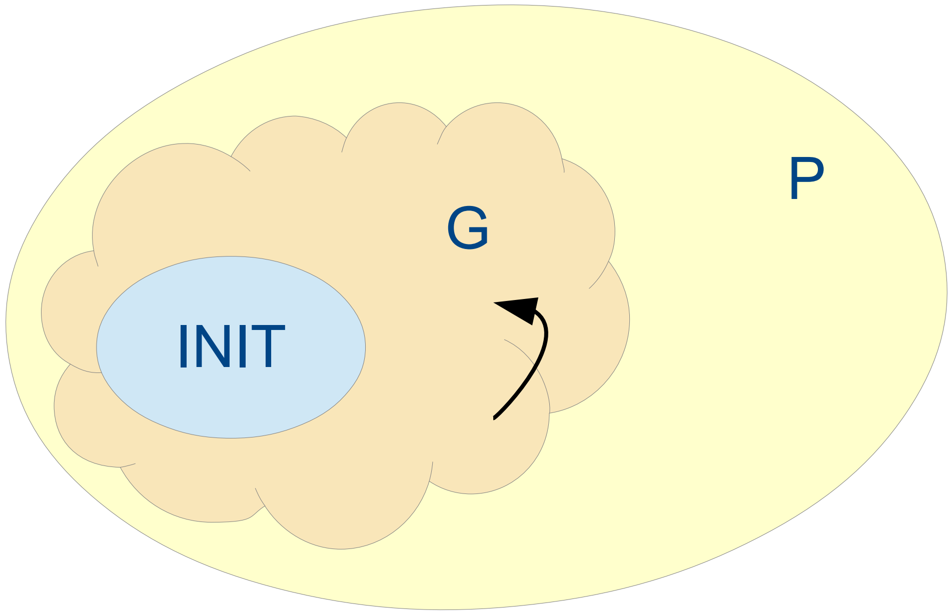
Small Inductive Safe Invariants

Alexander Ivrii, Arie Gurfinkel, Anton Belov

Introduction

- Consider a verification problem (INIT, TR, P)
- In the case that P holds, a Model Checker may produce a proof in terms of a safe inductive invariant
- A safe inductive invariant is a set of states G, satisfying:
  - G contains all the initial states
  - All the transitions from G lead back to G
  - G is contained in the set of states where P holds



Introduction

- Equivalently, a safe inductive invariant is a Boolean function G, satisfying:
  - $INIT \Rightarrow G$
  - $TR \wedge G \Rightarrow G'$ (inductive)
  - $G \Rightarrow P$ (safe)
- Following IC3, a recent trend is to produce such an invariant as a conjunction of many simple lemmas (such as clauses)
  - $G = C_1 \wedge \ldots \wedge C_n$
- A typical invariant may contain 10,000s of clauses

Introduction

- Our motivation is that smaller inductive invariants are more useful:
  - They are relevant in the context of FAIR [Bradley et al. 2011]
    - The cited paper introduces the problem and presents a solution
  - They produce better abstractions
    - A state variable not in the invariant is irrelevant for correctness
  - They increase user comprehension
  - They improve regression verification
- In this work we minimize inductive invariants by removing clauses
  - Look for minimal (or small) subsets
  - "Minimal" does not mean "of minimum size" (the latter is harder)

Problem Statement

- Following the standard (abuse of) notation for CNFs, we denote the conjunction of clauses as a set (and vice versa)

- Minimal Safe Inductive Invariants (MSIS): Given a safe inductive invariant $\{C_1, \ldots, C_n\}$, find a subset $\{C_{i1}, \ldots, C_{ik}\}$ of $\{C_1, \ldots, C_n\}$, so that:
  - $\{C_{i1}, \ldots, C_{ik}\}$ is also a safe inductive invariant
  - $\{C_{i1}, \ldots, C_{ik}\}$ is minimal (no proper subset of $\{C_{i1}, \ldots, C_{ik}\}$ is safe and inductive)

- We want the solution to be efficient (ideally the time to minimize a safe inductive invariant should be much smaller than to compute it)

Why finding an MSIS is not simple

- Recall that in particular we need to make sure that
  - $TR \wedge C_{i1} \wedge \ldots \wedge C_{ik} \Rightarrow C_{i1}' \wedge \ldots \wedge C_{ik}'$
- This query is non-monotone: each clause appears both as a premise and a conclusion
  - With fewer clauses, we need to prove less, but we can also assume less
- For example, it might be that:
  - $\{C_1, C_2, C_3, C_4\}$ is inductive,
  - $\{C_1, C_2, C_3\}$ is not inductive,
  - $\{C_1, C_2\}$ is inductive

Basic MSIS algorithm

- First, we present the approach described in [Bradley et al. 2011]
- The main idea is to tentatively remove a clause, and then to iteratively tentatively remove all no longer implied clauses, until:
  - Either a smaller inductive invariant is obtained
    - We can restrict to this smaller invariant
  - Or the property itself is no longer implied
    - We should restore all the tentatively removed clauses
- Repeat for every clause

Basic MSIS algorithm – Example

- Initially: $\{C_1, C_2, C_3, C_4, C_5, C_6\}$ is a safe inductive invariant for P
- Remove C_1 : $\{C_2, C_3, C_4, C_5, C_6\}$
  - Suppose that C_2' and C_4' are no longer implied
- Remove C_2 and C_4 as well (as they cannot be part of any MSIS of $\{C_2, C_3, C_4, C_5, C_6\}$) : $\{C_3, C_5, C_6\}$
  - Suppose that C_5' is no longer implied
- Remove C_5 as well : $\{C_3, C_6\}$
  - Suppose that C_6 and P are no longer implied
- It follows that C_1 cannot be removed (must be present in every MSIS of $\{C_1, C_2, C_3, C_4, C_5, C_6\}$)
- Restore all removed clauses

Basic MSIS algorithm – Example

- Currently:
  - $\{C_1, C_2, C_3, C_4, C_5, C_6\}$ is a safe inductive invariant for P
  - C_1 cannot be removed
- Remove C_2 : $\{C_1, C_3, C_4, C_5, C_6\}$
  - Suppose that C_3' and C_6' are no longer implied
- Remove C_3 and C_6 as well : $\{C_1, C_4, C_5\}$
  - Suppose that all remaining clauses and P are implied
- It follows that $\{C_1, C_4, C_5\}$ is a smaller safe inductive invariant

Basic MSIS algorithm – Example

- Currently:
  - $\{C_1, C_4, C_5\}$ is a safe inductive invariant for P
  - C_1 cannot be removed
- Proceed with the remaining clauses in a similar fashion

Basic MSIS algorithm

- Denote by MaxInductiveSubset(S, P) the procedure that computes the maximum inductive subset of S, aborting if it does not imply P
- Given a safe inductive invariant G for P, in the basic approach we
  - Iteratively
    - Choose a not-yet-considered clause C in G
    - Compute X = MaxInductiveSubset(G\C, P)
    - If X is safe (X implies P), then replace G by X
- Claim: the described algorithm computes an MSIS of G

- Unfortunately, this algorithm is not efficient
  - A large number of SAT calls is required (~quadratic)
  - Does repeated work

What can we do better?

- Efficiently under-approximate an MSIS
  - Find clauses that must be present in any MSIS of G
- Efficiently over-approximate an MSIS
  - Remove clauses that are not part of some MSIS of G
- Optimize the basic MSIS algorithm
  - Minimizing the amount of wasted work
  - Taking clause dependency into account
- Combine under- and over- approximations with the optimized MSIS algorithm

Under-Approximation

- Given a safe inductive invariant $G = \{C_1, \ldots, C_n\}$, we say that a clause C_i is safe necessary if C_i is present in every MSIS of G.
- We exploit the following observations:
  - Given a clause C in G, if $(G \setminus C) \wedge TR \Rightarrow P$ does not hold then C is safe necessary
  - Given a clause C in G and a safe necessary clause D (different from C), if $(G \setminus C) \wedge TR \Rightarrow D'$ does not hold then C is safe necessary
- The under-approximation algorithm iteratively applies the above two observations until fix-point
- The algorithm can be implemented very efficiently using an incremental SAT-solver

Under-Approximation – Example

- Initially:
  - $\{C_1, C_2, C_3, C_4, C_5, C_6\}$ is a safe inductive invariant for P
  - No clauses are marked as necessary
- Check if there is an unmarked clause without which P is not implied
  - Suppose that we find C_4
  - Mark C_4 as necessary
- Check if there is an unmarked clause without which P is not implied
  - Suppose that we find C_5
  - Mark C_5 as necessary
- Check if there is an unmarked clause without which P is not implied
  - Suppose that we find none

Under-Approximation – Example

- Check if there is an unmarked clause without which C_4' is not implied
  - Suppose that we find C_1
  - Mark C_1 as necessary
- Check if there is an unmarked clause without which C_4' is not implied
  - Suppose that we find none
- Check if there is an unmarked clause without which C_5' is not implied
  - Suppose that we find none
- Check if there is an unmarked clause without which C_1' is not implied
  - Suppose that we find none

- Therefore: C_1, C_4, C_5 belong to every MSIS of $\{C_1, C_2, C_3, C_4, C_5, C_6\}$

Under-Approximation

- Claim: the described algorithm computes a set of clauses that must be present in every MSIS of G

  (however, it does not compute all such clauses)

- The algorithm makes only a linear number of SAT calls (even in the size of the solution)

- The algorithm can be further improved if some clauses are initially known to be necessary

- For IC3 proofs, the algorithm is very efficient and usually marks a large number of clauses

Over-Approximation

- Given a safe inductive invariant $G = \{C_1, \ldots, C_n\}$ and two subsets A and B of G, we say that A inductively supports B (or equivalently that B is supported by A) if $TR \wedge A \wedge B \Rightarrow B'$
- Greedily compute a safe inductive subset of G as follows:
  - Choose any minimal subset A_1 of clauses needed to support P (and any necessary clauses, if known)
  - Choose any minimal subset A_2 of clauses needed to inductively support A_1
  - Choose any minimal subset A_3 of clauses needed to inductively support A_2

    ...
  - Stop when the last computed set is empty
- The over-approximation is the union of all the sets considered

Over-Approximation

- Claim: the described algorithm computes a safe inductive subset of G (however, it is not guaranteed to be minimal)

- The algorithm makes only a linear number of MUS calls

- The quality and the run-time of the algorithm are greatly improved
  - If we compute minimal supporting sets
  - If we follow the presented recursive approach
    - Instead of computing a global unsatisfiable core as suggested in [Bradley et al. 2011]
  - If we consider all the clauses of A_i together, rather than 1-by-1
  - If some of the clauses are initially marked as necessary

Optimized MSIS algorithm

- An immediate optimization to the basic MSIS algorithm consists of
  - Marking necessary clauses as soon as they are discovered, and
  - Aborting the computation as soon as one of the necessary clauses becomes non-implied
- Given a safe inductive invariant G for P, in the optimized approach we
  - Keep track of necessary clauses N
  - Iteratively
    - Choose a not-yet-considered clause C in G\N
    - Compute X = MaxInductiveSubset(G\C, P∪N')
    - If X is safe, then replace G by X
    - Otherwise, add C to N

Optimized MSIS algorithm – Example

- Consider the previous example:
  - $\{C_1, C_4, C_5\}$ is a safe inductive invariant for P
  - C_1 cannot be removed
- Remove C_4 : $\{C_1, C_5\}$
  - Suppose that C_1' is no longer implied
  - The basic algorithm removes C_1
  - The optimized algorithm aborts immediately
- Remove C_5 : $\{C_1, C_4\}$
  - Suppose that C_4' is no longer implied
  - The basic algorithm removes C_4 (and then possibly C_1, etc)
  - The optimized algorithm aborts immediately

Optimized MSIS algorithm

- The optimized algorithm is significantly better than the basic algorithm
- Moreover, the optimized algorithm is significantly improved when some of the clauses are initially marked as necessary
- However, the optimized algorithm still requires a quadratic number of SAT queries in the worst case:
  - Queries of the form "which clauses become not implied if certain other clauses are removed?"
  - Each time that we remove a clause C_i from a safe inductive invariant, might need to make a linear number of such queries
  - Might need to process a linear number of clauses

B.I.G. MSIS algorithm

- The B.I.G. algorithm makes use the following observation: given a safe inductive invariant G and a clause C
  - Either G \ C remains a safe inductive invariant
  - Or C is safe necessary for P or for some other clause in G
- The B.I.G. algorithm makes only a linear number of SAT queries
- The technique is inspired by the Binary Implication Graphs used in SAT-solvers
- Purely by coincidence, B.I.G. also represents the authors' initials ;-)

B.I.G. MSIS algorithm – Example

- Initially: $\{C_1, C_2, C_3, C_4, C_5, C_6\}$ is a safe inductive invariant for P
- Remove C_1 : $\{C_2, C_3, C_4, C_5, C_6\}$
  - Suppose that C_4' is no longer implied (and possibly other clauses)
  - We infer: C_1 is needed for C_4
    - Equivalently: if C_4 is in the invariant, then C_1 is in the invariant
  - Denote this graphically by $\{C_1\} \rightarrow \{C_4\}$
- Restore C_1 and remove C_4 : $\{C_1, C_2, C_3, C_5, C_6\}$
  - Suppose that C_5' is no longer implied (and possibly other clauses)
  - We infer: C_4 is needed for C_5
  - Denote this graphically by $\{C_1\} \rightarrow \{C_4\} \rightarrow \{C_5\}$ (note transitivity)
- Restore C_4 and remove C_5

B.I.G. MSIS algorithm – Example

- Currently:
  - C_5 is tentatively removed: $\{C_1, C_2, C_3, C_4, C_6\}$
  - Know: $\{C_1\} \rightarrow \{C_4\} \rightarrow \{C_5\}$

- Case I: P and all remaining clauses are still implied
  - In this case, can permanently remove the (last) clause C_5
  - Know: $\{C_1\} \rightarrow \{C_4\}$
  - Make the query for C_4

B.I.G. MSIS algorithm – Example

- Currently:
  - C_5 is tentatively removed: $\{C_1, C_2, C_3, C_4, C_6\}$
  - Know: $\{C_1\} \rightarrow \{C_4\} \rightarrow \{C_5\}$

- Case II: P (or one of known necessary clauses) is not implied
  - In this case, all of the clauses C_1, C_4, C_5 are necessary
  - Make the query for some new clause

B.I.G. MSIS algorithm – Example

- Currently:
  - C_5 is tentatively removed: $\{C_1, C_2, C_3, C_4, C_6\}$
  - Know: $\{C_1\} \rightarrow \{C_4\} \rightarrow \{C_5\}$

- Case III: A new clause (for example C_6) is not implied
  - Infer: C_5 is needed for C_6
  - Know: $\{C_1\} \rightarrow \{C_4\} \rightarrow \{C_5\} \rightarrow \{C_6\}$
  - Make the query for C_6

B.I.G. MSIS algorithm – Example

- Currently:
  - C_5 is tentatively removed: $\{C_1, C_2, C_3, C_4, C_6\}$
  - Know: $\{C_1\} \rightarrow \{C_4\} \rightarrow \{C_5\}$

- Case IV: A previous clause (for example C_4) is not implied:
  - Either
    - All clauses between C_4 and C_5 are in the final invariant
    - None of the clauses between C_4 and C_5 are in the invariant
  - Know: $\{C_1\} \rightarrow \{C_4, C_5\}$
  - Make the query for $\{C_4, C_5\}$

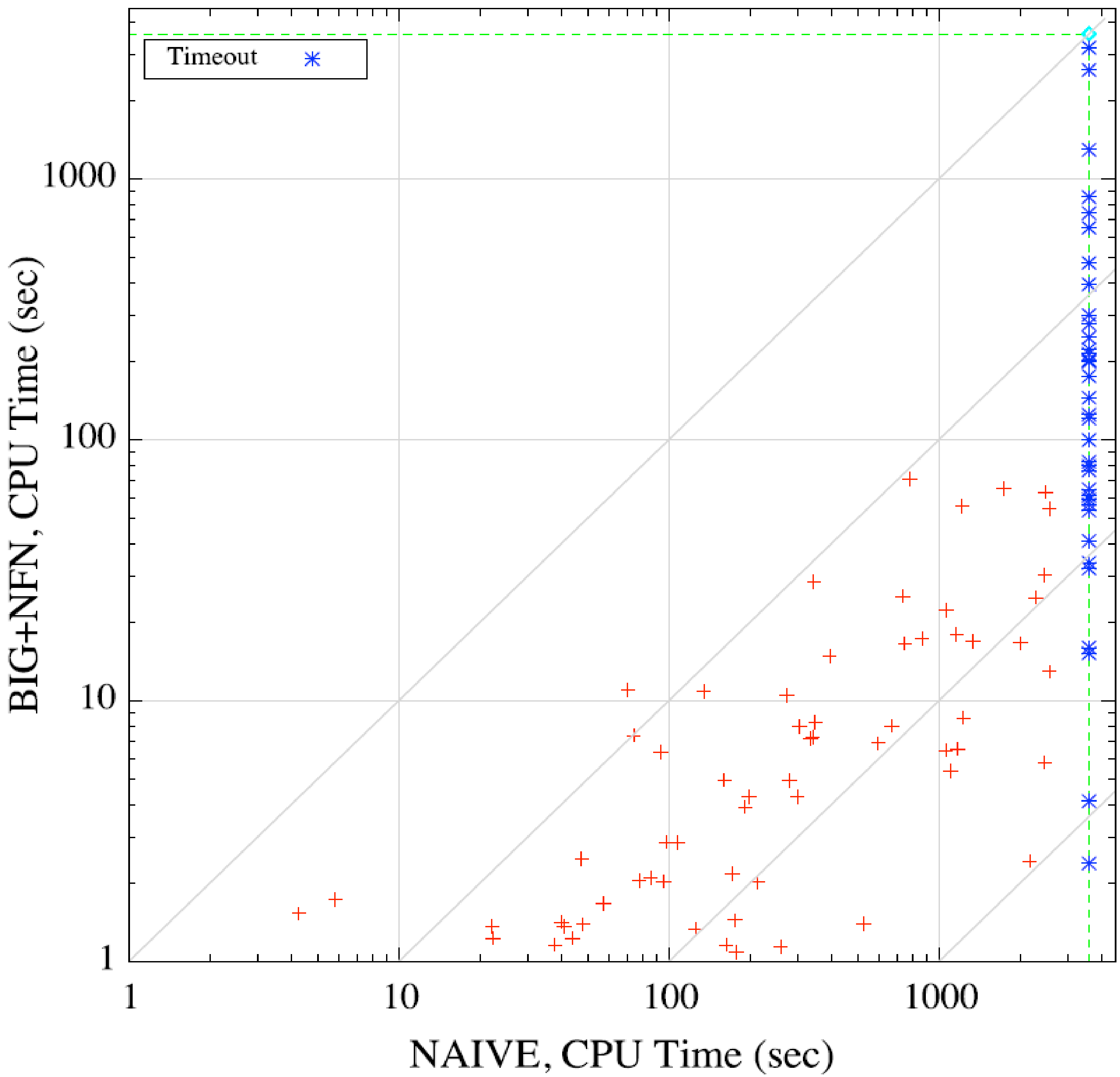
Combined MSIS algorithm

- Experimentally the following combination of the presented ideas works the best

1) Run under-approximation
   - About 70% of the final MSIS clauses are identified in this stage
2) Run over-approximation (with marked necessary clauses)
   - After this stage over-approximates the final MSIS by only 4%
   - In many cases already produces an MSIS
3) Run under-approximation (on the reduced invariant)
   - About 90% of the final MSIS clauses are identified
4) Run Optimized MSIS or B.I.G. MSIS on the remaining clauses

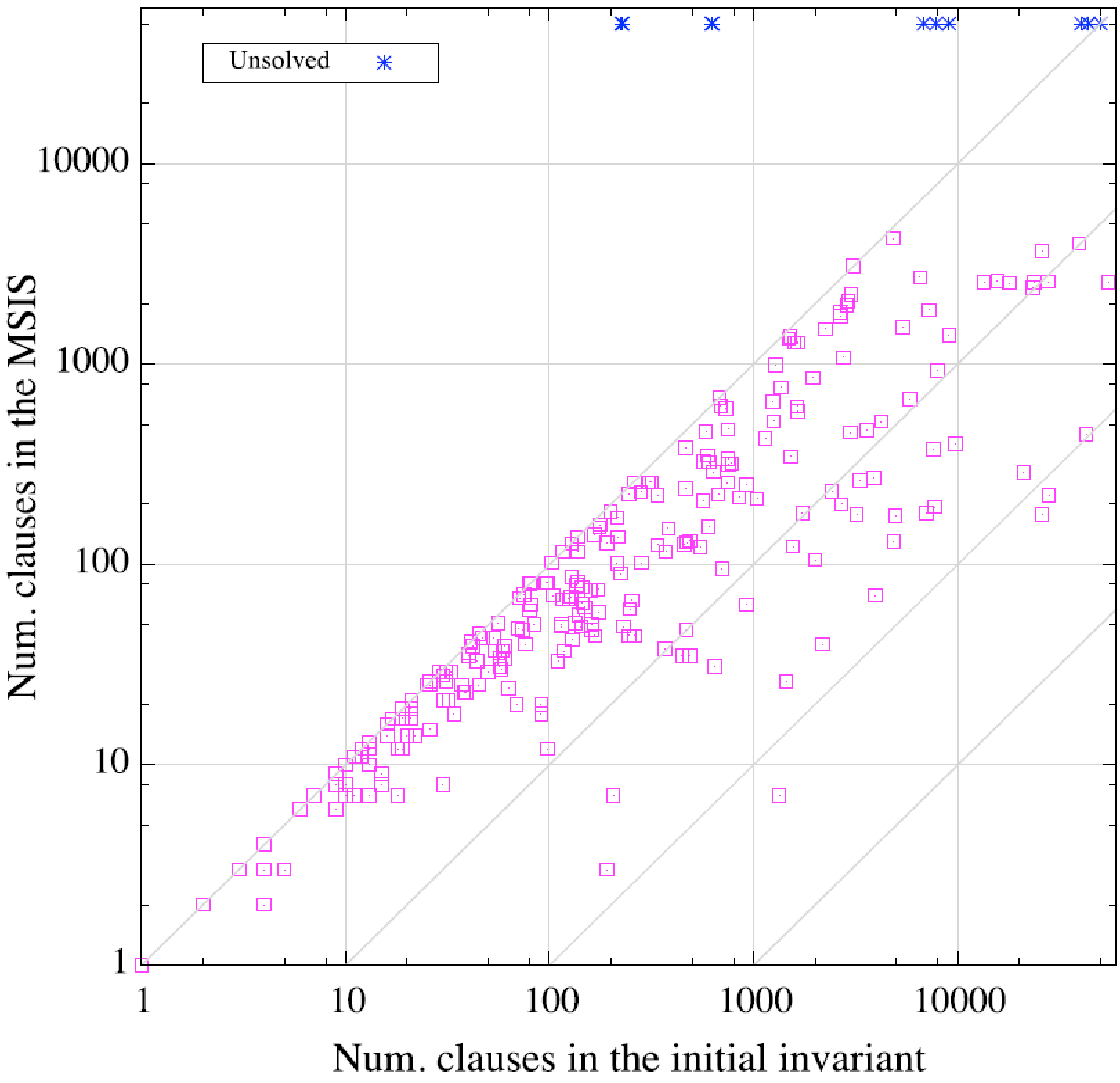
- On average improves the basic MSIS algorithm by 10 to 1000 times

Overall Improvement in Run-Time



Thank You!

Reduction in the Number of Clauses



Under-Approximation – Implementation

- Introduce an auxiliary variable a_i for every clause C_i of G
- Load $TR \wedge (a_1 \Leftrightarrow C_1) \wedge \ldots \wedge (a_n \Leftrightarrow C_n)$ into the solver
- Encode the constraint "at most one out of $\neg a_1, \ldots, \neg a_n$ is true"
- Keep unprocessed elements in a queue Q, initially $Q = \{P\}$
- Iteratively:
  - Consider the first element q in Q
  - Solve, passing $\neg q$ as assumptions
  - If SAT:
    - Exactly one of the a_i evaluates to false
    - Mark the corresponding C_i as necessary and set a_i = true
    - Add C_i' to Q
  - If UNSAT:
    - Proceed to the next element in Q