I) \subseteq supp(A) \cap supp(B)$.*

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

Consider now the Boolean formula bmc_B^k, representing the *bound-k* BMC check for depth k. If $bmc_B^k \equiv \bot$, then an ITP I_1 can be computed by partitioning bmc_B^k into:

$$\begin{array}{rcl} A & = & S_0(V^0) \wedge T(V^0, V^1) \\ B & = & \bigwedge_{i=1}^{k-1} T(V^i, V^{i+1}) \wedge \bigvee_{i=1}^{k} \neg p(V^i) \end{array} \quad (1)$$

By definition, I_1 represents an over-approximation of the states S_1 reachable in one step from S_0, with the property that it does not contains states on paths of length $k-1$ to a failure state. If we replace S_0 with I_1 in bmc_B^k, it is possible to find a new ITP I_2. The process can be iteratively extended to produce 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.

```
ITPVERIF (S0, T, p)
 for (k = 1; ; k++)
   A = S0 ∧ T
   B = ⋀_{i=1}^{k-1} T ∧ ⋁_{i=1}^{k} ¬p
   cex = SAT(A ∧ B)
   if (cex ≠ ∅) return FAIL
   R0 = S0
   for (j = 1; cex = ∅; j++)
       Ij = ITP(A, B)
       if (Ij ⇒ Rj-1) return PASS
       Rj = Rj-1 ∨ Ij
       A = Ij ∧ T
       cex = SAT(A ∧ B)
```

Fig. 1. Interpolant-based verification.

C. Interpolation Sequences

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

Given a refutation proof Π for $\bigwedge_i A_i$, the computation of a sequence $I_{0..n} = \text{ITPSEQ}(\Gamma_{1..n})$ can be achieved through the following relation for any $0 < j < n$:

$$I_j \quad = \quad \text{ITP}(\bigwedge_{i=1}^{j} A_i, \bigwedge_{i=j+1}^{n} A_i) \quad (2)$$

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

Given an unsatisfiable formula bmc_E^k, an ITPSEQ $I_{0..k+1}^k$ can be computed by partitioning bmc_E^k into a set $\Gamma_{1..k+1}(bmc_E^k) = \{A_1, \ldots, A_{k+1}\}$, such that:

$$\begin{array}{rcl} A_1 & = & S_0(V^0) \wedge T(V^0, V^1) \\ A_i & = & T(V^{i-1}, V^i) \;\; for\ all \;\; 2 \leq i \leq k \\ A_{k+1} & = & \neg p(V^k) \end{array}$$

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

By the observations above, if we define $\mathcal{I}_j = \bigwedge_{i=j}^{k} I_j^i$, with each I_j^i computed by partitioning the BMC formulae generated for smaller depths, it holds that $\mathcal{I}_j$ 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 ITPVERIF in Figure 1.

Figure 2 reports an UMC algorithm based on ITPSEQs.

```
ITPSEQVERIF (S0, T, p)
  for (k = 1; ; k++)
      cex = SAT(bmc_E^k)
      if (cex ≠ ∅) return FAIL
      R0 = S0
      I0..k+1 = ITPSEQ(Γ1..k+1(bmc_E^k))
      for (j = 1; j < k; j++)
          Ij = Ij ∧ Ij
          if (Ij ⇒ Rj-1) return PASS
          Rj = Rj-1 ∨ Ij
      Ik = Ik
      if (Ik ⇒ Rj-1) return PASS
```

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 bmc_E^1 is checked: if it is unsatisfiable, then the interpolation sequence $I_{0..2}^1$ is computed as $(I_0^1 = \top, I_1^1 = \text{ITP}(S_0 \wedge T, \neg p), I_2^1 = \bot)$. I_1^1 is also assigned to $\mathcal{I}_1$ and checked against S_0: if $\mathcal{I}_1 \Rightarrow S_0$ the process is immediately stopped with a PASS result, otherwise a new iteration is started. Upon an unsatisfiable result for bmc_E^2 (Figure 3(b)), $I_{0..3}^2$ is generated, being $I_0^2 = \top$, $I_2^1 = \text{ITP}(S_0 \wedge T, T \wedge \neg p)$, $I_2^2 = \text{ITP}(S_0 \wedge T \wedge T, \neg p)$ and $I_3^2 = \bot$. This time, $\mathcal{I}_1$ is obtained as the conjunction between I_1^2 and the previously generated ITP I_1^1, whereas I_2^2 is directly assigned to $\mathcal{I}_2$. Both $\mathcal{I}_1$ and $\mathcal{I}_2$ are checked for inclusion in $R_0 = S_0$ and $R_1 = R_0 \vee \mathcal{I}_1$, 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)).

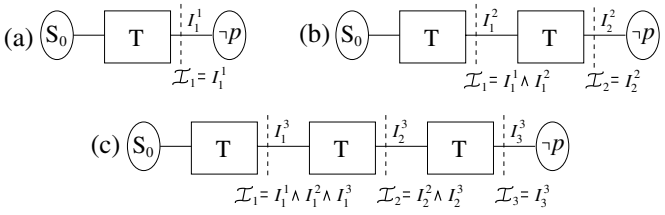

Fig. 3. A graphical representation of the interpolation sequence computation.

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:

$$B_B^k(V^{1..k}) \quad = \quad T^k(V^{1..k}) \wedge \bigvee_{j=1}^{k} \neg p(V^j)$$

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*:

$$\begin{aligned} B_E^k(V^{1..k}) &= T^k(V^{1..k}) \wedge \neg p(V^k) \\ B_A^k(V^{1..k}) &= T^k(V^{1..k}) \wedge \bigwedge_{j=1}^{k-1} p(V^j) \wedge \neg p(V^k) \end{aligned}$$

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

$$B_B^k \quad = \quad \bigvee_{j=1}^{k} B_E^j \quad = \quad \bigvee_{j=1}^{k} B_A^j$$

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

$$\text{ITP}(A, B_B^k) \quad = \quad \bigwedge_{j=1}^{k} \text{ITP}(A, B_E^j) \quad = \quad \bigwedge_{j=1}^{k} \text{ITP}(A, B_A^j)$$

The formulation is quite similar to the one adopted for ITPSEQs, where the over-approximate reachability set $\mathcal{I}_j$, i.e., the column-based conjunction in the matrix-like arrangement of ITPs, is given by $\mathcal{I}_j = \bigwedge_{i=j}^{k} I_j^i$, each I_j^i being the j-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 bmc_A^k 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 (SITPSEQ). 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 ($k_{fp} \leq d_B$), 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" BMC 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 $\mathcal{I}_{j_{fp}} \Rightarrow R_{j_{fp}-1}$). Because of the way the $\mathcal{I}$ 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 $j_p < d_F$, 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 $\text{ITP}(A, B)$, 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 $R_j \wedge T \Rightarrow R_j$.

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:

$$I_j \quad = \quad \text{ITP}(I_{j-1} \wedge T, B^k)$$

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:

$$I_j^k \quad = \quad \text{ITP}(S_0 \wedge T^j, B^{k-j})$$

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:*

$$I_j \quad = \quad \text{ITP}(I_{j-1} \wedge A_j, \textstyle\bigwedge_{i=j+1}^{n} A_i) \qquad (3)$$

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 α_s parameter ($0 \leq \alpha_s \leq 1$). More precisely, the first n_s 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 $\alpha_s = 0.5$.

```
SITPSEQ (Γ_{1..n}, α_s)
    n_s = ⌊α_s · n⌋
    I_0 = ⊤
    for (j = 1; j ≤ n_s; j++)
        I_j = ITP(I_{j-1} ∧ A_j, ⋀_{i=j+1}^{n} A_i)
    I_{n_s+1..n} = ITPSEQ({I_{n_s}, Γ_{n_s+1..n}})
    return I_{0..n}
```

Fig. 4. Computation of serial and parallel interpolation sequence.

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, α_s)
    T_A = ABSTRACT (T)
    for (k = 1; ; k++)
        cex = SAT(bmc^k_E(S_0, T_A, p))
        while (cex ≠ ∅)
            if (EXTEND (cex, T, T_A, k) ≠ ∅) return FAIL
            T_A = REFINE (cex, T, T_A, k)
            cex = SAT(bmc^k_E(S_0, T_A, p))
        R_0 = S_0
        I_{0..k+1} = SITPSEQ (Γ_{1..k+1}(bmc^k_E(S_0, T_A, p)), α_s)
        for (j = 1; j < k; j++)
            I_j = I_j ∧ I_j
            if (I_j ⇒ R_{j-1}) return PASS
            R_j = R_{j-1} ∨ I_j
        I_k = I_k
        if (I_k ⇒ 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 $\alpha_s = 0.5$), 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.

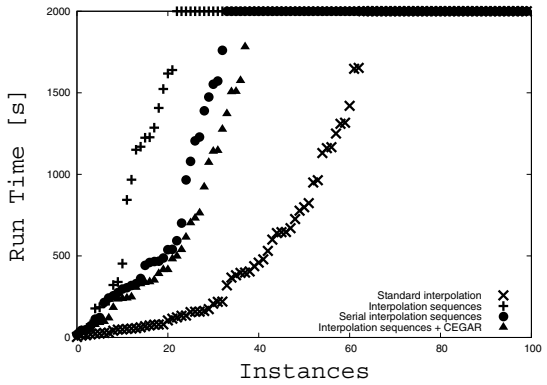

Fig. 6. Performance comparison between standard interpolation and interpolation sequences (possibly enhanced with serialization and CBA).

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, Time$_F$) and backward (d_B, 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) at overflow 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

[1] W. Craig, "Three Uses of the Herbrand-Gentzen Theorem in Relating Model Theory and Proof Theory," *The Journal of Symbolic Logic*, vol. 22, no. 3, pp. 269–285, 1957.

TABLE I

PERFORMANCE COMPARISON AMONG STANDARD INTERPOLATION (ITP), PARALLEL (ITPSEQ) AND SERIAL (SITPSEQ) INTERPOLATION SEQUENCES, AND INTERPOLATION SEQUENCES ENHANCED WITH CBA (ITPSEQCBA). ovf MEANS OVERFLOW ON TIME, A DASH (−) MEANS DATA NOT AVAILABLE.

Model			BDDs				ITP			ITPSEQ			SITPSEQ			ITPSEQCBA		
Name	#PI	#FF	d_F	Time$_F$	d_B	Time$_B$	Time	k_{fp}	j_{fp}	Time	k_{fp}	j_{fp}	Time	k_{fp}	j_{fp}	Time	k_{fp}	j_{fp}
bj08amba2g3f3	8	25	14	1	9	1	**89**	11	11	710	19	11	327	17	8	318	17	8
bj08amba2g4f3	12	35	–	ovf	–	ovf	**14**	11	0	14	11	0	22	11	0	28	11	0
eijkS820	18	58	11	1	4	22	**22**	4	10	125	13	9	237	14	10	174	13	9
eijkS832	18	62	11	1	4	17	**17**	4	10	131	13	9	264	14	9	248	14	10
eijkS953	16	105	11	2	3	22	**37**	4	10	67	13	9	131	13	9	220	15	9
intel006	345	350	–	ovf	9	62	**19**	9	15	ovf	(36)	–	1348	29	20	638	25	15
intel049	136	141	11	40	15	49	**21**	11	12	ovf	(27)	–	747	25	15	ovf	(29)	–
neclaftp3001	32	2826	–	ovf	–	ovf	**203**	14	0	480	14	0	1144	14	0	274	14	0
neclaftp3002	32	2826	–	ovf	–	ovf	427	16	0	643	16		1670	16	0	**425**	16	0
nusmvguidancep6	84	86	72	8	28	2	254	19	15	ovf	(33)	–	**240**	22	13	254	22	14
nusmvguidancep9	84	86	72	12	31	9	**4**	9	11	302	23	12	ovf	(28)	–	696	25	12
nusmvreactorp4	74	76	272	9	13	25	**8**	15	24	179	43	30	171	32	19	288	42	27
nusmvtcasp5	146	169	–	ovf	–	ovf	**54**	25	0	112	25	0	427	25	0	405	25	0
nusmvtcastp6	146	171	–	ovf	–	ovf	**11**	18	0	24	18	0	56	18	0	55	18	0
pdtviscoherence3	6	29	28	2	6	1	**10**	6	11	12	14	9	16	13	7	11	11	6
pdtviscoherence4	6	29	28	2	15	2	**29**	10	8	ovf	(25)	–	218	18	10	175	17	8
pdtviscoherence5	6	29	28	2	15	2	**23**	9	9	1781	25	10	425	20	8	160	17	8
pdtvisns2p0	16	75	16	3	10	10	**40**	7	15	607	19	10	241	16	5	416	17	5
pdtvisns2p2	16	73	16	3	9	4	**115**	11	12	ovf	(21)	–	ovf	(22)	–	680	19	10
pdtvisretherrtf4	3	43	–	1	–	3	**25**	33	0	637	33	0	712	33	0	673	33	0
pdtvissfeistel	68	296	–	ovf	–	ovf	**273**	21	11	329	23	10	438	24	9	ovf	(21)	–
pdtvisvending00	2	27	118	3	12	1	**14**	7	15	388	22	9	194	20	9	ovf	(28)	–
prodconsp1negnv	61	86	–	50	–	87	**36**	23	0	938	23	0	757	23	0	1158	23	0
prodconsp5	63	84	–	ovf	–	ovf	**42**	23	0	991	23	0	832	23	0	1296	22	0
texasPImainp01	14	47	16	1	36	4	**281**	19	29	1240	38	24	1352	34	21	ovf	(30)	–
viscoherencep3	6	29	28	1	14	2	63	10	12	ovf	(24)	–	**44**	14	6	137	17	7
viseisenberg	7	22	–	1	–	1	**10**	21	0	78	21	0	92	21	0	90	21	0
industrialA$_1$	26	151	12	35	28	74	**1**	15	9	160	37	21	106	27	9	79	27	11
industrialA$_2$	44	251	18	264	18	112	**53**	16	34	ovf	(52)	–	322	29	9	680	33	17
industrialA$_3$	44	251	–	ovf	20	347	**45**	16	20	ovf	(41)	–	1710	35	21	1221	29	15
industrialA$_4$	44	251	18	ovf	24	ovf	**117**	16	30	ovf	(39)	–	ovf	(35)	–	994	31	17
industrialB$_1$	402	782	–	ovf	–	ovf	ovf	(9)	–	ovf	(9)	–	ovf	(8)	–	**1726**	6	4
industrialB$_2$	395	771	–	ovf	–	ovf	659	4	3	**599**	5	2	712	5	2	907	5	2
industrialB$_3$	402	782	–	ovf	–	ovf	ovf	(9)	–	ovf	(9)	–	ovf	(8)	–	**1692**	6	3
industrialC$_1$	407	771	–	ovf	–	ovf	**210**	4	1	418	4	2	311	4	2	459	4	2
industrialC$_2$	266	607	–	ovf	–	ovf	690	6	8	ovf	(14)	–	219	9	7	**182**	8	6
industrialC$_3$	266	607	–	ovf	–	ovf	ovf	(8)	–	ovf	(14)	–	478	10	8	**357**	9	7
industrialD$_1$	71	99	–	ovf	–	ovf	**36**	9	17	831	29	19	285	21	13	226	20	12
industrialE$_1$	252	607	–	ovf	–	ovf	ovf	(9)	–	ovf	(9)	–	ovf	(10)	–	**1691**	16	10

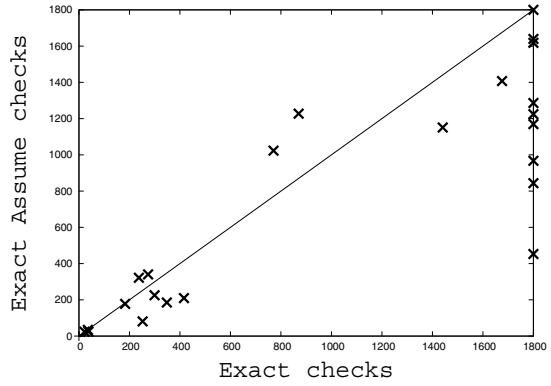

Fig. 7. Performance comparison between interpolation sequences adopting *exact-k* or *assume-k* checks.

[2] R. C. Lyndon, "An Interpolation Theorem in the Predicate Calculus," *Pacific Journal of Mathematics*, pp. 155–164, 1959.

[3] K. L. McMillan, "Interpolation and SAT-based Model Checking," in *Proc. Computer Aided Verification*, Boulder, CO, USA, 2003, pp. 1–13.

[4] K. L. McMillan and R. Jhala, "Interpolant-based Transition Relation Appproximation," in *Proc. Computer Aided Verification*, Edimburgh, Scotland, UK, 2005, pp. 39–51.

[5] J. P. Marques-Silva, "Improvements to the Implementation of Interpolant-based Model Checking," in *Proc. Correct Hardware Design and Verification Methods*, Edimburgh, Scotland, UK, 2005, pp. 367–370.

[6] J. P. Marques-Silva, "Interpolant Learning and Reuse in SAT-Based Model Checking," *Electronic Notes in Theoretical Computer Science*, vol. 174, no. 3, pp. 31–43, 2007.

[7] V. D'Silva, M. Purandare, and D. Kroening, "Approximation Refinement for Interpolation-Based Model Checking," in *Proc. Verification, Model Checking and Abstract Interpretation*, 2008, pp. 68–82.

[8] G. Cabodi, P. Camurati, L. Garcia, M. Murciano, S. Nocco, and S. Quer, "Trading-off SAT search and Variable Quantifications for effective Unbounded Model Checking," in *Proc. Formal Methods in Computer-Aided Design*, Portland, Oregon, USA, Nov. 2008, pp. 205–212.

[9] Y. Vizel and O. Grumberg, "Interpolation-Sequence based Model Checking," in *Proc. Formal Methods in Computer-Aided Design*, Austin, Texas, USA, Nov. 2009, pp. 1–8.

[10] G. Cabodi, M. Murciano, S. Nocco, and S. Quer, "Stepping Forward with Interpolants in Unbounded Model Checking," in *Proc. Int'l Conf. on Computer-Aided Design*, San Jose, California, Nov. 2006, pp. 772–778.

[11] F. Copty, L. Fix, R. Fraer, E. Giunchiglia, G. Kamhi, A. Tacchella, and M. Y. Vardi, "Benefits of Bounded Model Checking at an Industrial Setting," in *Proc. Computer Aided Verification*, Paris, France, Jul. 2001, pp. 435–453.

[12] E. Clarke, D. Kroening, N. Sharygina, and K. Yorav, "Predicate Abstraction of ANSI-C Programs Using SAT," *Formal Methods in System Design*, vol. 25, no. 2-3, pp. 105–127, 2004.

[13] N. Een, A. Mishchenko, and N. Amla, "A Single-Instance Incremental SAT Formulation of Proof and Counterexample Abstraction," in *Proc. Int'l Workshop on Logic Synthesis*, May 2010.

[14] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu, "Symbolic Model Checking Using SAT Procedures instead of BDDs," in *Proc. Design Automation Conference*, New Orleans, Louisiana, Jun. 1999, pp. 317–320.