Property Guided Secure Configuration Space Search

You Li*$^{1\,[0000-0003-2632-3804]}$, Kaiyu Hou*$^{2\,[0000-0002-7551-1345]}$,
Yunqi He$^{1\,[0000-0002-1108-089X]}$, Yan Chen$^{1\,[0000-0003-4103-1498]}$, and
Hai Zhou$^{1\,[0000-0003-4824-7179]}$

[1] Northwestern University, USA
[2] Alibaba Cloud, China

Abstract. Complex reactive systems such as 5G cellular networks must have flexible configuration options to fit different deployment scenarios. However, not every possible configuration combination is risk-free. Some of them may lead to availability issues or even security vulnerabilities. Asking the system engineers to check each configuration via model checking for every deployment or re-configuration is impractical if not impossible.
In this paper, we propose the concept of secure configuration space and develop a symbolic model checking algorithm, INCISE, to compute a large configuration space for a given reactive system. Such a space will be characterized by a logical condition (*e.g.*, a Boolean formula). A system engineer can check any candidate configuration against the condition with a single SAT query to know whether it is secure. The target properties could be general safety and liveness properties. The algorithm enjoys the same benefits including efficiency and expressiveness as modern symbolic model checkers. We demonstrate the algorithm's performance on industrial benchmarks and leverage it to address security issues in cellular network protocols.

Keywords: formal analysis, network system and protocol, symbolic model checking, secure configuration space

1 Introduction

Model checking is a powerful tool to verify systems. It can either provide solid proof that the system is correct and secure with regard to a set of properties, or find a counterexample that demonstrates the system indeed violates the properties. Such a process can help detect design errors, mitigate attacks, or certify that the design is correct.

There is an implicit assumption for traditional model checking: for all reachable states, the system should *always* be correct and secure. This is usually the case for hardware systems. For instance, an integrated circuit should always follow the specified behaviors regardless of the inputs. However, for some other systems, this is not true.

Network protocols and network systems are highly complex and flexible. A large number of configurable parameters are left open by the protocol designers to the system engineers and administrators. The purpose of doing so is to allow better flexibility to fit

* Contributed equally.

different circumstances. Nevertheless, not every configuration is correct and secure. Only with a high level of expertise can the system engineers be confident of their configurations. In addition, network protocols usually have multiple participants. One participant can check if its own system is correctly configured, yet the correctness of the entire system involving all participants is hard to guarantee. Behaviors of reactive systems may also depend on the environment. The designers may not be able to foresee all potential environments that can affect the system's execution. A system working well under one circumstance may have vulnerabilities under others.

It is highly desirable for the system engineers to run verification before those complex systems are released to customers. On the other hand, network deployment should be agile: to be adjusted responsively and to be deployed in real-time. It is impractical for those engineers to wait on-site for the time-consuming verification process to finish before they can actually deploy them. To make things worse, if one configuration or environment is proved unsafe, the entire process needs to be iterated again.

Does there exist an approach to execute the time-consuming model checking process beforehand? Traditional model checking problem is a decision problem: given a formal model $\mathbb{M}$, a set of correctness properties P and an initial condition I, it decides whether $(\mathbb{M}, I) \models P$. One can encode the configurations or the environments into I and run model checking for every possible pattern. Unfortunately, because the number of patterns grows exponentially as the number of configurable parameters grows, this approach is not affordable.

In this paper, we create the concept called secure configuration space: any configurations lying in the space are guaranteed to be secure. We then propose an algorithm named INCISE (**IN**cremental **C**onf**I**guration **S**pace s**E**arch). It translates the *decision problem* into a *search problem*: given a formal model $\mathbb{M}$ and a set of correctness properties P, the algorithm finds a secure configuration space in a concise Boolean formula. The system engineers can now query if a candidate configuration or an environment is within the space with a single SAT query, which can be finished in a negligible amount of time. We concentrate on finite-state transition systems, but the same methodology can be generalized to infinite-state transition systems [19, 25]. Because our proposed algorithm is based on cutting-edge model checking algorithms, it enjoys high performance and expressiveness, while its results still have rigorous mathematical guarantees.

The proposed approach differs from precondition inference [22] and preimage computation [39] in three main aspects. *i)* Our algorithm can deal with the rich languages of LTL and CTL properties. *ii)* Our algorithm is fully symbolic and thus scales to large systems. *iii)* Our algorithm is incremental and thus can easily adapt to various use cases. When real-time checking is required, the algorithm can return a partial secure configuration space, which is a sound approximation of the whole space of secure patterns. When full coverage is preferred, it also initiates a second stage that iteratively enlarges the found space until convergence.

Contributions: The contributions of our paper include:

- We propose the concept of secure configuration space and demonstrate its strength on network system verification and deployment.
- We design an efficient secure configuration space search algorithm, INCISE, to enhance the capability of state-of-the-art model checking algorithms. We develop

techniques to enlarge secure configuration spaces. The results guarantee soundness and will approximate the maximal secure configuration space.
- We demonstrate ways to encode configurations into formal specifications and leverage the found secure configuration space for validation and correction.
- We investigate the cellular emergency call system with our approach. Besides, experiments on industrial benchmarks show that our algorithm achieves a 55.7% improvement in space coverage compared with other invariant-based model checking algorithms at a minimal overhead.

2 Motivation

The demand for computing the secure configuration space emerges when we tried to integrate formal methods into network studies. This section explains how the concept of secure configuration space can be applied to network protocols. We use the cellular emergency call system as an example.

The cellular emergency call system plays an essential role in public safety and thus deserves a careful analysis by formal methods. Nevertheless, the critical mission in this analysis is not to prove that the system is *always* secure and reliable. In fact, it is not: complaints about the system's availability can easily be found online [28]. Existing formal verification tools can point out some situations where the system is insecure. Instead, we were expecting to figure out a practical way to mitigate vulnerabilities and prevent potential attacks.

Traditional model checking deals with systems which have fixed initial conditions. However, during our collaboration with major cellular network carriers and network device manufacturers, we realized that it is impractical to provide only one single configuration for the system. Cellular network system involves multiple participants. Different carriers have divergent concerns when configuring their systems. As for emergency calls, each country has its local emergency numbers, regulations, and historical settings. All those differences make a universal configuration impossible. Meanwhile, cellular network protocols keep evolving. Major upgrades take place in around ten years, while standard organizations publish minor revisions every three months. A single unified configuration may not meet the requirements of new versions.

From the view of formal verification researchers, we believe a generic way to fulfill the carriers' requirements is to provide a configuration space: one configuration is guaranteed to be secure if it lies in this space. The need for secure configuration space also exists in other network protocols. For instance, system administrators of routing protocols (*e.g.*, BGP and OSPF) would like to know routing policy configurations' boundaries, where converged routing tables are always secure.

We have collected the emergency call system configurations of three world-leading carriers. Their configurations differ in many ways, yet running model checking can easily find violations to the basic requirements specified in the protocols. Meanwhile, it is not straightforward to fix those violations. For example, one carrier noticed a potential vulnerability in the deployed system: some emergency calls from their subscribers cannot be properly routed. This carrier attempted to mitigate the issue by modifying system configurations. With model checking, we found that the modification introduces another problem: roaming users cannot dial emergency numbers when the screen is locked.

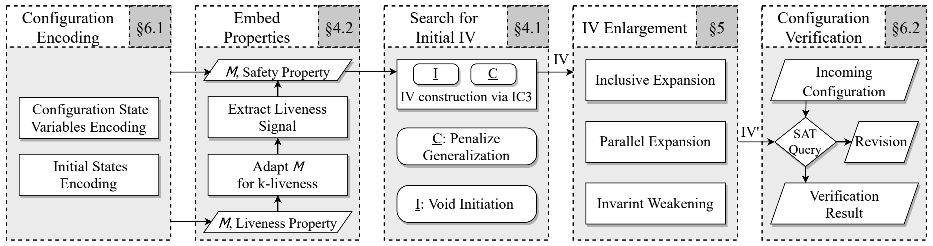

Fig. 1. Architecture of INCISE.

Therefore, it is not practical to trial-and-error every possible configuration pattern, and a more systematic method is urgently needed. We believe our work on secure configuration space search can address this challenge. We will show more details about this motivating example in §8.

3 Preliminaries

A finite state transition system $\mathbb{M} : (\bar{i}, \bar{x}, I, Tr)$ is described by the primary inputs $\bar{i}$, the state variables $\bar{x}$, the initial condition $I(\bar{x})$, and the transition relation Tr. $Tr(\bar{i}, \bar{x}, \bar{x}')$ is a mapping from the input values and the current state to the next state. The next state is denoted by the primed state variables $\bar{x}'$.

A *state* s is a Boolean assignment to all state variables. Specifically, it is a *cube* which is a conjunction of literals, and each *literal* is either a variable or the negation of the variable. A state either satisfies a formula: $s \models E$, or falsifies it: $s \not\models E$. If $s \models E$, we call that s is an E-state. The set of all E-states is denoted as E-set. The negation of a cube is a *clause*, which is a disjunction of literals. A *trace* is a sequence of states $s_0, s_1 \cdots$, where any adjacent pairs of states (s_i, s_{i+1}) is a valid transition: $\exists \bar{i} : (\bar{i}, s_i, s'_{i+1}) \in Tr$, and $s_0 \in I$.

During model checking, the model is checked in conjunction with a set of correctness properties. Correctness properties are formulas describing those specifications the model should meet. Safety properties specify that some bad things should never happen. Liveness properties specify that some good things should eventually happen. A *secure state* is such a state that once reached, the model will always satisfy the correctness properties from then on. A model is secure if all its *initial states* are secure states.

4 INCISE: the Algorithm of Secure Configuration Space Search

Conventional network system verification uses model checking in such a way: a configuration is conjoined with the specification to check whether it is secure. If it turns out to be insecure or suffers from attacks, the engineer makes changes until the model is proven to be correct.

We suggest an alternative method. The model checking algorithm is executed in advance to figure out the safe configuration space. The engineer can then query any particular configuration to check if it resides in the safe configuration space, or select the closest safe configuration if the current one is unsafe. The former step is executed only once, so any duplicated efforts are avoided. The latter step has a minimal computational burden and can be finished in real-time.

Searching for the secure configuration space is not a trivial task. There is an exponential number of assignments to all configuration variables, so it is impractical to enumerate all assignments to check whether they satisfy the correctness properties. We address this challenge by proposing a new algorithm, INCISE. The procedures of applying INCISE on system verification are shown in Fig. 1 and are detailed in sections (§4-§6). The paper is organized logically for clarity, and that differs from the chronological order of Fig. 1.

Informally, finding the secure configuration space requires *i)* to find a close to minimal Boolean representation, *ii)* to approximate the underlying maximal secure state space, and *iii)* to determine the security of states with model checking. INCISE addresses the three objectives simultaneously. It searches for a large set of secure states during model checking, while attempting to minimize the returned Boolean formula.

Detailed discussions on how it efficiently searches for the secure configuration space about safety and liveness properties are presented in §4.1 and §4.2, respectively.

4.1 Secure Space Search for Safety Properties

Secure Space Search as Inductive Model Checking A safety property P specifies that some bad things should never happen. Given the initial conditions, the model checking problem can be treated as a reachability problem: whether a $\neg P$-state is reachable from an I-state. If it does, a trace that connects those two states is a counterexample that refutes the safety property.

Otherwise, the model checking algorithm needs to construct a proof for unreachability. One symbolic approach is to build an inductive proof. Specifically, it finds an *inductive invariant* IV, such that the IV-set includes all initial states (1), IV itself is inductive (2), and the IV-set has no intersection with the $\neg P$-set (3):

$$I \Rightarrow IV, \tag{1}$$

$$IV \wedge Tr \Rightarrow IV', \tag{2}$$

$$IV \Rightarrow P. \tag{3}$$

There must exist at least one such IV if the model satisfies the safety property.

It follows that *every IV-state is a secure state*, because $s \models P$ and there exist no traces from s to $\neg P$-states. So the first step to search for the secure state space is to find an IV.

Searching Inductive Invariants with IC3 There are many approaches to find such an invariant [14,15,32,33,36,38,42]. Among those, we choose IC3 [14] as our starting point because *i)* it adopts symbolic techniques such as inductive generalization and clause propagation thus has been proved to be efficient [24], and *ii)* it searches backward from $\neg P$, thus tends to find a larger IV if multiple IVs exist. A larger IV contains more secure states.

IC3 is a symbolic model checking algorithm. It aims to construct an IV incrementally. During the execution, IC3 maintains a sequence of *frames* $F_0, F_1, \cdots, F_k$, where $F_0 = I$.

A frame F_i consists of a set of clauses, denoted as F_i-clauses. Any clauses belonging to a frame should also belong to the previous frames: F_{i+1}-clauses $\subseteq F_i$-clauses for $0 \leq i < k$. It indicates that the F_i-set is a subset of the F_{i+1}-set. Moreover, every frame should be *inductive relatively* to its previous frame: $F_{i-1} \wedge Tr \Rightarrow F_i'$. An IV is found and the algorithm terminates if any two adjacent frames meet with each other: when $F_i = F_{i+1}$, $F_i \wedge Tr \Rightarrow F_i'$, and thus F_i becomes an invariant.

In every iteration, the algorithm starts to query: $F_k \wedge Tr \Rightarrow P'$. If the query fails, there must exist a state s^*, such that $s^* \wedge Tr \Rightarrow \neg P'$. Therefore, s^* must be excluded from IV as well as from F_k. The negated clause $c^* = \neg s^*$ is guaranteed to be inductive relative to at least one previous frame [14]. Let F_m be such a frame with maximal i. IC3 then runs inductive generalization: it finds a *minimal inductive subclause* $c_{min} \subseteq c^*$ by dropping as many literals as possible in c^*. During this process, c_{min} should satisfy

- *initiation*: $I \Rightarrow c_{min}$, and
- *consecution*: $F_m \wedge c_{min} \wedge Tr \Rightarrow c'_{min}$ (inductive relative to F_m)

This generalization helps to exclude a large number of states, instead of a single state, at a time. c_{min} is then conjoined with each of $F_0, F_1, \cdots, F_m$.

The above process may break the relatively inductive relations between adjacent frames. In that case, IC3 continues with the same process on a predecessor state t^* of s^*. An iteration finishes when no more s^* can be found, and the relative inductive relation is reconstructed. Then a new frame $F_{k+1} = P$ is added to the sequence of frames to start a new iteration.

If initiation fails in any processes, there must be a trace from an I-state to a $\neg P$-state. IC3 returns the counterexample and disproves the property P.

Initial-blind Invariant Construction A model checking problem decides whether $(\mathbb{M}, I) \models P$. Its result is depending on both the model and the corresponding initial condition. In comparison, for the secure space search problem, models are not classified into safe and unsafe ones. Instead, a model partitions its state space as secure and insecure. The model should satisfy the target property if all its assigned *initial states* were secure.

A secure space search algorithm should be agnostic to initial conditions. The original IC3 algorithm *i)* keeps track of traces that can reach $\neg P$-states through s^* and t^*. When the I-set intersects with any of those states, IC3 terminates and concludes the model is unsafe. *ii)* Its inductive generalization process queries initiation when it attempts to drop a literal. Hence, all clauses in any frames should satisfy initiation, *i.e.*, containing the whole I-set.

We deactivate the initial conditions to bypass *i)* and *ii)*. The I-set thus becomes an empty set, and any IV-clauses are now independent of the initial conditions. With this modification, the algorithm will not be terminated by counterexamples, but it is still guaranteed to terminate when two adjacent frames meet. Specifically, k is monotonically increasing while the total number of states is finite, there must be two identical frames as soon as k is greater than the number of states. In the worst case, the foremost frames are monotonically strengthened and eventually converge to the empty set.

A drawback of this approach is the over-generalization of the minimal inductive subclauses. Originally, the generalizations of F_1-clauses are limited by the initial conditions through initiation, and a previous frame limits the generalizations of the clauses in every subsequent frame through consecution. Having no constraints in the initial condition voids initiation and could result in stronger clauses that exclude more states. Stronger clauses mean smaller frames, and that in turn weakens the consecution constraints for all subsequent frames. A model can have multiple IVs. The drawback above increases the chance of finding a relatively small IV, even the empty set as

We limit the extent of generalization, thus mitigating the drawback mentioned above and improving the chance of finding a relatively larger IV. We introduce a method that penalizes over-generalization while still maintaining efficiency. Every time a literal is dropped, we let a growing chance to terminate the current generalization and return the current inductive subclause. Consecution remains to be a hard constraint to generalization. Such a method tends to produce weaker clauses. Recall that in every iteration, the newly added frame, F_{k+1}, is set to P. In other words, the algorithm searches backward from the largest possible secure space. As all frames consist of weaker clauses, they tend to move closer to P during the early iterations. As a result, the algorithm has a greater opportunity to return an IV that covers more states.

When the computed IV is used by system engineers, the cost of storing and querying it is minimal when compared with the cost of IV construction. IV is derived from one frame, and the algorithm maintains a multiple of frames during execution. Additionally, each frame is queried by the SAT solver at least once in every iteration of the algorithm.

4.2 Secure Space Search for Liveness Properties

A liveness property specifies that some good things, described by a liveness condition, should eventually happen. It can be equivalently formulated as that a liveness event q should eventually always hold ($\Diamond\Box q$). Several efficient symbolic model checking algorithms have been proposed for liveness checking, including FAIR [9], LTS [16], and k-liveness [20]. Safety properties are theoretically easier to check than liveness properties [20]. LTS and k-liveness reduce liveness checking to safety checking. The original liveness property holds if the corresponding safety condition is verified. We choose k-liveness as our base algorithm. k-liveness is the state-of-the-art liveness checking algorithm with the best efficiency among the above-mentioned algorithms [20]. More importantly, the safety secure state space search algorithm proposed in §4.1 can still be applied after the liveness to safety conversion by k-liveness.

Fundamentally, k-liveness makes use of a simple observation: if a liveness event q will eventually always hold, the number of $\neg q$-states along any traces must be bounded by a constant k. So the remaining challenge is to find the value of k.

k-liveness runs an iterative decision process to find k. In each iteration, one additional absorbing circuitry is attached to the q event signal [20]. This circuitry can eliminate the foremost $\neg q$-state along all paths from I-states. Then the algorithm checks a safety property: event q holds forever along all traces ($\Box q$). The property is satisfied when every $\neg q$-state along each path is absorbed. This process will repeat k times until the liveness property is proved.

However, there is a caveat when directly applying the safety secure state search algorithm to the k-liveness result: every single state, when set to the initial state, can have a different k value. For this reason, the k value corresponds to one initial state does not necessarily cover all secure states. To address this challenge, we observe that k copies of absorbing circuitries can still absorb less than k $\neg q$-states. Let k_i be the k value when state s_i is set to be the initial state. With k_i copies of absorbing circuitries added, the corresponding secure state space will also contain state s_j, if $k_j \leq k_i$.

But another state, s_l, is not included if $k_l > k_i$. Our algorithm addresses this issue by computing a k_{max}, which should be sufficiently large to cover most of the secure states. It selects a set of random initial states, each of them is a secure state to the liveness property. By assigning k_{max} as the greatest k among n randomly selected initial states, regardless of the distribution, the secure state space corresponds to k_{max} covers $\frac{n}{n+1}$ of all secure states in estimate [34].

It is unnecessary to query for k from scratch when running k-liveness for each initial state. Instead, we can always maintain the current k_{max} as well as the model with k_{max} copies of absorbing circuitries. The model only needs to be incrementally updated when an incoming state requires a larger k.

5 Enlargement of Secure Configuration Space

5.1 Maximal Secure State Space

The algorithm proposed in §4 attempts to find a large IV, but a maximal or close-to-maximal IV is always desirable if the timing budget allows. The maximal IV always exists and it contains exactly all secure states. According to the definition, $\neg P$ is never reachable from a secure state and is reachable from an insecure state. There exist no valid transitions from the set of all secure states to the insecure ones, and the maximal IV interpolates these two sets.

We propose three techniques that can further expand the IV found by INCISE. These techniques can be initiated in turn until converging to the maximal IV or when the timing budget is exhausted.

5.2 Inclusive Expansion

This technique aims to find a larger IV, which includes and expands the current IV. In specific, the new IV interpolates the current non-maximal IV and the underlying maximal one. This is enforced by executing the algorithm in §4.1 (the basic INCISE) again with one exception: setting the current IV as the initiation constraint. Hence, all new clauses generated in the execution contains the current IV, and so is the resulted new IV. For example, as shown in Fig. 2, suppose IV_1 is the current IV found by the basic INCISE. IV_2 could be the resulted new IV by setting IV_1 as the initiation constraint and executing INCISE again.

Is it possible to apply this technique iteratively until converging to the maximal IV? As the current IV expands, it becomes an even stronger constraint for inductive generalization. IC3 relies on inductive generalization to accelerate convergence. Thwarting

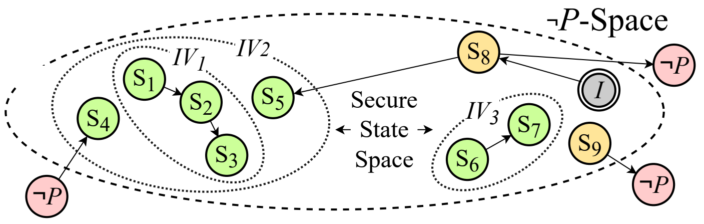

Fig. 2. Techniques to enlarge the secure configuration space. Suppose IV_1 is the initial secure state space found by the basic INCISE, **Inclusive Expansion** (§5.2) sets IV_1 as the initiation constraint and expands the secure state space to IV_2; **Parallel Expansion** (§5.3) sets S_6 as the initiation constraint, and yields a larger secure state space $IV_2 \cup IV_3$.

inductive generalization will largely increase the execution time. After a few iterations, the algorithm is prohibited from terminating within a reasonable amount of time. Therefore, this technique alone does not guarantee to find the maximal IV.

5.3 Parallel Expansion

An IV from a single IC3 execution has limited flexibility, as later produced clauses are dependent on former ones in each frame. Meanwhile, the maximal IV could be irregular, containing multiple separate components. Thus, we have observed that a single IV produced by IC3 is hard to cover the underlying maximal IV.

Parallel expansion attempts to cover the underlying maximal IV with multiple invariants. To be specific, it picks a state which is secure and not yet covered by the current IV, finds a new IV containing the state, and joins the new IV with the current one. Such a new IV can be found by setting this uncovered state as the initiation constraint in a new IC3 execution. For example, suppose IV_2 in Fig. 2 is the current IV after executing the basic INCISE as well as the inclusive expansion technique. S_6 is a secure state which has not yet been covered by the current IV. INCISE could yield a new IV, IV_3, by setting S_6 as the initiation constraint. Thus, $IV_2 \cup IV_3$ is the expanded IV after parallel expansion.

This technique has the same drawback as the inclusive expansion technique: the time required for each execution increases after several successful attempts. Because it is unknown whether an uncovered state $s \in \neg IV \wedge P$ is indeed secure, one needs to draw a sample from that space, and then query if $\neg P$ is reachable from s. As the current IV converging to the maximal IV, more trials are required to find a valid sample in estimate. In practice, we run inclusive expansion and parallel expansion alternatively, depending on their efficiencies during execution.

5.4 Invariant Weakening

Alongside the techniques mentioned above, we can weaken the Boolean formula of IV to enlarge it. To implement this idea, this technique eliminates clauses in IV, while the new IV after elimination is still an invariant.

It considers the mutually exclusive formula of the invariant condition: $IV \wedge Tr \Rightarrow \neg IV'$. The invariant condition should hold when this formula is unsatisfiable. Notice that the negated formula can be efficiently created by introducing auxiliary Tseitin Variables [40]. Then a SAT solver is utilized to extract an unsatisfiable core [26], a subset of clauses that together makes the formula unsatisfiable. A clause in IV can be eliminated if neither itself nor its corresponding clauses in $\neg IV'$ are in the unsatisfiable core.

This technique takes only 1 SAT query, as opposed to a large number of SAT queries for an IC3 execution. So it can be initiated throughout the enlargement process.

6 Secure Configuration Space in Practice

So far, we have derived algorithms to compute IV, which is a succinct representation of the results from the time-consuming model checking process. In this section, we show how to efficiently query it to obtain a secure configuration or to determine whether a configuration is secure.

6.1 Configuration State Variables

One way to encode configurations into a formal specification is through adding configuration state variables: $\bar{x} = \bar{x}_{conf} + \bar{x}_s$. The values of the configuration state variables $\bar{x}_{conf}$ keep unchanged after the initialization, while the values of the normal state variables $\bar{x}_s$ evolve in every state transition. INCISE does not distinguish the two sets of variables when searching for the secure configuration space. Another way is to encode configurations without explicit configuration state variables, such that $\bar{x} = \bar{x}_s$. The model has different behaviors depending on the initial values of the state variables, which implicitly encode the system configurations.

6.2 Querying the Security of a Configuration

A configuration is secure if all corresponded initial states reside in IV. The task is trivial if the configuration s_{conf} is a full assignment to $\bar{x}$ and only one corresponded initial state exist: the security can be decided by a single SAT query $SAT(IV \wedge s_{conf})$. If s_{conf} is a partial assignment, a system engineer may want to ensure all possible initial states overlapping with s_{conf} are secure. In that regard, we check whether the secure state space intersects with the complement of the partial assignment: $SAT(\neg IV \wedge s_{conf})$. If it does, s_{conf} is not necessarily secure, and users need to refine it. If the result is unsatisfiable, we can safely claim s_{conf} is secure. The Boolean formula for $\neg IV$ can be obtained by introducing auxiliary Tseitin variables [40] and executing CNF simplification [27, 31].

Secure configuration spaces make other interesting applications possible, for instance, the "minimum revision". Network verification tools have a common feature: suggesting a correct alternative configuration when the original one is insecure. Leveraging the secure configuration space, users can find such a configuration that is as close as possible to the original configuration. In specific, Hamming distances can be efficiently encoded as cardinality constraints in CNF [7]. To find a correct alternative in IV that has a distance no more than n from the original configuration, one can query $SAT(IV \wedge card(s_{conf}, n))$,

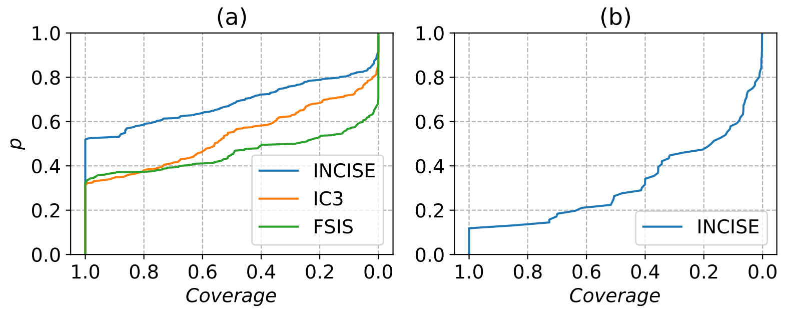

Fig. 3. Comparisons of coverage in Cumulative Distribution Function (CDF) for 466 HWMCC benchmarks with safety properties. (a) **Safe benchmarks**: our algorithm (INCISE) achieves better coverage than traditional algorithms on safe benchmarks. INCISE can reach 100% coverage on more than 50% of these benchmarks, which is 55.7% better than the other algorithms. (b) **Unsafe benchmarks**: our algorithm still works on unsafe benchmarks, while the other algorithms cannot find a valid invariant.

where $card(s_{conf}, n)$ denotes the cardinality constraint that the Hamming distance is at most n. Searching for n in an ascending order gives the configuration with the minimal revision. Applying such a configuration can fix the violations, while it will only incur a small impact on the deployed system.

7 Evaluation

7.1 Implementation

Our system consists of three parts. Part 1 is based on the reference IC3 implementation [12]. We added functionalities that support initial-blind model checking and control the extent of generalization. We also maintain a symbol mapping from the variables in IV to the state variables. Part 2 uses the k-liveness implementation in the IIMC [13] for constraint extraction and computing the k value. Part 3 integrates the above two parts and the enlargement techniques. For liveness properties, it constructs safety properties as well as revised models by leveraging Part 2, and pass them to Part 1. Additionally, it can query a configuration against a secure configuration space and computes the minimum revision if the configuration is insecure. We made these implementations and modifications in $\sim$1050 lines of code.

Part 1 and Part 2 are based on the reference implementations of IC3 and k-liveness, respectively. Nevertheless, INCISE is not depending on any particular versions of these algorithms, and can benefit from their more efficient variants.

7.2 Performance Analysis

Metrics and Testbed We evaluate the performance of INCISE from two aspects: *i)* execution time and *ii)* coverage of secure states. Recall that any states in IV must be

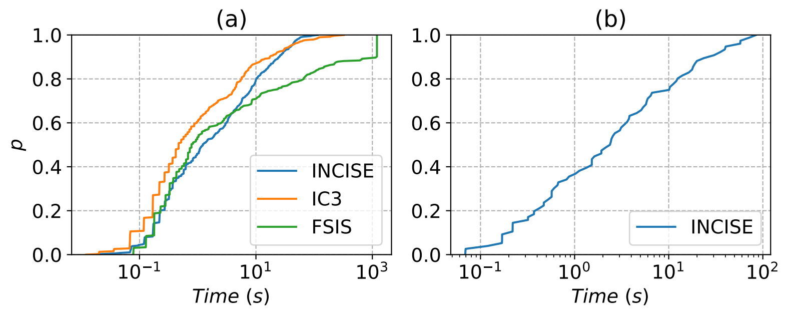

Fig. 4. Comparisons of execution time in CDF for 413 HWMCC benchmarks with safety properties. (a) **Safe benchmarks**: our algorithm (INCISE) runs slightly slower than the original IC3, but can finish on all benchmarks that the original IC3 can finish. (b) **Unsafe benchmarks**: Our algorithm still works on unsafe benchmarks, while traditional algorithms cannot find a valid invariant.

indeed secure, but an IV found by INCISE may not be the maximal secure state space that covers all secure states. Coverage is defined as the ratio of the secure space found by INCISE to the underlying maximal secure state space.

Computing this ratio analytically is difficult in practice. So we estimate it by sampling sufficiently many random configurations and count if each is secure and covered by IV. For binomial data, drawing 1,000 samples can achieve a 95% confidence interval of 0.04 in the worst case.

All experiments are executed on a machine with 128GB of RAM and a 2.9GHz processor. Only one thread was allowed at a time throughout our evaluation.

Evaluation for Safety Benchmarks We collected all single safety benchmarks from HWMCC [11] benchmark suites from 2008 to 2020. The benchmarks are classified into two categories: safe ones, which do not violate the corresponding properties given their original initial states, and unsafe ones, which violate the properties from their original initial states. To estimate the coverage ratio, we randomly draw 1,000 independent states for every benchmark, and compute their ground truth (safe/unsafe when set as the initial state) using ABC [17]. Duplicate benchmarks as well as those taking too long to generate the ground truth (over 1,200 seconds on ABC) were dropped. Afterward, 466 benchmarks were remaining.[3]

Safe configuration space is a new concept. As far as we are aware, no other algorithms aim to compute it. However, there exists efficient SAT-based symbolic model checking algorithms such as IC3 and FSIS [15], and they can also find an inductive invariant that is a sound subset of all secure states. In this regard, we compare INCISE against original IC3 and FSIS to show its performance for safety benchmarks.

We attempt inclusive extension 2 times and parallel extension 1 time after the basic INCISE execution for every benchmark.

[3] We have released raw experimental results in GitHub: https://github.com/FormalCellular/SecureCofigurationSpace

Table 1. Secure configuration space search results of INCISE on 20 HWMCC benchmarks with liveness properties.

Bench	#Latch	k_{max}	Time(s)	AC Ratio	Bench	#Latch	k_{max}	Time(s)	AC Ratio
Safe Benchmarks									
cuabq2mf	35	11	1.05	96.3%	lmcs06bc57sp3	129	27	32.13	99.9%
cuffl10	151	26	3.74	93.6%	lmcs06brp0	89	2	1.26	100%
culock	11	102	95.12	97.8%	lmcs06brp2	89	9	3.01	100%
cusarb16	50	27	1.53	85.4%	lmcs06mutex0	13	5	0.58	100%
cutf3	35	7	2.24	90.1%	lmcs06prodcell2	172	99	649.75	100%
lmcs06abp4p1	54	26	17.69	99.8%	lmcs06prodcell4	172	88	656.75	100%
lmcs06bc57sp1	129	56	215.20	100%	lmcs06short0	10	2	0.49	99.5%
lmcs06bc57sp2	129	73	179.66	100%	lmcs06srg5p0	46	8	0.97	100%
Unsafe Benchmarks									
cujc128f	129	1	3.01	100%	lmcs06short1	10	1	0.85	100%
lmcs06mutex1	13	2	1.17	96.1%	lmcs06srg5p2	46	1	1.35	99.2%

Fig. 3 (a) compares the coverage of our algorithm against IC3 and FSIS in Cumulative Distribution Function (CDF). The x-axis represents the coverage ratio, while the y-axis represents the portion of benchmarks that have their coverage greater than that x value. For instance, if a line goes across (0.8, 0.6), it means for 60% of all benchmarks, the coverage is better than 80%. It is shown that our algorithm achieves significantly better coverage. Specifically, on 202 out of the 389 benchmarks, our algorithm achieves the best possible coverage, *i.e.*, it finds out all secure states among the 1,000 samples. This result is a 55.7% improvement compared with the other algorithms.

Fig. 3 (b) plots the CDF of INCISE on unsafe benchmarks. Original IC3 and FSIS cover no states, as they cannot deal with the situation that the initial state is not secure. It can be seen that the coverage of INCISE on the unsafe benchmarks is worse than that on the safe benchmarks. We suppose it is because unsafe benchmarks are more difficult for model checkers in nature.

Fig. 4 (a) and (b) show the execution time of the algorithms in CDF. Similarly, if a line intersects with a point (10^1, 0.8), for 80% of the benchmarks, their execution time is less than 10 seconds. It turns out INCISE requires more execution time than the original IC3, mainly due to the enlargement. This overhead is reasonable since a conventional model checking algorithm can only validate a single configuration at a time. For all benchmarks that IC3 can finish in 1,200 seconds, INCISE can also finish.

Evaluation for Liveness Benchmarks INCISE reduces a liveness checking problem to a safety checking problem. The key challenge here is how to find a suitable *k*, which is neither too small (so that it reduces the coverage) nor too large (which increases the size of the model checking problem). Remember that every secure state, when set to the initial state, may have a different *k* value, representing the number of absorbing circuitries it requires. Given a value of *k*, the AC (accuracy) ratio depicts the proportion of secure states that require less or equal to *k* such circuitries. For each of the benchmarks, we initiate k-liveness 19 times on 19 randomly selected secure initial states and pick up the

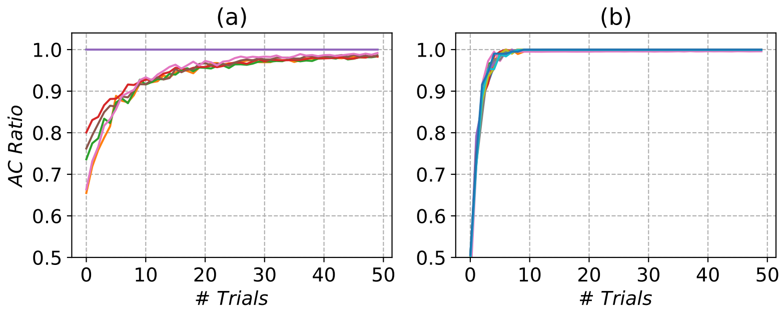

Fig. 5. The AC Ratio gets improved when the number of random trials increases. After 19 trials, the estimate of the AC Ratio is greater than 0.95. The trends of the two sub-figures are different because they are collected from two distinct families of benchmarks: (a) CU series and (b) LMCS06 series.

largest k among all trials. Theoretically, the largest k is greater than or equal to 95% of all ks in estimation [34].

We collected all liveness benchmarks from HWMCC benchmark suites from 2008 to 2020. Duplicate benchmarks and those taking too long to generate the ground truth (over 1,200 seconds on default k-liveness [13]) are dropped. Table 1 lists the total execution time and the AC Ratio for those benchmarks. The benchmarks can still be successfully solved in the timing budget when the AC Ratio reaches an ideal level.

Furthermore, we demonstrate how the AC Ratio gets improved when the number of random k-liveness trials grows. The x-axis in Fig. 5 (a) and (b) represents the numbers of trials, while the y-axis represents the average AC Ratio of 100 random tests. Each line is corresponding to a benchmark. As it is supposed to be, the AC Ratio gets improved when the number of trials grows. After 19 trials, the estimate is greater than 0.95. The two sub-figures' trends are different because they were collected from two distinct families of benchmarks: (a) CU series and (b) LMCS06 series.

8 Case Study: Cellular Emergency Protocols

We have successfully applied INCISE to extract the secure configuration space of the cellular emergency call system. Our approach resolved real-world availability and security issues in the systems of three world-leading cellular network carriers. We use this case study to show how the proposed algorithm can be applied to network protocols.

An emergency call session involves two parties: the user equipment (UE) and the cellular network provider (Carrier). Though all UEs and Carriers follow the same protocols for emergency call services, their designers' understandings and implementations of the protocols could vary. For example, UEs are made by various manufacturers and are using different operating systems; Carriers have distinct business considerations and device providers. Hence, given the flexibility of configurations in protocols, different parties have their preferences to determine each option.

Following the emergency call-related protocols [2–5], we built a formal model of emergency call system by extracting the related states, transitions, conditions, and ac-

Table 2. Secure configuration space search results for the cellular emergency call system.

Total #Secure	Algorithm	Time(s)	Found #Secure	Coverage
518	INCISE (Basic)	0.036	367	70.8%±3.9%
	w/Inclusive Expansion	0.069	518	100%±0%
	w/Parallel Expansion	0.068	518	100%±0%

tions [6, 35]. The skeleton of the model is based on the state machine of the cellular network call setup process described in [4]. All configurations in the emergency number identification process were modeled as configuration state variables. The formal specification and the corresponding CTL properties went through the same secure configuration space search processes as those in Section 7.2. The specification of the emergency call system as well as the corresponding verification codes will be made publicly available.

In the first stage, we assigned the cellular network configurations of three major carriers to the formal model and checked them with a model checker. The system violates the properties with any of those configurations. We verified the counterexamples we yielded with off-the-shelf UEs on those three carriers. One counterexample we found and reproduced has surprised our collaborating carrier: a roaming user, whose home country uses a different emergency number, cannot dial the local emergency number on the locked screen emergency dial panel. This study suggests a new problem: how could a system engineer know if a configuration is secure in real-time and leverage all the benefits of a model checker? How can the engineer fix a newly discovered issue on a fast-evolving system by finding a secure configuration?

We believe the secure configuration space is the answer. We applied our proposed algorithm, INCISE, to generate the secure configuration space for the formal specification of the cellular emergency call system. We first applied the basic INCISE to the model, and then used the inclusive expansion and parallel expansion to enlarge the generated secure configuration space. We used the same sampling strategy to measure their coverages.

As shown in Table 2, the three versions can all finish very quickly, in just several milliseconds. The basic INCISE can reach a 70.8% (±3.9%) coverage at a 95% confidence level. Given such a coverage, a random secure configuration proposed by an engineer has a large probability to bypass the model checking process and get verified in real-time. Both enlargement techniques can trade-off execution time for better coverage. After performing any of them, the coverage can reach 100% over all the secure samples. In this case, the real-time judgment is both sound and complete. Following the methods in Section 6, we extracted the "minimum revision" from the three major carriers' deployed configurations. The revised configurations can solve all the counterexamples we found.

9 Related Work

9.1 Symbolic Model Checking Algorithms

Graph-based Checking: Binary Decision Diagram (BDD) is a symbolic model which provides a canonical representation for Boolean formulas [18]. Recent SAT-based model checking algorithms have significantly outperformed BDD-based algorithms in terms of execution time and memory consumption [21]. When applying BDD to our problem, determining whether each state satisfies a correctness property may require a traversal on the diagram. Instead, INCISE can determine the security of a state in 1 SAT query once the invariant is found.

Safety Checking: SAT-based symbolic model checking with backward search [32] iteratively blocks cubes that can reach an error. At termination, it can find a weak invariant, which is desirable for the secure state search. However, it lacks advanced concepts to improve efficiencies like relative induction and clause propagation, as seen in IC3 [14]. Moreover, the algorithm only guarantees soundness when it terminates, meaning it is not possible to use an intermediate result as a partial secure state space. k-induction [38] and interpolation [33] are improvements to the bounded model checking [10]. They can find invariants as IC3 does, but their efficiency is believed to be behind the state-of-the-art IC3 algorithm [24].

Liveness Checking: Similar to k-liveness, Liveness-to-safety [37] translates a liveness problem to a single safety problem. However, k-liveness incrementally generates safety problems based on the k value, which allows the trade-off between completeness and execution time. FAIR [16] incrementally constructs fair cycles in the state transitions using symbolic safety checkers. Although FAIR is capable to find a secure state space for liveness properties, it does not reduce liveness problems to safety ones. Thus, it cannot benefit from the techniques for safety secure state search.

9.2 Formal Methods for Cellular Network Security

Formal methods provide an automatic and exhaustive way to verify the cellular network protocols and systems. Tu et al. [41] analyzed the control plane interaction between 3G and 4G systems by formally modeling their interaction and applying model checkers. Hussain et al. [30] specified 4G NAS layer protocols by NuSMV. Furthermore, they instrumented the specification with a Dolev-Yao style adversary model to study the security of 4G protocols. Hou et al. [29] formally specified the cellular emergency call system by TLA+ and discovered several availability and security issues on it. Symbolic theorem provers also have been used to formally verify the Authentication and Key Agreement (AKA) protocols of 3G [1] and 5G [8, 23]. To summarize, existing work mainly focuses on how to specify and verify protocols and systems. In comparison, INCISE aims to find a secure configuration space from a formal model. From that space, INCISE can efficiently validate a secure configuration or correct an insecure one. INCISE is also complementary to existing work and can directly utilize their formal models.

10 Conclusion

In this paper, we propose a novel concept called secure configuration space. Facilitated by the secure configuration space, system engineers can query whether a configuration satisfies either safety or liveness properties in real-time; security researchers can figure out the way to tame a problematic system.

We present an algorithm, INCISE, to compute the secure configuration space. It leverages state-of-the-art symbolic model checking algorithms. INCISE computes a decent sound result in a short period of time. Moreover, the result can be further improved as much as the timing budget allows until convergence. Experiment results on industrial benchmarks demonstrate the efficiency and flexibility of INCISE. We also showcase the capability of our proposed algorithm with a security study on cellular network protocols.

11 Acknowledgements

This work was partially supported by the National Science Foundation (NSF) under grants 2113704, 2148177, and 2229454. Any opinions, recommendations, or findings are those of the authors and do not reflect the views of Alibaba Cloud.

References

1. 3GPP: Formal Analysis of the 3G Authentication Protocol. Technical Report (TR) 33.902, 3rd Generation Partnership Project (3GPP) (2001), version 4.0.0
2. 3GPP: IP Multimedia Subsystem (IMS) emergency sessions. Technical Specification (TS) 23.167, 3rd Generation Partnership Project (3GPP) (2020), version 16.2.0
3. 3GPP: Non-Access-Stratum (NAS) protocol for 5G System (5GS); Stage 3. Technical Specification (TS) 24.501, 3rd Generation Partnership Project (3GPP) (2020), version 16.5.1
4. 3GPP: Non-Access-Stratum (NAS) protocol for Evolved Packet System (EPS); Stage 3. Technical Specification (TS) 24.301, 3rd Generation Partnership Project (3GPP) (2020), version 16.5.1
5. 3GPP: Service aspects; Service principles. Technical Specification (TS) 22.101, 3rd Generation Partnership Project (3GPP) (2020), version 17.2.0
6. Al Ishtiaq, A., Das, S.S.S., Rashid, S.M.M., Ranjbar, A., Tu, K., Wu, T., Song, Z., Wang, W., Akon, M., Zhang, R., et al.: Hermes: Unlocking security analysis of cellular network protocols by synthesizing finite state machines from natural language specifications. 33rd USENIX Security Symposium (USENIX Security 24) (2024)
7. Bailleux, O., Marquis, P.: Some computational aspects of distance-sat. Journal of Automated Reasoning **37**(4), 231–260 (2006)
8. Basin, D., Dreier, J., Hirschi, L., Radomirovic, S., Sasse, R., Stettler, V.: A formal analysis of 5G Authentication. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 1383–1396. ACM (2018)
9. Biere, A., Artho, C., Schuppan, V.: Liveness checking as safety checking. Electronic Notes in Theoretical Computer Science **66**(2), 160–177 (2002)
10. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic model checking without bdds. In: International conference on tools and algorithms for the construction and analysis of systems. pp. 193–207. Springer (1999)
11. Biere, A., Claessen, K.: Hardware model checking competition. In: Hardware Verification Workshop (2010)
12. Bradley, A.: Ic3-ref. https://github.com/arbrad/IC3ref (2015)
13. Bradley, A., Cox, A., Dooley, M., Hassan, Z., Somenzi, F., Zhang, Y.: Iimc. https://github.com/mgudemann/iimc (2018)
14. Bradley, A.R.: Sat-based model checking without unrolling. In: International Workshop on Verification, Model Checking, and Abstract Interpretation. pp. 70–87. Springer (2011)

15. Bradley, A.R., Manna, Z.: Checking safety by inductive generalization of counterexamples to induction. In: Formal Methods in Computer Aided Design (FMCAD'07). pp. 173–180. IEEE (2007)
16. Bradley, A.R., Somenzi, F., Hassan, Z., Zhang, Y.: An incremental approach to model checking progress properties. In: 2011 Formal Methods in Computer-Aided Design (FMCAD). pp. 144–153. IEEE (2011)
17. Brayton, R., Mishchenko, A.: Abc: An academic industrial-strength verification tool. In: International Conference on Computer Aided Verification. pp. 24–40. Springer (2010)
18. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. Computers, IEEE Transactions on **100**(8), 677–691 (1986)
19. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: Ic3 modulo theories via implicit predicate abstraction. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 46–61. Springer (2014)
20. Claessen, K., Sörensson, N.: A liveness checking algorithm that counts. In: 2012 Formal Methods in Computer-Aided Design (FMCAD). pp. 52–59. IEEE (2012)
21. Clarke, E.M., Henzinger, T.A., Veith, H.: Introduction to model checking. In: Handbook of Model Checking, pp. 1–26. Springer (2018)
22. Cousot, P., Cousot, R., Fähndrich, M., Logozzo, F.: Automatic inference of necessary preconditions. In: International Workshop on Verification, Model Checking, and Abstract Interpretation. pp. 128–148. Springer (2013)
23. Cremers, C., Dehnel-Wild, M.: Component-Based Formal Analysis of 5G-AKA: Channel Assumptions and Session Confusion. In: Symposium on Network and Distributed Systems Security (NDSS) (2019)
24. Een, N., Mishchenko, A., Brayton, R.: Efficient implementation of property directed reachability. In: 2011 Formal Methods in Computer-Aided Design (FMCAD). pp. 125–134. IEEE (2011)
25. Goel, A., Sakallah, K.: Avr: Abstractly verifying reachability. Tools and Algorithms for the Construction and Analysis of Systems **12078**, 413 (2020)
26. Guthmann, O., Strichman, O., Trostanetski, A.: Minimal unsatisfiable core extraction for smt. In: 2016 Formal Methods in Computer-Aided Design (FMCAD). pp. 57–64. IEEE (2016)
27. Heule, M.J., Järvisalo, M., Biere, A.: Efficient cnf simplification based on binary implication graphs. In: International Conference on Theory and Applications of Satisfiability Testing. pp. 201–215. Springer (2011)
28. Hou, K., Li, Y., Yu, Y., Chen, Y., Zhou, H.: Discovering emergency call pitfalls for cellular networks with formal methods. In: Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services. pp. 296–309 (2021)
29. Hou, K., Li, Y., Yu, Y., Chen, Y., Zhou, H.: Discovering emergency call pitfalls for cellular networks with formal methods. In: Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys). pp. 296–309 (2021)
30. Hussain, S.R., Chowdhury, O., Mehnaz, S., Bertino, E.: LTEInspector: A Systematic Approach for Adversarial Testing of 4G LTE. In: Symposium on Network and Distributed Systems Security (NDSS). pp. 18–21 (2018)
31. Manthey, N.: Coprocessor 2.0–a flexible cnf simplifier. In: International Conference on Theory and Applications of Satisfiability Testing. pp. 436–441. Springer (2012)
32. McMillan, K.L.: Applying sat methods in unbounded symbolic model checking. In: International Conference on Computer Aided Verification. pp. 250–264. Springer (2002)
33. McMillan, K.L.: Interpolation and sat-based model checking. In: International Conference on Computer Aided Verification. pp. 1–13. Springer (2003)
34. Mitzenmacher, M., Upfal, E.: Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis. Cambridge university press (2017)

35. Pacheco, M.L., von Hippel, M., Weintraub, B., Goldwasser, D., Nita-Rotaru, C.: Automated attack synthesis by extracting finite state machines from protocol specification documents. In: 2022 IEEE Symposium on Security and Privacy (SP). pp. 51–68. IEEE (2022)
36. Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Scalable analysis of linear systems using mathematical programming. In: International Workshop on Verification, Model Checking, and Abstract Interpretation. pp. 25–41. Springer (2005)
37. Schuppan, V., Biere, A.: Efficient reduction of finite state model checking to reachability analysis. International Journal on Software Tools for Technology Transfer **5**(2), 185–204 (2004)
38. Sheeran, M., Singh, S., Stålmarck, G.: Checking safety properties using induction and a sat-solver. In: International conference on formal methods in computer-aided design. pp. 127–144. Springer (2000)
39. Sheng, S., Hsiao, M.: Efficient preimage computation using a novel success-driven atpg. In: 2003 Design, Automation and Test in Europe Conference and Exhibition. pp. 822–827. IEEE (2003)
40. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: Automation of reasoning, pp. 466–483. Springer (1983)
41. Tu, G.H., Li, Y., Peng, C., Li, C.Y., Wang, H., Lu, S.: Control-plane protocol interactions in cellular networks. In: ACM SIGCOMM. pp. 223–234. ACM (2014)
42. Vizel, Y., Gurfinkel, A.: Interpolating property directed reachability. In: International Conference on Computer Aided Verification. pp. 260–276. Springer (2014)

Table of Notation

A Symbol Table

Notation	Description
$\overline{x}$	state variables
$\overline{x}'$	next state variables
s	a state; an assignment to $\overline{x}$
s'	the next state version of s
literal	a variable or its negation
cube	a conjunction of literals
clause, c	a disjunction of literals
E	a formula
E-state	a state s satisfies E: $s \models E$
E-set	the set of all E-states
$\mathbb{M}$	a finite state transition system $\mathbb{M} : (\bar{i}, \overline{x}, I, Tr)$
$\bar{i}$	primary input variables
I	initial condition
Tr	transition relation
trace	a sequence of states, s.t. $s_0 \models I$ and any $s_i, s'_{i+1} \models Tr$.
P	a property
F_i	a frame in IC3 algorithm; a conjunction of clauses
F_i-clauses	all clauses in F_i
IV	inductive invariant
s^* / c^*	a state violating the consecution condition / $c^* = \neg s^*$
c_{min}	a minimal inductive subclause of a clause c
q	the liveness event; a formula which should be eventually always satisfied
$\overline{x}_{conf}$ / $\overline{x}_s$	the state variables that encode configurations $\overline{x} = \overline{x}_{conf} + \overline{x}_s$
s_{conf}	an assignment to $\overline{x}_{conf}$