Hermes: Unlocking Security Analysis of Cellular Network Protocols by Synthesizing Finite State Machines from Natural Language Specifications

Abdullah Al Ishtiaq, Sarkar Snigdha Sarathi Das, Syed Md Mukit Rashid, Ali Ranjbar
Kai Tu, Tianwei Wu, Zhezheng Song, Weixuan Wang, Mujtahid Akon
Rui Zhang, Syed Rafiul Hussain
Pennsylvania State University

Abstract

In this paper, we present Hermes, an end-to-end framework to automatically generate formal representations from natural language cellular specifications. We first develop a neural constituency parser, NEUTREX, to process transition-relevant texts and extract transition components (i.e., states, conditions, and actions). We also design a domain-specific language to translate these transition components to logical formulas by leveraging dependency parse trees. Finally, we compile these logical formulas to generate transitions and create the formal model as finite state machines. To demonstrate the effectiveness of Hermes, we evaluate it on 4G NAS, 5G NAS, and 5G RRC specifications and obtain an overall accuracy of 81-87%, which is a substantial improvement over the state-of-the-art. Our security analysis of the extracted models uncovers 3 new vulnerabilities and identifies 19 previous attacks in 4G and 5G specifications, and 7 deviations in commercial 4G basebands.

1 Introduction

The cellular standard body, i.e., the 3rd Generation Partnership Project (3GPP), designs, creates and maintains technical specifications for cellular network systems used by network operators and equipment vendors worldwide [1]. However, a wide range of cellular stakeholders and network entities, a vast diversity of use cases such as SMS, data access, and roaming connections, and tighter backward compatibility requirements make it difficult for 3GPP to maintain the specifications in a simple form. Especially, the natural language description of the cellular network design spreading over hundreds of documents makes it extremely laborious and often error-prone to create formal models necessary for security analysis. Moreover, the inherent ambiguities and complexities of natural language often lead to misinterpretations by the developers, resulting in deviations and exploitable flaws in implementations [43, 50, 51, 69]. Such deviations are difficult to identify without access to the baseline/gold standard. However, the cellular standard body does not provide any formal model of the system, thereby leaving the lack of a gold standard.

Although formal analysis of cellular network design has uncovered several new vulnerabilities, all of them rely on hand-crafted models [14, 29, 40, 42, 64, 71]. Unfortunately, manually constructing such formal models is tedious and error-prone due to the sheer size of cellular systems and the large number of documents. Moreover, these models require significant time and effort and are often limited by modeling flaws (incorrectly representing the system) [40], inconsistent levels of abstraction [14, 42], and oversimplified representations of complex protocol details [40, 42], leading to inadequate formal security analysis. Furthermore, although 3GPP introduces new generations (e.g., 3G, 4G, 5G) of cellular technology roughly every decade, within a generation, specifications are frequently updated, e.g., on average, the specification documents are updated 5-6 times each year, and each update includes several hundreds of line changes [44]. Despite not being major technology shifts, these changes in the protocol can often lead to new vulnerabilities. For example, changes in the collision resolution mechanism between two particular procedures introduced in the 4G specification [10] led to a new vulnerability [26]. Likewise, in this work, we find a new vulnerability in 5G (§8.4.1), stemming from a new cause of message rejections introduced in the 5G specification update [4]. Conversely, due to time and effort constraints, frequently updating hand-crafted models is not always feasible. In fact, these prior models have never undergone revisions to account for new changes as they have emerged. Thus, automated extraction of formal models from cellular specifications is pivotal for analyzing the security of cellular protocols.

Recently, researchers used Natural Language Processing (NLP) tools to identify *hazard indicators* in cellular specifications for creating concrete test cases [26] and to discover *security-relevant change requests* [24]. Unfortunately, none of them extract formal models automatically. Furthermore, among the attempts to extract structured representation from natural language in other domains, e.g., protocols [67], geographical questions [34, 47, 90], medicine [19], and database queries [34, 47, 88, 89], RFCNLP [67] extracts Finite State Machines (FSMs) from Request for Comments (RFC) docu-

ments, albeit for small protocols. In contrast, cellular protocols have a substantially large number of transitions, a myriad of use cases, variables, events, message fields, and complex interactions across layers and sub-layers, rendering RFCNLP ineffective. The absence of standardized datasets or benchmarks further compounds the difficulty in automatically extracting formal representations.

In this paper, we address these challenges by developing Hermes (Hermes was the God of translator/interpreter in Greek mythology) that automatically extracts the formal representation of cellular network protocols from the given natural language specification documents. We develop three main components: a neural constituency parser called `NEUTREX`, `IRSynthesizer`, and `FSMSynthesizer`. With these components, Hermes generates FSMs as a set of transitions containing states, conditions (e.g., received messages), and actions (e.g., sending messages) in a logical format. At a high level, Hermes first uses `NEUTREX`, developed with a domain-knowledge-informed grammar and deep-learning-based neural parsing model, to extract the portions of natural language texts related to transitions and detect components (e.g., states, conditions, actions) within them, along with their logical relations. Hermes then leverages `IRSynthesizer`, designed with a Domain Specific Language (DSL) and a neural dependency parser, to identify the key information items, including variables and events, and convert the conditions and actions to logical formulas or *intermediate representations (IR)*, which can be easily transpiled to specific formal language amenable for formal analysis. Finally, the `FSMSynthesizer` of Hermes combines the logical formulas corresponding to the transition components to construct complete transitions and the FSMs.

To demonstrate the efficacy of Hermes across multiple cellular layers, generations, and releases, we evaluate it on 4G NAS [11], 5G NAS [5], 5G RRC [6], and RFC documents [67]. Our results show that Hermes-generated FSMs achieve up to 87.21% accurate transitions compared to human-annotated transitions for 4G and 5G NAS specifications. Regarding identifying transition constituents from cellular specification, `NEUTREX` achieves 68.69% labeled F1-score, showcasing a substantial improvement over the state-of-the-art RFCNLP's 38.52%. We also evaluate Hermes on RFCs, and it achieves 57.06% labeled F1-score compared to 47.76% of RFCNLP.

To demonstrate that Hermes facilitates security analysis of cellular networks, we use it to perform two different security analyses of the cellular protocols. First, we transpile the generated formal models to SMV language [62] by applying prior tools [40,42] and perform model checking. This analysis identifies **19** previous and **3** new vulnerabilities in the 4G and 5G specifications. Second, we compare the Hermes-generated FSM with commercial 4G cellular baseband FSMs to identify flaws in the implementations. This analysis uncovers **7** deviations in 9 basebands with an accuracy of 78%.

Responsible Disclosure. We have shared our findings with the GSMA CVD panel, and they have confirmed and acknowledged our contributions with CVD-2023-0071.

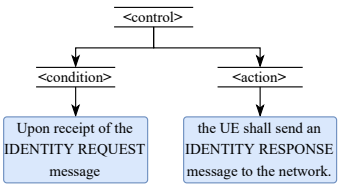

Figure 1: A constituency parse tree.

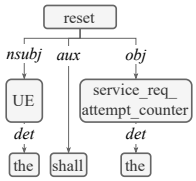

Figure 2: A dependency parse tree.

Contributions. We summarize our contributions as follows:

- We design and implement Hermes, a framework for automatically extracting formal models from natural language cellular specification documents. We show the effectiveness of Hermes across different cellular generations (4G and 5G) and layers (NAS and RRC), significantly outperforming state-of-the-art in both cellular and other domains.
- We demonstrate the use of the formal models extracted by Hermes in the security analysis of the *design* of 4G and 5G cellular network protocols. We identify 19 previous and 3 new issues in those design documents.
- We also show that the extracted formal representation of the 4G NAS specification effectively identifies 7 *implementation-level* flaws in 9 cellular devices by comparing those devices' FSMs with Hermes-generated FSM.
- We provide a curated dataset of natural language transition components in cellular specifications and the source code of Hermes, serving as a benchmark for future research. The dataset, source code, and properties are available at: https://github.com/SyNSec-den/hermes-spec-to-fsm.

2 Background

❑ **Cellular Network Architecture.** The User Equipment (UE), which includes USIM and cryptographic keys, is the device users use to access cellular services. A base station, gNB in 5G or eNodeB in 4G, provides cellular services to users and manages radio resources in a geographical cell area. Multiple base stations communicate with one another through a radio access network (RAN). The core network consists of several components or network functions (NF). The AMF (MME in 4G) is a major NF that is responsible for UEs' mobility management, e.g., registration, in the network through the Non-Access Stratum (NAS) protocol.

❑ **Word Embeddings and Pretrained Language Models.** Word embedding represents a word as a real-valued vector, which encodes its meaning. BERT [31] leverages Transformer [81] based encoders to learn contextualized word embeddings by pretraining over general-domain corpora. However, Liu et al. [60] reveal that the next sentence prediction objective of BERT degrades the embeddings, and the same words are masked in different training epochs of BERT, leading to sub-optimal training. They incorporated these decisions

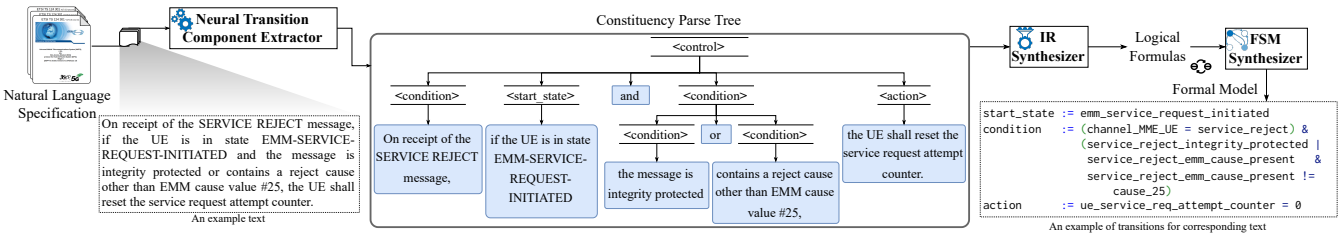

Figure 3: Overview of Hermes.

into RoBERTa to deliver stronger performances.

❑ **Constituency Parsing.** Constituency parsing builds a hierarchical tree, which describes the internal structure of a sentence, including *non-terminal* constituents and *terminal* input words. Figure 1 shows the input sentence parsed into its constituents– condition and action spans. Conventional constituency parsing methods [37, 79] rely on explicit grammar production rules demanding extensive domain knowledge without leveraging any constituent semantics. Neural constituency parsers [52, 53, 93] address this problem by learning those *grammar*s during training through marginal probabilities, which are utilized during inference.

❑ **Dependency Parsing.** While constituency parsing detects the constituent spans from a sentence, dependency parsing [23] provides the explicit relations that we need to synthesize FSMs from cellular specifications. For example, in Figure 2, if a span is detected as an *action* ("reset"), we also need to know the **subject** ("UE") and **object** ("service_req attempt_counter") associated with it. Accordingly, dependency parsing identifies the main **verb** word as the root node, and all other words have exactly one headword as their parent node. This gives a unique path from the root verb node to all other word nodes to facilitate the synthesis of logical expression.

3 Overview of Hermes

In this section, we define the problem, present an overview of Hermes, and discuss the challenges and insights.

3.1 Problem Statement

Given a cellular system specification in natural language, Hermes aims to generate a formal model, $\mathcal{M} = \langle \mathbb{P}, \mathbb{F}, \mathbb{H}, \mathbb{V} \rangle$:

- $\mathbb{P}$ is the set of protocol participants, $\{\mathcal{P}_1, \mathcal{P}_2, ...\}$, in the given specification.
- $\mathbb{F}$ is the set of synchronous communicating FSMs, $\{\mathcal{F}_1, \mathcal{F}_2, ...\}$, for each $\mathcal{P} \in \mathbb{P}$.
- $\mathbb{H}$ is the set of one-way communication channels between the entities in $\mathbb{P}$, i.e., $\{\mathcal{H}_{ij} | \forall \mathcal{P}_i, \mathcal{P}_j \in \mathbb{P} \wedge \mathcal{P}_i \neq \mathcal{P}_j\}$.
- $\mathbb{V}$ is the set of variables, $\{\mathcal{V}_1, \mathcal{V}_2, ...\}$, in $\mathcal{M}$.

We define each FSM in $\mathbb{F}$ as $\mathcal{F} = \langle \mathbb{S}, \mathcal{S}_0, \mathbb{T} \rangle$, where $\mathbb{S}$ is the set of states $\{\mathcal{S}_1, \mathcal{S}_2, ...\}$, $\mathcal{S}_0$ is the initial state, and $\mathbb{T}$ is the set of transitions. We further define each transition in $\mathbb{T}$ as $\mathcal{T} = \langle \mathcal{S}_i, \mathcal{S}_f, \mathcal{C}, \mathbb{A} \rangle$, where $\mathcal{S}_i$ is the starting state, and $\mathcal{S}_f$ is the ending state of the transition $\mathcal{T}$, $\mathcal{C}$ is the logical condition for the transition $\mathcal{T}$ to take place, and $\mathbb{A}$ is the set of actions, $\{\mathcal{A}_1, \mathcal{A}_2, ...\}$, which also occur when the transition $\mathcal{T}$ takes place. Here, an action $\mathcal{A} \in \mathbb{A}$ can be a message transmission through a channel $\mathcal{H} \in \mathbb{H}$ or an update to a variable $\mathcal{V} \in \mathbb{V}$.

3.2 Solution Sketch of Hermes

We address the problem in §3.1 with three primary components of Hermes: Neural Transition Component Extractor (`NEUTREX`), Intermediate Representation Synthesizer (`IRSynthesizer`), and Finite State Machine Synthesizer (`FSMSynthesizer`). The workflow is shown in Figure 3.

`NEUTREX` takes natural language texts and identifies the constituent spans (transition components in natural language, $\mathsf{TC}^{\mathsf{NL}}$s) corresponding to the states, conditions, actions, and the transition itself (`ctl_block`). For that, we extend a state-of-the-art neural constituency parser [93] and generate constituency parse trees for input natural language texts. Figure 3 demonstrates `NEUTREX` extracted $\mathsf{TC}^{\mathsf{NL}}$s (start_state, action, condition, and control) and logical relations (e.g., and, or) in the constituency parse tree corresponding to the input text.

In the next stage, `IRSynthesizer` takes $\mathsf{TC}^{\mathsf{NL}}$s (state, condition, or action) within the `ctl_block`, identified as constituent spans by `NEUTREX`, and constructs corresponding transition components in intermediate representations (IR), $\mathsf{TC}^{\mathsf{IR}}$, in a logical format. Finally, the generated IRs are compiled with `FSMSynthesizer` to create the transitions, ultimately constituting the formal representation of the cellular specification.

3.3 Challenges and Insights of Extracting FSM

C1: Complex associations of FSM transition components in natural language. Identifying transition components ($\mathsf{TC}^{\mathsf{NL}}$s), i.e., start state, end state, conditions, and actions of an FSM's transitions from natural language specifications is challenging. This is because the associations among the components are highly complex and depend on both syntactic features and semantic relations. For example, consider the sentence "*The UE initiates attach procedure* by sending an ATTACH REQUEST to the network" in TS 24.301 [11]. It can have two interpretations: (1) "the UE shall initiate the attach

```
par       ::= ctl_block par | text par | text
ctl_block ::= "<control>" ctl_unit "</control>"
ctl_unit  ::= ctl_block ctl_unit | start_state_tag ctl_unit |
              end_state_tag ctl_unit | act_tag ctl_unit |
              cond_tag ctl_unit | text ctl_unit | text
start_state_tag ::= "<start_state>" text "</start_state>"
end_state_tag   ::= "<end_state>" text "</end_state>"
cond_tag        ::= "<condition>" cond_unit "</condition>"
cond_unit       ::= cond_tag cond_unit | text cond_unit | text
act_tag         ::= "<action>" act_unit "</action>"
act_unit        ::= cond_tag act_unit | text act_unit | text
text            ::= token text | ε
```

Figure 4: Defined grammar for annotated data.

procedure", and "the UE shall send an ATTACH REQUEST to the network", resulting in two actions by a UE without any condition, or (2) The first part is treated as "when the UE initiates attach procedure" and thus can be considered as a condition for the second part–"sending the ATTACH REQUEST message". Therefore, domain knowledge is required to disambiguate such cases of complex associations.

❍ **Insight (I1)**: We adopt a domain-knowledge-informed grammar for transition components and a neural model to extract them automatically. First, we define a nested grammar for $\mathsf{TC}^{\mathsf{NL}}$s considering complex and nested associations prevalent in cellular specifications. We curate an expert-annotated dataset by labeling $\mathsf{TC}^{\mathsf{NL}}$s in cellular specifications accordingly. Next, we design a neural constituency parsing-based transition component extractor (NEUTREX) that extracts complex nested structures by comprehending semantic information. The resulting parse tree (as in the parse tree in Figure 3) precisely maintains the hierarchy of different constituent spans ($\mathsf{TC}^{\mathsf{NL}}$s), allowing us to automatically handle the nested grammar for cellular documents. In contrast, existing neural models for network protocols [67] fail to automatically detect and handle such complex associations and nested constituents.

C2: Limitations of existing NLP models on cellular specification. Off-the-shelf constituency parsers face several challenges when extracting cellular $\mathsf{TC}^{\mathsf{NL}}$. As cellular specifications are written in a highly technical language with 3GPP-specific abbreviations and terms, existing word embeddings trained on general-purpose corpora are inadequate owing to the domain shift. Also, existing constituency parsers do not perform reasonably well in the cellular dataset because of complex nested associations. Finally, transformer-based embedding models often face token limits surpassed by cellular document paragraphs. This requires appropriate measures to avoid truncated parse trees and inaccurate FSMs.

❍ **Insight (I2)**: To close the domain gap in the embedding model, we curate and clean a large natural language dataset consisting of 22,000 cellular documents, including technical specifications, change requests, and technical reports, for unsupervised pretraining of a RoBERTa [60] embedding model. Second, while training the constituency parser [93], we observe that it struggles to extract both span boundaries and labels simultaneously in the first few steps. Therefore, at the beginning of the training, we prioritize labeling the tokens and gradually give the same priority to identifying spans as the training progresses. Lastly, long datapoints are handled by splitting and generating embeddings for all input tokens.

C3: Absence of annotated cellular datasets for translation from NL phrases to logical form. Transition components ($\mathsf{TC}^{\mathsf{NL}}$s) are not directly usable for any formal security analysis and need to be converted to logical expressions and statements, such as $\mathsf{UE\_service\_req\_attempt\_counter} := 0$. Existing data-driven approaches for such translation [33,45,54,78] are largely domain-specific and fail to generalize across domains or capture complex logical relations [88]. Further, no dataset exists for translating natural language to logical formulas for cellular systems, and annotating such a large dataset is tedious and error-prone.

❍ **Insight (I3)**: To circumvent the lack of annotated datasets, we simplify the NL-to-logic task and take an intermediary step similar to schema-based translations [72,73]. Accordingly, we define a Domain Specific Language (DSL) (partially shown in Figure 7) that enables parsing and translating $\mathsf{TC}^{\mathsf{NL}}$s to precise and consistent logical representations. Moreover, the DSL postulates a structure for the translation and highlights the required information for each logical formula through commands and corresponding arguments. If any information is missing in $\mathsf{TC}^{\mathsf{NL}}$ due to natural language ambiguities or constituency parsing inaccuracies, the DSL can guide the search for them through metadata, e.g., type information.

C4: Text to DSL rule mapping requires explicit arguments. The DSL provides a pathway to logical formulas from natural language strings. However, it has a strict syntax and fixed arguments, whereas natural language is inherently diverse and imprecise. For example, different sentence structures may express the same meaning, such as "the MME shall send an ATTACH ACCEPT message to the UE", "an ATTACH ACCEPT shall be sent to the UE by the MME", and so on. Mapping such diverse texts to the same DSL rule is non-trivial.

❍ **Insight (I4)**: The DSL commands are inherently tree-structured and can be represented as Abstract Syntax Trees (AST). Conversely, natural languages follow sequential patterns. To map them to DSL, a tree format is a reasonable intermediate step. Among the different natural language tree representations, dependency trees extract useful relations among the tokens and often demonstrate similar structures as the DSL ASTs. Figures 8 (a) and 8 (c) show such similarities. Here, the primary verb–"receive", is the root in both trees, and the messages are their descendants. Even in cases where a similar meaning is represented by different sentence structures (e.g., examples in C4), these relations are preserved reasonably well. These similarities indicate that dependency parsing on the transition components ($\mathsf{TC}^{\mathsf{NL}}$) can help in mapping texts to DSL rules and, thereafter, generating logical formulas.

4 Extracting Transition Components

The neural transition component extractor (NEUTREX) takes each paragraph of a cellular specification as input and extracts the constituency parse trees based on the grammar in Figure 4. It identifies ctl_block with transition components (TC^{NL}s) and simultaneously refrains from including natural language texts that do not contain any transition information.

4.1 Grammar and Annotated Dataset

As discussed in §3.3, the TC^{NL}s are interrelated through complex associations and require domain expertise for correct interpretations. To address this challenge, we define a grammar for TC^{NL}s as shown in Figure 4 and annotate three specification documents, namely 4G-NAS-Release16 [11], 5G-NAS-Release17 [5], and 5G-RRC-Release17 [6]. The annotation is done by manually inspecting each paragraph and sentence, and identifying states, conditions, and actions in them. Further, we consider their semantic relations and dependencies, and follow the defined grammatical structure during annotation.

A string in the language defined by this grammar can consist of English tokens (token) or can have one or more control blocks (ctl_block). Here, the control block is the primary construct for transitions and contains TC^{NL}s (i.e., start_state_tag, end_state_tag, cond_tag, and act_tag). Figure 3 illustrates an example control block with multiple condition spans (cond_tag), start state span (start_state_tag), and action spans (act_tag). Moreover, our grammar supports nested spans and facilitates extracting complex logical relationships among them, as demonstrated in the example in Figure 3.

4.2 Constituency Parsing - NEUTREX

We build NEUTREX, a neural constituency parser that extracts ctl_block and corresponding TC^{NL}s from natural language texts. As cellular specifications contain rich and complex associations of transition components, which pose a significant challenge for existing NLP models trained on general domains, we propose the following innovations to adapt and enhance state-of-the-art constituency parser [93] for our task:

(i) Instead of the original RoBERTa model [60] that cannot recognize most cellular-specific technical words and phrases, we pretrain a new embedding model, CellulaRoBERTa, from scratch with cellular-specific documents, as described in §4.3. This ensures that the constituency parser can correctly comprehend the tokens used in cellular specification documents.

(ii) We reformed the loss function so that it properly integrates span semantics and span structure information during objective optimization. We demonstrate that this is critical in improving performance in detecting the constituents.

Workflow. NEUTREX works in a two-stage scheme, as shown in Figure 5. In the **Bracketing** stage, the parser takes an input sentence and outputs an unlabeled tree housing all the

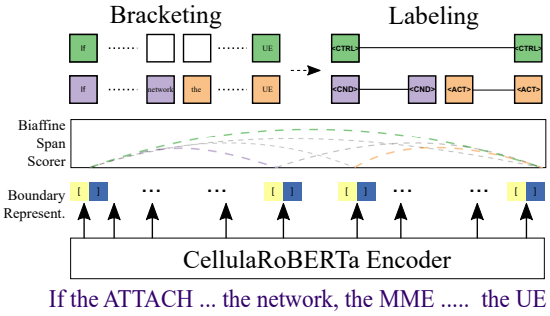

Figure 5: Detailed architecture of NEUTREX.

constituents. In the **Labeling** stage, these constituents are classified into different labels to get the output tree. In this way, we not only get the tags of different constituents for the input, but also preserve the structural association.

Bracketing. Given a sentence x, the conditional probability of an unlabeled tree y is calculated as: $P(y|x) = \frac{\exp^{S(x,y)}}{\sum_{y'} \exp^{S(x,y')}}$, where $S(x,y) = \sum_{(i,j)\in y} s(i,j)$ is the sum of the scores of all constituent spans (i,j) under the tree y, and y' enumerates all possible trees using the CKY algorithm [38]. For each word w_i, we use multi-layer perceptrons over domain-specific CellulaRoBERTa embeddings to produce left boundary representations u_l^i and right boundary representations u_r^i for this word. Then, for a given span (i,j), we calculate the biaffine score using the left boundary word representation w_i and right boundary word representation w_j to determine the score for the span: $s(i,j) = u_l^{i\top} W u_r^j + b$, where W and b are trainable parameters. After calculating the score of all possible trees, the best tree $\hat{y}$ is chosen as: $\hat{y} = \arg\max_y P(y|x)$.

Labeling. Given an input sentence x and a tree y, the label is predicted for each of the constituents (i,j) in the tree independently. We denote a label as $l \in \mathcal{L}$, where $\mathcal{L}$ is the label set in our grammar. For a constituent (i,j) and a label l, we calculate a biaffine score $s(i,j,l)$ similarly as done in the bracketing stage, and we have $|\mathcal{L}|$ biaffine heads for all the labels. We pick the best label as: $\hat{l} = \arg\max_{l\in\mathcal{L}} s(i,j,l)$.

Training. Given a sentence x with the corresponding ground truth constituency tree y and label l, the vanilla loss [93] is calculated as $L = L_{\text{bracket}} + L_{\text{label}}$, where L_{bracket} denotes the global conditional random field (CRF) loss in sentence level, and L_{label} is the cross entropy loss for labeling using teacher forcing by training only with ground truth trees. However, we found that L_{bracket}, which encodes the bracketing loss to encode the tree structure, does not work well in the early stages of training, and the model quickly converges to local minima where it only generates trivial span prediction with suboptimal performance (labeled F1 score of only 0.2849).

Consequently, we postulate that the constituent label information from L_{label} could enhance the integration of span semantics that is lacking in the early training phase, which can later facilitate the model to learn structured informa-

tion by training with L_{bracket}. Thus, we emphasize labeling training in the early stage by modifying the loss as $L = \lambda_1 L_{\text{bracket}} + \lambda_2 L_{\text{label}}$. In the first few stages, when bracket loss is unstable, we give high weights to L_{label} (i.e., $\lambda_2 > \lambda_1$). Later, we normally train with $\lambda_2 = \lambda_1$. This ensures smooth training, resulting in accurate constituency parsing.

4.3 Pretraining CellulaRoBERTa

The original RoBERTa [60] model used in prior constituency parser [93] fails to perform satisfactorily on cellular documents because of many unknown tokens and new domain knowledge, such as the frequent use of 3GPP-specific words, abbreviations, and technical terms. Therefore, we pretrain a RoBERTa model from scratch on cellular documents so that it can accurately generate embeddings for cellular specification tokens. Accordingly, we curate a large dataset of cellular documents consisting of more than 22,000 documents, including cellular specifications, change requests, technical reports, etc. We preprocess these documents by removing page headers and footers, non-ASCII characters, tables, and figures. We then pretrain a RoBERTa model on our cleaned corpus. In this manner, we create CellulaRoBERTa, a RoBERTa-based specialized language model providing domain-specific contextualized word representations of cellular specifications.

5 Synthesizing IR from $\mathsf{TC}^{\mathsf{NL}}$

For each ctl_block, IRSynthesizer takes in the transition components ($\mathsf{TC}^{\mathsf{NL}}$s) extracted by NEUTREX and generates corresponding intermediate representations (IR) as logical formulas. FSMSynthesizer then uses the IRs to build FSMs.

The architecture of IRSynthesizer is illustrated in Figure 6. At first, it extracts keywords (e.g., messages, variables, events, etc.) from natural language specifications and links those keywords to $\mathsf{TC}^{\mathsf{NL}}$ strings with Keyword Extractor. After that, the DTGenerator converts these strings to dependency trees, which is used by IRTranslator for mapping the $\mathsf{TC}^{\mathsf{NL}}$s to DSL commands and extracting the arguments in order to finally generate logical IRs ($\mathsf{TC}^{\mathsf{IR}}$).

5.1 Domain-Specific Language

For IR, we consider the conditions as logical expressions (e.g., chan_ue_mme = AUTHENTICATION FAILURE) and actions as logical or arithmetic statements (e.g., counter := counter + 1). However, these expressions and statements are often not syntactically/structurally similar to their corresponding natural language texts (e.g., "if the UE returns AUTHENTICATION FAILURE", "increment counter"). As a result, direct translation from natural language to logical formulas is challenging. Further, the lack of generalizability of existing data-driven approaches (discussed in §3.3) makes it even more challenging to generate logical formulas directly from natural language in the cellular domain.

To address these disparities, we define a Domain Specific Language (DSL) to reliably translate $\mathsf{TC}^{\mathsf{NL}}$s to logical formulas. The DSL is partially shown in Figure 7. Hermes relies on DSL because its commands can encode logical and linear integer arithmetic expressions and logical operations, and allow the use of variables, constants, and quantifiers. For example, the NL string–"the *UE* shall *reset* the *service_req_attempt_counter*" can be mapped to the DSL command reset(UE, service_req_attempt_counter) and corresponding arguments, which can then be easily translated to UE_service_req_attempt_counter := 0. While defining the DSL, we observe that the operations or conditional checks usually occur as action verbs in $\mathsf{TC}^{\mathsf{NL}}$s. Thus, we extract action verbs as DSL command names from the specification using a neural parts-of-speech tagger [74], and examine the $\mathsf{TC}^{\mathsf{NL}}$s containing these verbs to identify required arguments. Finally, we manually define the output logical forms for each DSL rule and implement an interpreter to generate them.

5.2 Keyword Linking

Identifying arguments of DSL commands is an integral step in applying the DSL rules. These are various identifiers, such as variables, states, messages, etc., consisting of multiple natural language tokens, whereas an argument is a unit. Although 3GPP defines a few abbreviations, messages, etc., the list is not complete and often misses important identifiers (e.g., *"service request attempt counter"*). As manual extraction of these identifiers is tedious and error-prone, we take an automated approach to identify and link them in $\mathsf{TC}^{\mathsf{NL}}$ strings with a single token keyword, which can be used as arguments to DSL commands. We observe that the identifiers are usually noun phrases, so we use a neural phrase-structure constituency parser [59, 74] to identify all noun phrases. After further filtering, processing, and applying cellular specification-specific heuristics, we extract the desired identifiers and assign types to them by combining automated and manual efforts.

While linking single token keywords to identifiers in $\mathsf{TC}^{\mathsf{NL}}$s, we observe slight variations in identifiers with the same meaning (e.g., "tracking area updating accept" and "tracking area update accept"), smaller identifiers within larger ones (e.g., IMEI, IMEISV, and IMEISV request), typographic errors, and inconsistent uses of punctuation. To address these challenges, we utilize Levenshtein distance [57] to correctly identify the keyword. We also embed type information to the keywords for the next phases of IRSynthesizer.

5.3 Constructing Dependency Trees

As discussed in §3.3, DSL rules require specific arguments, and representing natural language transition components ($\mathsf{TC}^{\mathsf{NL}}$s) as a dependency tree helps identify these rules and

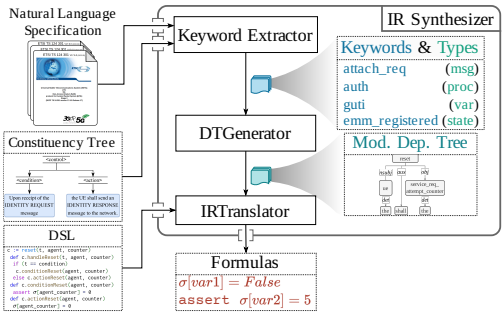

Figure 6: Architecture of `IRSynthesizer`.

```
m := receive(t, src, msg, dst)
def m.handleReceive(t, agent, msg, dst)
 if (t == condition)
  m.conditionForReceiveMsg(src, msg, dst)
 else m.actionForReceiveMsg(src, msg, dst)

def m.conditionForReceiveMsg(src, msg, dst)
 if (msg == constant)
  then assert σ[chan_src_dst] = msg
 elif (msg == logical_operator(msg1, msg2))
  logical_operator(
   m.conditionForReceiveMsg(src, msg1, dst),
   m.conditionForReceiveMsg(src, msg2, dst))

def m.actionForReceiveMsg(src, msg, dst)
  σ[chan_src_dst] = msg
```

```
c := reset(t, agent, counter)
def c.handleReset(t, agent, counter)
 if (t == condition)
  c.conditionForReset(agent, counter)
 else c.actionForReset(agent, counter)

def c.conditionForReset(agent, counter)
 assert σ[agent_counter] = 0

def c.actionForReset(agent, counter)
 σ[agent_counter] = 0
```

Figure 7: Definitions of DSL commands–receive and reset. σ denotes the variable store, and t denotes if the current use case is a condition or an action.

arguments. For that purpose, `DTGenerator` takes each $\mathsf{TC}^{\mathsf{NL}}$ with clearly mapped keywords and type information as inputs. As output, it generates representative dependency trees for DSL mapping.

Usage of dependency parse trees. The DSL commands and arguments are inherently tree-structured [16] and can be represented with abstract syntax trees (AST). Therefore, generating ASTs from natural language strings is naturally the first step toward mapping them to DSL. Based on this insight, `IRSynthesizer` first aims to map the sequential strings to DSL ASTs. We observe that *dependency parse trees* can discover internal grammatical relations (e.g., subject-verb, verb-object, etc.) between tokens of a natural language string and represent it as a tree that has main verbs, e.g., "send", "reset", etc., at the central node, and the arguments at its children. We leverage this hierarchical structure to map them to ASTs as it is similar to the DSL command and argument relations in the ASTs.

To demonstrate this insight, we consider the string "If the UE receives auth_reject or tau_reject", whose dependency tree is shown in Figure 8 (a). Conversely, the desired DSL command is receive(UE, or(auth_reject, tau_reject)), whose AST is shown in Figure 8 (c). These two trees have several common features, such as, *"receive"* is the root, *"UE"* is its child, and *"auth_reject"* and *"tau_reject"* are *receive*'s descendants. Based on this observation, we utilize the dependency tree to reach DSL rules from natural language strings.

Off-the-shelf dependency parsers are not enough for cellular specifications. Although dependency trees [23,74] largely correspond to our translation scheme, we require some modifications before adopting them for obtaining DSL rules. This is because of semantic differences in how conjunctions, prepositions, or adverbs are used in typical English sentences in contrast to formal languages. For example, "and", "or", "not", etc. are generally leaves of dependency trees in natural languages, while these words denote logical operations similar to verbs in formal languages. To illustrate, in Figure 8 (a),

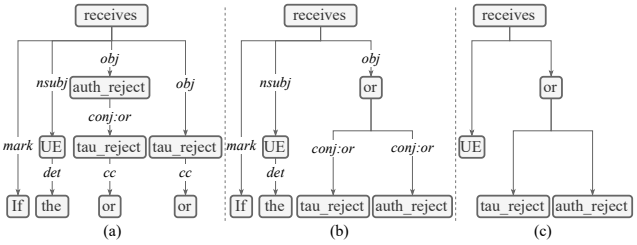

Figure 8: (a) Original dependency tree, (b) Modified dependency tree, (c) Abstract syntax tree.

the extracted parent-child relation between "auth_reject" and "tau_reject" is not applicable in a logical translation setting. Rather, "auth_reject" and "tau_reject" should be siblings under the "or" operation, as shown in the AST in Figure 8 (c).
Solution. Based on these observations, we perform a post-order traversal to find the logical operators and move the logically connected tokens under their subtree. We show the modified dependency tree for the discussed example in Figure 8 (b), where "or" is changed to the suitable position. Finally, the `DTGenerator` produces dependency trees for each $\mathsf{TC}^{\mathsf{NL}}$ and passes them to the `IRTranslator` for DSL mapping.

5.4 Synthesizing Intermediate Representation

The `IRTranslator` takes dependency trees corresponding to each transition component ($\mathsf{TC}^{\mathsf{NL}}$), maps these trees to DSL ASTs, and finally produces logical formulas or IR ($\mathsf{TC}^{\mathsf{IR}}$) for $\mathsf{TC}^{\mathsf{NL}}$s by following the DSL semantics.
Mapping dependency parse tree to AST. A dependency tree, although similar to the target AST, does not directly represent the target DSL command. For that, `IRTranslator` first identifies the corresponding DSL command and its arguments. Consequently, `IRTranslator` recursively traverses the dependency tree using a top-down approach, i.e., it matches the DSL command first and then the arguments. It also considers syntactic information from dependency

trees and type information from `Keyword Extractor` to map the arguments based on their type and produce the correct $\mathsf{TC}^{\mathsf{IR}}$. For the example in Figure 8, `IRTranslator` first matches the rule $\mathsf{receive}(\mathsf{t}, \mathsf{src}, \mathsf{msg}, \mathsf{dst})$ and recursively finds arguments. Using the subject-object relation and *agent* type information, `IRTranslator` decides the dst. The final output is "$\mathsf{assert}\ \sigma[\mathsf{chan\_ue\_mme}] = \mathsf{auth\_reject} \mid \mathsf{assert}\ \sigma[\mathsf{chan\_ue\_mme}] = \mathsf{tau\_reject}$", where σ is the variable store.

Diverging dependency trees. Natural language may have many forms of text expressing the same meaning, e.g., "The UE, after receiving auth_reject or tau_reject", "If auth_reject or tau_reject is received by the UE", etc. The corresponding dependency trees are structurally different as well. The first one has "UE" as the left child, whereas the other one has it as the right child of "receive". However, a reliable interpreter must generate the same $\mathsf{TC}^{\mathsf{IR}}$ for both. We ensure this by mapping dependency trees to DSL rules in a position-independent manner by using type information and syntactic information, e.g., part-of-speech and dependency relation. In this example, "UE" has type *agent*, and "auth_reject" and "tau_reject" have type *message*, which help map them as DSL arguments.

Cellular specification directives. Cellular specifications often use the directives– "may", "should", and "can", which signify that the operation is recommended or implementation-dependent [2]. Prior works [69] identify these recommendations as ambiguities and potential security flaws. `IRTranslator` addresses these directives by adding uniform random variables to the corresponding conditions [40, 42]. These variables enable an automated reasoner to non-deterministically explore and test all possible options, resulting in a comprehensive security analysis.

Addressing missing information. `IRTranslator` works with each $\mathsf{TC}^{\mathsf{NL}}$, but it may not contain all the information necessary to generate the corresponding $\mathsf{TC}^{\mathsf{IR}}$. For instance, in Figure 3, the condition–"the message is integrity protected" does not mention the message name, which needs to be inferred from the overall context. Conversely, in many cases, the `ctl_block`, or even the whole paragraph, may not be sufficient for accurately inferring the information (e.g., actions for cause #7 in Section 5.5.1.2.5 TS 24.301 [11]). Consequently, it needs to be inferred from previous paragraphs. For these reasons, we implement a context resolver in `IRTranslator` to find any missing information and produce the correct IR. For that, `IRTranslator` first checks the context of the concerned `ctl_block`, then the whole paragraph, and follows a heuristic backtracking mechanism to search previous ones.

Modeled actions and granularity of FSM. Modeled actions and the granularity of the extracted FSMs largely depend on the defined DSL rules and arguments. If necessary, Hermes can adopt more actions and more detailed modeling easily. In this work, we consider security-critical actions, such as sending/receiving messages, activating/deactivating services, incrementing counters and sequence numbers, starting timers, etc., totaling 94 DSL commands provided in [8]. In contrast, limitations in our logical abstraction do not allow us to model some actions, e.g., list modifications (adding or removing).

Further, Hermes models the specifications with more details than existing works [40, 42] by considering more message fields and the corresponding protocol behavior. Hermes also models 13 services (e.g., emergency bearer, EPS, etc.), 11 modes (e.g., EMM-CONNECTED, etc.), etc., along with states, procedures, counters, sequence numbers, timers, etc. However, compared to prior works [14, 29, 64, 71], Hermes does not model interactions among multiple base stations or multiple network functions in the core network. Instead, Hermes abstracts these interactions because the analyzed specifications mainly involve interactions between UE and AMF (or MME) and between UE and gNB. Certain procedures, such as handover, span multiple documents [71]. Although Hermes has the ability to extract an FSM from multiple documents, the current instantiation of Hermes extracts FSMs from individual documents. As a result, 5G NAS and 5G RRC FSMs provide a partial representation of the handover procedure.

6 Synthesizing FSM

The `FSMSynthesizer` combines the intermediate representations corresponding to the transition components ($\mathsf{TC}^{\mathsf{NL}}$) generated by `IRSynthesizer` to produce complete transitions. It also compiles all the transitions and performs logical checks to ensure the consistency of generated FSMs.

Constructing a transition by combining its components. As mentioned in §4.1, `ctl_block` is the primary construct for transitions, so `FSMSynthesizer` combines the IRs of each $\mathsf{TC}^{\mathsf{NL}}$ for a `ctl_block`. While doing so, it deals with multiple logically connected condition and action spans. For instance, the example in Figure 3 contains two condition spans connected with "or", and the whole condition is connected with another condition using the "and". `FSMSynthesizer` first finds these logical relations from the constituency parse tree, combines them into a single condition expression, and compiles the actions for that transition. Moreover, `FSMSynthesizer` identifies the start state and end state from corresponding IR from `IRSynthesizer` by considering them as conditions and actions, respectively. This is useful for logical connections (e.g., or, not, etc.) between the states. For the running example in Figure 3, after combining the outputs from the components, the transition with a condition, a start state, and an action is the output. Note that if the end state is not mentioned, the FSM remains in the same state.

Compiling all transitions. To generate a complete formal model, `FSMSynthesizer` follows the above procedure and constructs all possible transitions identified by `NEUTREX` in a given specification document. Further, as defined in §3.1, these documents describe protocol interactions among multiple participants (e.g., UE and AMF). Ac-

Component	Libraries Used	Lines of Code
Text extraction and cleaning	pdfplumber [77]	380
NEUTREX	Huggingface [9], NLTK [17]	1200+
Keyword Extractor	NLTK [17], Stanza [74], Levenshtein [57]	1343
DTGenerator	NLTK [17], Stanza [74]	378
IRTranslator		3209
FSMSynthesizer	Z3 [30]	1206
nuXmv transpiler	LTEInspector transpiler [40]	416
UEGEN		1000+

Table 1: Implementation efforts for Hermes components.

cordingly, FSMSynthesizer partitions the list of transitions and generates separate FSMs for each participant. Finally, FSMSynthesizer outputs each FSM as a graph, where the states are represented as unique nodes, and the transitions are represented as directed edges from the start state node to the end state node. We also label these edges with IR representations ($\mathsf{TC}^{\mathsf{IR}}$) of corresponding conditions and actions.

Addressing split transitions. Hermes takes input in natural language at the paragraph level since each paragraph in the cellular specifications usually discusses a particular scenario (start state, conditions, and corresponding actions). However, actions in these documents are oftentimes mentioned across multiple paragraphs. For instance, in Section 5.5.1.2.4 of 4G NAS specifications [11], when ATTACH REQUEST is accepted, the network behavior is discussed in multiple paragraphs, leading to a divided transition. As a result, formal analysis of the extracted FSM may not execute all of the actions, potentially failing to explore all possible paths. We observe two cases of such complicated transition pairs where the start state is the same. First, when the conditions in any two transitions T_1 and T_2 are logically equivalent ($C_1 \iff C_2$), we merge the transitions into one by combining the actions. Second, the conditions are not equivalent, but one implies the other, i.e., $\neg(C_1 \iff C_2) \wedge (C_1 \implies C_2)$ (e.g., $C_1 = x \wedge y$ and $C_2 = x$). Here, whenever C_1 is TRUE, both transitions T_1 and T_2 are valid. Thus, we extend the list of actions in T_1 with the actions from T_2 and change the condition in T_2 to $(C_2 \wedge \neg C_1)$. This ensures that the formal analysis explores all possible paths in the FSM.

Recovering missing transitions. NEUTREX sometimes fails to recognize a few action spans in natural language texts, leading to missing transitions. To recover such actions, we introduce a rule-based mechanism into FSMSynthesizer. Specifically, within a ctl_block, if we find a string with no act_tag, we scan it for action verbs and corresponding arguments covered in the action DSL commands of IRSynthesizer. If such a DSL command can be mapped, we consider the string as an action $\mathsf{TC}^{\mathsf{NL}}$, generate the corresponding $\mathsf{TC}^{\mathsf{IR}}$ with IRSynthesizer, and add it to the output FSM.

7 Implementation

We annotate cellular specification datasets with 4G-NAS Release16 [11], 5G-NAS Release17 [5], and 5G-RRC Release17 [6] documents according to the grammar shown in Figure 4. These three annotated documents have ∼16,000 datapoints, and it required a total of ∼2800 man-hours to complete the annotations, which are executed by four cellular systems researchers and verified by two domain experts.

For NEUTREX, we use neural CRF-based constituency parsing [93]. To train NEUTREX with the annotated documents, we use a learning rate of $lr = 5e^{-5}$ with a warmup of 0.1. During the initial stages of training, when the bracketing loss is unstable, we use $\lambda_1 = 0.1$ and $\lambda_2 = 0.9$. During the rest of the training, we use $\lambda_1 = \lambda_2 = 0.5$, giving equal importance to both loss components. All models are trained for 200 epochs, and the best model is chosen based on a held-out dev set. The summary of the implementation efforts is shown in Table 1.

8 Evaluation

To evaluate the performance of Hermes, we aim to answer the following research questions:

- **RQ1.** What is the efficacy of NEUTREX to automatically annotate a cellular specification document, and how it compares with existing baselines (§8.1)?
- **RQ2.** What is the efficacy of IRSynthesizer and FSMSynthesizer to translate $\mathsf{TC}^{\mathsf{NL}}$ to IR (§8.2)?
- **RQ3.** What is the accuracy of the Hermes extracted FSM, and how it compares with existing formal models (§8.3)?
- **RQ4.** How effective is the extracted FSM in identifying attacks and finding noncompliance (§8.4, §8.5, and §8.6)?
- **RQ5.** How efficient is Hermes in extracting FSMs and performing security analysis with these FSMs (§8.7)?

8.1 Effectiveness of NEUTREX

To answer RQ1, we compare NEUTREX with the BIO entity tagger in RFCNLP [67] in terms of identifying constituency spans. For this comparison, we report traditional precision, recall, and F1-score metrics [75]. We consider both unlabeled and labeled metrics– in **unlabeled** comparison, we use these metrics for the detected spans without any consideration of the labels assigned to them. In **labeled** comparison, we also take into consideration the labels assigned.

Results on cellular specifications. We compare RFCNLP [67] and NEUTREX using 3 different training and testing combinations of annotated documents reflecting the generalization capabilities of NEUTREX across layers, releases, and generations. We first train NEUTREX on annotated 4G-NAS-Release16 specifications [11] and evaluate on the 5G-NAS-Release17 [5]. Again, we train a separate NEUTREX model from scratch on 5G-NAS-Release17 specifications and evaluate on both 4G-NAS-Release16 and 5G-RRC-Release17 [6]. We consider each paragraph of the documents as separate datapoints during the training and the evaluation.

We show these results in Table 2, and NEUTREX significantly outperforms the BIO entity tagger in RFCNLP. We

	4G NAS → 5G NAS						5G NAS → 4G NAS						5G NAS → 5G RRC					
	UP	UR	UF	LP	LR	LF	UP	UR	UF	LP	LR	LF	UP	UR	UF	LP	LR	LF
RFCNLP	8.22	27.53	12.66	8.15	27.27	12.54	39.52	38.71	39.12	38.92	38.12	38.52	13.45	10.84	12.01	11.45	9.22	10.22
NEUTREX	66.88	68.79	67.82	64.30	66.13	65.20	71.38	71.28	71.33	68.25	68.15	68.20	72.81	74.44	73.62	67.94	69.45	68.69

Table 2: NEUTREX and RFCNLP entity tagger [67] results on cellular specifications. X → Y means we train on X and text on Y. **UP, UR, and UF** denotes unlabeled precision, recall, and F1-metrics. Similarly, **LP, LR, and LF** denotes labeled metrics.

	BGPv4, DCCP, LTP, PPTP, SCTP → TCP						BGPv4, LTP, PPTP, SCTP, TCP → DCCP					
	UP	UR	UF	LP	LR	LF	UP	UR	UF	LP	LR	LF
RFCNLP	60.03	55.05	57.43	49.92	45.78	47.76	38.75	39.00	38.88	33.80	34.02	33.91
NEUTREX	64.53	55.60	59.73	61.64	53.11	57.06	53.78	59.97	56.71	52.98	59.08	55.06

Table 3: NEUTREX and RFCNLP entity tagger results on the RFC dataset [67].

	5G NAS		4G NAS	
	Action	Condition	Action	Condition
Synthesizer FSM (M_{Syn})	93.86	94.45	92.23	92.24
Hermes FSM (M_{Hermes})	81.39	86.40	81.14	87.21

Table 4: Transition accuracy against M_{Gold}: M_{Syn} for IR & FSM synthesizers, M_{Hermes} for Hermes.

find that **BIO** entity tagging in RFCNLP has no explicit span handling mechanism, and thus, it suffers in the challenging cellular dataset containing complex structure and several levels of nested constituents. Moreover, RFCNLP breaks sentences into chunks using an off-the-shelf chunker, often providing erroneous results in highly technical cellular specifications. In contrast, NEUTREX shows stable performance across different generations, protocols, and releases of the cellular specifications, reaching $64-74\%$ precision, recall, and F1 score. This enables our method to be used in future generations of cellular systems as well. The lost accuracy is mainly due to the distribution shift resulting from the domain change (e.g., 4G to 5G or NAS to RRC) and the difficulty of the dataset stemming from the increasing degree of nested constituents, which complicates the semantic relationship between hierarchical spans. Additionally, compared to 4G NAS specifications, we find that there is a slight accuracy drop in 5G NAS, which can be attributed to the increased difficulty (length and nesting).

Results on the RFC dataset. To show NEUTREX's generalizability to other domains, we train and test it using the RFC dataset (TCP, DCCP, SCTP, PPTP, LTP, BGPv4) with the same dataset split as RFCNLP [67]. For comparison, we compute the same metrics on the predictions by RFCNLP [66]. Table 3 presents these results and shows that although NEUTREX is primarily designed for cellular specifications, it outperforms RFCNLP even in the RFC dataset. This manifests NEUTREX's applicability to other domains.

8.2 Effectiveness of IR and FSM Synthesizers

To answer RQ2, we use human-annotated ground truth transition components instead of using NEUTREX, and leverage the IRSynthesizer and FSMSynthesizer components of Hermes to synthesize transitions and generate FSMs (M_{Syn}). The annotations are from 4G NAS [11] and 5G NAS [5] specifications. We check M_{Syn}'s accuracy against the formal models manually constructed by prior works [40, 42]. These works, however, abstracted away a lot of details from the original specification. As a result, a head-to-head comparison with these models is not plausible. Thus, to evaluate M_{Syn}, we manually construct ground truth Gold FSMs (M_{Gold}) with the same actions and scope as these prior works but matching the same abstraction as M_{Syn}. We use the ground truth annotations of the same paragraphs within the scope of M_{Gold} to generate M_{Syn}. These include procedures covered by prior works, such as (de-)registration, service request, security mode control, etc. Finally, we compute the accuracy of conditions, actions, and states in the transitions of M_{Syn}, considering M_{Gold} as ground truth. The summary of the FSMs is provided in Table 5, and details of computing this accuracy are in Appendix A.

The results of M_{Syn} in Table 4 show $92-94\%$ accuracy of transitions extracted by IRSynthesizer and FSMSynthesizer as compared to the transitions manually crafted in M_{Gold}. This proves that our approach is better than the state-of-the-art framework on network protocols, RFCNLP [67], which achieves $67-80\%$ accuracy. The lost accuracy of $6-8\%$ in M_{Syn} can be attributed to the wide diversity of natural language strings, imperfections in Keyword Extractor, and off-the-shelf dependency parsers. Instead of overfitting the DSL rules to these rare cases, we strive to design IRSynthesizer generic to the cellular specifications across different generations, layers, and releases.

FSM	TC^{NL} Generated By	TC^{IR} Generated By
Gold FSM (M_{Gold})	Human	Human
Synthesizer FSM (M_{Syn})	Human	IRSynthesizer & FSMSynthesizer
Hermes FSM (M_{Hermes})	NEUTREX	IRSynthesizer & FSMSynthesizer

Table 5: Transition extraction for FSMs used in evaluation.

8.3 Accuracy of the Extracted FSM

We address RQ3 by extracting FSMs end-to-end with Hermes (M_{Hermes}) for 4G NAS [11] and 5G NAS [5] specifications. We compute the accuracy of conditions, actions, and states in Hermes FSM (M_{Hermes}) against the Gold FSM (M_{Gold}).

Annotated span	NEUTREX prediction	Generated IR
<control> <action> The network shall stop timer T3522 </action> <condition> upon receipt of the DEREGISTRATION ACCEPT message . </condition> </control>	<control> The network shall <action> stop timer T3522 </action> <condition> upon receipt of the DEREGISTRATION ACCEPT message . </condition> </control>	condition: chan_ue_amf = deregistration_accept actions: timer_t3522_started = FALSE
<control> The network shall, <condition> on the first expiry of the timer T3460, </condition> <action> retransmit the AUTHENTICATION REQUEST message </action> </control>	<control> <condition> <condition> The network shall, <condition> on the first expiry of the timer T3460, </condition> </condition> <action> retransmit the AUTHENTICATION REQUEST message </action> </control>	condition: timer_t3460_started & timer_t3460_expired & timer_t3460_expire_counter = 1 actions: chan_amf_ue = auth_request
<control> <condition> If the SECURITY MODE COMMAND message can be accepted, </condition> <action> the UE shall take the 5G NAS security context indicated in the message into use. </action> </control>	<control> If <condition> the SECURITY MODE COMMAND message can be accepted, </condition> <action> the UE shall take the 5G NAS security context indicated in the message into use. </action> </control>	condition: accept_sm_command actions: nas_security_context_update = TRUE, nas_security_context_valid = TRUE
<control> <condition> The UE, when receiving the DEREGISTRATION ACCEPT message, </condition> <control> <action> stop timer T3519 </action> <condition> if running, </condition> </control> </control>	<control> <condition> The UE, when receiving the DEREGISTRATION ACCEPT message, </condition> <control> stop timer T3519 <condition> if running, </condition> </control> </control>	condition: chan_amf_ue = deregistration_accept & timer_t3519_started actions: timer_t3519_started = FALSE

Table 6: Examples of spans with incorrect NEUTREX prediction, but correct IR [5, 11].

As shown in Table 4, M_{Hermes} has an overall accuracy of $81-87\%$, demonstrating the robustness of the approach because Hermes achieves a higher holistic accuracy than the F1-score of NEUTREX ($65-68\%$ in Table 2). This improvement can be attributed to cases where NEUTREX makes errors in identifying the conditions or actions span boundaries, but includes the action verbs and arguments in them (discussed in §5.4). In these cases, the rest of the framework is robust enough to capture the logical meanings of the identified TC^{NL}s and correctly produce the transitions. Moreover, FSMSynthesizer identifies strings in ctl_block without any tag and scans it to find action verbs and DSL arguments. A few such cases are shown in Table 6. However, despite the efforts in IRSynthesizer and FSMSynthesizer, we find that 16 transitions are missing in M_{Hermes}. We analyze such cases by consulting with the specification and find that they can be attributed to errors in NEUTREX and IRSynthesizer (details with examples are given in Table 7).

Conformance Testing. To further test the extracted FSM's accuracy, we translate M_{Hermes} to the nuXmv model following the procedure in §8.4 without instrumenting adversary capabilities into the model. Then, we translate 43 security-related conformance test cases provided by 3GPP to Linear Temporal Logic (LTL) properties and check them against the model. Violations of a property signify nonconformance/inaccuracies in M_{Hermes} transitions. We found that the 5G NAS FSM passes in 33 test cases and 4G NAS FSM passes in 32.

8.4 Effectiveness of FSMs Extracted by Hermes

To answer RQ4, we leverage the approach provided by prior works [40, 42], which automatically transpiles compatible FSMs to nuXmv [22] model and incorporate Dolev-Yao adversary capabilities [32] (drop, modify, inject, or replay) into communication channels based on sent/received messages. These adversary-instrumented nuXmv models are amenable to formal model checking of security properties on the FSMs. Thus, to ensure compatibility with the transpiler, Hermes outputs FSMs for cellular specification documents in a Graphviz-like format with conditions and actions as propositional logic expressions, which are then translated to nuXmv and checked against security properties. After performing the model checking, nuXmv spits out traces of counterexamples when any of the properties are violated.

Counterexample verification and vulnerability detection. A counterexample from nuXmv is basically a trace of message exchanges and internal variable updates leading to the violation of the property. However, FSMs, even manually crafted ones, often include erroneous transitions [40], as is the case with Hermes-extracted FSMs. Thus, after a counterexample is extracted, we manually verify it by consulting the specification document. Further, when we find an erroneous counterexample, we manually refine the model and correct the related incorrect transitions. This process is counterexample-guided model refinement, which facilitates further checking security properties on the refined model and identifying more counterexamples. Finally, we follow prior works [40, 42] and manually model the remaining counterexamples in ProVerif [18] to verify their feasibility under cryptographic assumptions.

Experiment setup, properties, and procedures. We extract separate FSMs from 4G-NAS-Release16 [11], 5G-NAS-Release17 [5], and 5G-RRC-Release17 [6] specifications, and generate separate adversary-instrumented nuXmv models. While extracting the 5G-NAS-Release17 FSM, we use NEUTREX trained with 4G-NAS-Release16 specification, and while extracting the FSMs from 4G-NAS-Release16 and 5G-RRC-Release17 specifications, we use NEUTREX trained with 5G-NAS-Release17 specification (the same setting as §8.1).

In this experiment, we check the 20 security properties from prior works [40, 42] and 7 new properties that cover the scope of other related works [20, 27, 28, 51, 70, 76, 80, 87] written in Linear Temporal Logic (LTL). Moreover, for model checking, adversary-instrumented nuXmv models are cre-

Cause	Example	#missing transitions
NEUTREX missed <action>	<control> <action> The MME shall initiate the GUTI reallocation ... </action> and starting the timer T3450. </control>	6
NEUTREX merged two separate <action>s together	<control> <action> The MME initiates the NAS security mode control ... and starting timer T3460 (see example in figure 5.4.3.2.1). </action> </control>	2
NEUTREX divided unit <action>	<control> <action> the UE shall include ... in SERVICE REQUEST message, </action>, or in the REGISTRATION REQUEST message. </control>	2
NEUTREX classified <action> as condition	<control> <condition> b) proceed with the pending procedure; and </condition> </control>	2
NEUTREX missed <control>	The detach procedure is initiated ... DETACH REQUEST message. The Detach type IE .. to a "switch off" or not.	2
DTGenerator produces wrong part-of-speech tag	a) enter 5GMM-IDLE mode;	1
Incomplete set of DSL rules in IRSynthesizer	a) the UE shall check the authenticity of the core network by means of the AUTN ... AUTHENTICATION REQUEST message.	1

Table 7: Examples of missing transitions [5, 11].

ated from the FSMs extracted from 4G-NAS-Release16 [11], 5G-NAS-Release17 [5], and 5G-RRC-Release17 [6] specifications. These models cover 9 NAS and 10 RRC layer procedures, including registration, deregistration, configuration update, service request, RRC Setup, Reconfiguration, paging, and Resume. The properties and procedures are listed in the full version of this paper [44].

Results. The formal analysis identifies 19 previous and 3 new vulnerabilities in the extracted FSMs. Table 8 shows the list of attacks and summarizes the findings of the security analysis on the extracted FSMs. We also list the previous vulnerabilities (within the scope of the analyzed specification) not detected by the analysis with Hermes FSMs.

Among the not identified ones, attacks #15 and #30 are due to errors in transition component extraction by NEUTREX, and attack #10 is due to underspecification of the specifications. For the rest of the non-identified attacks, a manual inspection uncovered that the corresponding vulnerabilities are not present in the analyzed release of the specifications, and thus, Hermes does not model those behaviors. For example, the 5G-RRC specification was updated due to the report of the AKA-Bypass attack [51], so the version we analyzed does not contain the related vulnerability. In addition, due to the imperfection in the extracted FSM, the counterexample verification process, as discussed above, finds 8 erroneous counterexamples from nuXmv model checker. We correct 47 transitions to refine the model through the counterexample-guided refinement process. On the other hand, attacks #20-21 and #35 are new attacks identified using Hermes FSMs. These new attacks are identified using properties from prior works [40, 42], which further bolsters that Hermes-generated FSMs exhibit larger scopes and details than prior manually crafted models. Note that similar issues as 20 and 21 are concurrently reported in change requests [39, 65], but these reports do not discuss the exploits.

ID	Attack	Gen/Layer	N	D
1	Downgrade to non-LTE network services [76]	4G NAS	✗	✓
2	Denying all network services [76]	4G NAS	✗	✓
3	Denying selected service [76]	4G NAS	✗	✗
4	Signaling DoS [15, 48, 55, 56]	4G NAS	✗	✓
5	S-TMSI catching [51]	4G NAS	✗	✓
6	IMSI catching [80]	4G NAS	✗	✓
7	EMM Information [70]	4G NAS	✗	✓
8	Impersonation attack [27]	4G NAS	✗	✗
9	Synchronization Failure attack [87]	4G NAS	✗	✗
10	Malformed Identity Request [63]	4G NAS	✗	✗
11	Neutralizing TMSI refreshment [42]	5G NAS	✗	✗
12	NAS Counter Reset [42]	5G NAS	✗	✓
13	Uplink NAS Counter Desynchronization [42]	5G NAS	✗	✓
14	Exposing NAS Sequence Number [42]	5G NAS	✗	✓
15	Cutting off the Device [42]	5G NAS	✗	✗
16	Exposure of SQN [20]	5G NAS	✗	✓
17	5G AKA DoS Attack [21]	5G NAS	✗	✓
18	SUCI catching [28]	5G NAS	✗	✗
19	IMSI cracking [41]	5G NAS	✗	✗
20	NAS COUNT update attack	5G NAS	✓	✓
21	Deletion of allowed CAG list	5G NAS	✓	✓
22	Downgrade using ATTACH REJECT/ REGISTRATION REJECT [76]	4G/5G NAS	✗	✓
23	AUTHENTICATION REJECT attack [87]	4G/5G NAS	✗	✓
24	DETACH REQUEST/ DEREGISTRATION REQUEST attack [40]	4G/5G NAS	✗	✓
25	SERVICE REJECT attack [76]	4G/5G NAS	✗	✓
26	Denial-of-Service with RRCSetupRequest attack [42]	5G RRC	✗	✗
27	Installing Null Cipher and Null Integrity [42]	5G RRC	✗	✓
28	Lullaby Attack [42]	5G RRC	✗	✓
29	Incarceration with RRCReject and RRCRelease [42]	5G RRC	✗	✓
30	Measurement report [76]	5G RRC	✗	✗
31	RLF report [76]	5G RRC	✗	✓
32	Blind DoS attack [51]	5G RRC	✗	✗
33	AKA bypass [51]	5G RRC	✗	✗
34	Paging channel hijacking [40]	5G RRC	✗	✗
35	Energy Depletion with RRCSetup	5G RRC	✓	✓

Table 8: Vulnerabilities identified in Hermes extracted FSMs. N: New, D: Detected.

8.4.1 Identified Three New Vulnerabilities

(1) Deletion of allowed CAG list. When a UE in a closed access group (CAG) cell sends a REGISTRATION REQUEST message through a gNB, the AMF determines whether any of the CAG ID(s) from the gNB are in the list of Allowed CAGs for that UE. In case this check fails, the AMF rejects the registration and sends a REGISTRATION REJECT message with 5GMM cause #76. Upon receiving this message, if the integrity check is successful, the UE deletes its allowed CAG list [5]. An adversary can exploit this behavior and force the UE to delete the allowed CAG list, resulting in a denial of service (DoS).

Root cause. The AMF sends the REGISTRATION REJECT based on the received information from the gNB. However, the UE has no means to verify the CAG IDs for which the AMF sends the REGISTRATION REJECT message. Without this knowledge, the UE cannot identify if there is a mismatch in CAG ID(s) when it receives the REGISTRATION REJECT with 5GMM cause #76. Thus, the vulnerability stems from a design flaw.

Attack. The adversary, at first, lets the victim UE connect to a benign base station (gNB_1) and complete registration with the AMF. After that, the adversary lures the victim to connect to a machine-in-the-middle (MitM) fake base station. She then finds a gNB with a different set of CAG IDs (gNB_2) and relays the REGISTRATION REQUEST to it (Figure 9). gNB_2 for-

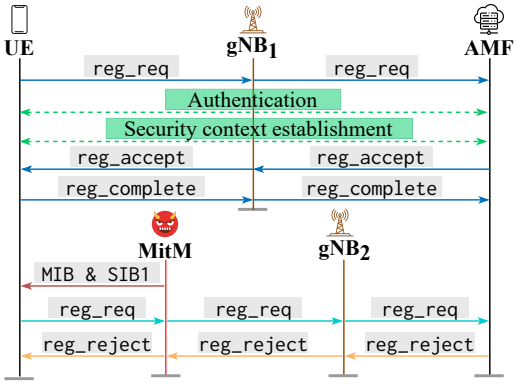

Figure 9: Deletion of allowed CAG list.

wards the message to the benign AMF, and the AMF checks the CAG list from gNB_2. After finding a mismatch, the AMF responds with integrity-protected REGISTRATION REJECT with 5GMM cause #76. By forwarding this message to the UE, the adversary forces it to delete the allowed CAG list. Note that the initial registration through gNB_1 ensures that the AMF responds with integrity-protected REGISTRATION REJECT. Without integrity protection, the UE does not accept this message [5].

Impact. The deletion of the allowed CAG list results in DoS. Without the allowed CAG list, the UE cannot get access to the non-public network (NPN) that it is originally permitted to. Repeating this attack will cause a prolonged DoS.

(2) Energy Depletion using RRCSETUP. The RRC resume procedure, introduced in 5G, strives to reduce power consumption and connection setup overhead. However, during this procedure, the UE accepts insecure RRCSETUP messages in response to RRCRESUMEREQUEST. Upon receiving this message, the UE deletes RRC security contexts and releases radio bearers. Reestablishing a secure RRC connection requires expensive cryptographic operations, leading to energy depletion.

Root cause. The root cause of this vulnerability can be attributed to the specification allowing non-integrity-protected RRCSETUP messages in response to the RRCRESUMEREQUEST [6].

Attack. As the RRCSETUP messages are not integrity protected nor ciphered, an adversary can easily craft a plaintext RRCSETUP message. Thus, she listens for an RRCRESUMEREQUEST message in the communication channel. After that, as shown in Figure 10, when she detects an RRCRESUME message from the legitimate base station, she drops the message and spoofs a plaintext RRCSETUP. As a result, upon receiving this message, the UE deletes its RRC security contexts and releases radio bearers.

Impact. By interrupting the RRC resume procedure, the adversary forces the UE to perform redundant, expensive cryptographic operations to set up the security context again. This leads to extraneous energy consumption. An adversary can repeatedly perform this attack and cause battery depletion.

(3) NAS COUNT update attack. 5G NAS specification [5] allows a UE to store the estimated NAS COUNT, i.e., the NAS

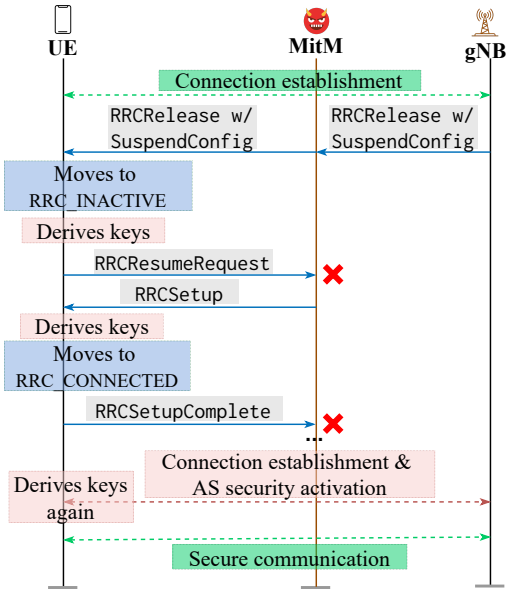

Figure 10: Energy depletion using RRCSetup.

COUNT used in the *last* successfully integrity-verified NAS message. However, the adversary can exploit this behavior to manipulate the NAS COUNT and perform replay attacks.

Root cause. The root cause is a design flaw– UE stores the NAS COUNT of the *last* integrity-verified message [5]. This allows the UE to accept integrity-protected NAS messages that have been delayed and are received out of sequence.

Attack. The adversary sets up a MitM fake base station (gNB) between the victim UE and a benign AMF. Then, she can arbitrarily delay any message to update the NAS COUNT to a lower value. For example, in Figure 11, the NAS COUNT of msg_1 is $X||a$, where X is the overflow counter, a is the sequence number, and $||$ denotes concatenation. The adversary delays msg_2 (NAS COUNT: $X||a+1$) and sends it after msg_4 (NAS COUNT: $X||a+3$). This message (msg_2) is valid and integrity-protected by the AMF. Thus, this out-of-sequence message causes the victim to update its NAS COUNT to $X||a+1$. This opens the scope of replayed messages with NAS COUNTs $X||a+2$ and $X||a+3$.

Impact. As the UE becomes vulnerable to replayed NAS messages, the adversary can exploit this to perform traceability attacks to extract a victim's presence in a target area [12, 14, 42]. This violates security requirements in 5G networks [7].

8.5 Analysis of Unannotated Document

We also analyze an unannotated document, i.e., 5G-NAS-Release15 [3], by using NEUTREX trained on the 4G-NAS-Release16 and extracting the FSM of 5G-NAS-Release15. Following the formal analysis technique discussed in §8.4, we check the same properties and procedures. This analysis finds the attacks migrating from 4G to 5G (attacks #22-25

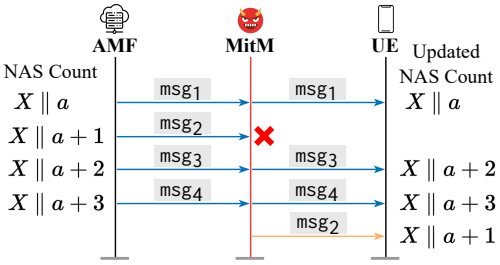

Figure 11: NAS COUNT update attack.

ID	Deviation	Detected
1	Replayed GUTI reallocation at specific sequence	✓
2	Replayed GUTI reallocation anytime	✓
3	Plaintext AUTHENTICATION REQUEST	✗
4	Plaintext IDENTITY REQUEST	✗
5	Selective replay of SECURITY MODE COMMAND	✓
6	GUTI reallocation before attach procedure completed	✓
7	AUTHENTICATION RESPONSE after SECURITY MODE REJECT	✓
8	AUTHENTICATION FAILURE reply	✓
9	Replayed SECURITY MODE COMMAND	✓

Table 9: Identified deviations in 4G UE implementations [43].

in Table 8), and five previous 5G attacks (attacks #12-14 and #16-17 in Table 8). The analysis also found 5 spurious counterexamples, handled similarly as discussed in §8.4.

8.6 Analysis of Cellular Implementations

To further address RQ4, we analyze the security of 4G NAS FSM [11] of 9 cellular baseband implementations (M_I) using M_{Hermes}. We run DIKEUE's equivalence checker [43] between M_{Hermes} and M_I and detect exploitable deviations in M_I from M_{Hermes}. As shown in Table 9, M_{Hermes} identifies 7 deviations, which were also identified by DIKEUE. However, the other two deviations that M_{Hermes} misses (#3 & #4), but DIKEUE detects result from conflicts between two separate documents [43]. In addition, due to imperfect FSM, we face 3 false positive deviations, which we manually verify.

8.7 Efficiency of Hermes

To address RQ5, we compute the time required for FSM extraction and subsequent security analysis by Hermes. On an NVIDIA RTX A6000 machine, training `NEUTREX` takes ∼4 hours, and prediction takes less than a minute for each document. `Keyword Extractor` takes 8-12 hours of computation on a machine with an Intel Core i7-10610U processor and 48 GB RAM and around 5 hours of manual effort. `IRSynthesizer` and `FSMSynthesizer` require 1.5-2 hours, and formal model checking requires 1-2 minutes. Finally, manually verifying the counterexamples requires ∼5 hours for each FSM. These requirements are detailed in Table 10.

Component	Document	Time required
`NEUTREX` training	4G-NAS-Release16	4 hour 3 minutes
	5G-NAS-Release17	4 hour 9 minutes
`NEUTREX` prediction	4G-NAS-Release16	42 seconds
	5G-NAS-Release17	33 seconds
	5G-NAS-Release15	16 seconds
	5G-RRC-Release17	19 seconds
`Keyword Extractor` computation	4G-NAS-Release16	13 hour 6 minutes
	5G-NAS-Release17	13 hour 39 minutes
	5G-RRC-Release17	7 hour 8 minutes
Keyword type assignment	4G-NAS-Release16	5 hour 27 minutes
	5G-NAS-Release17	5 hour 19 minutes
	5G-RRC-Release17	5 hour 46 minutes
`IRSynthesizer` & `FSMSynthesizer`	4G-NAS-Release16	43 minutes
	5G-NAS-Release17	43 minutes
	5G-NAS-Release15	35 minutes
	5G-RRC-Release17	12 minutes
Formal model checking	4G-NAS-Release16	78 seconds
	5G-NAS-Release17	117 seconds
	5G-NAS-Release15	106 seconds
	5G-RRC-Release17	67 seconds
Counterexample verification	4G-NAS-Release16	5 hour 24 minutes
	5G-NAS-Release17	5 hour 39 minutes
	5G-NAS-Release15	6 hour 16 minutes
	5G-RRC-Release17	2 hour 34 minutes

Table 10: Time requirement for FSM extraction and analysis.

9 Related Work

Security Protocol Analysis Using NLP Methods. RFC-NLP [67] extracts FSMs from RFC documents with a BIO entity tagging mechanism. However, it fails to address nested constituents nor identify control blocks. Some works [46, 85] also parse RFCs to extract source code, ambiguities, or test cases. Chen et al. [25] analyze payment system guidelines to find logical flaws. Several works also leverage explicit rules to extract permissions, inputs, or ontology from software documentation [68, 83, 84]. Atomic [26] finds proof-of-concept of vulnerabilities using hazard identifiers, while CREEK [24] identifies security-relevant change request documents. These works do not extract any formal model of the analyzed system. Another group of works uses NLP tools and program analysis on source code to extract logical errors or formal models [58, 91, 92]. In contrast, Hermes aims to build a formal representation from the specifications.

Text-to-logic conversion Using NLP. Several works use semantic parsing to transform natural language into logical forms, such as lambda calculus [90], meaning representation [13], programming language [45, 88, 94], and first-order logic [36]. Variations of neural encoder-decoder models have been adopted for these tasks [34, 78, 86]. However, these works are not easily applicable to the extraction of formal representations from cellular specifications because of the domain gap, lack of cellular text-to-logic dataset, or increased complexity.

Security Analysis of Cellular Specifications. Several works manually build formal models from cellular specifications [14, 29, 40, 42, 64, 71], and use model checkers to detect vul-

nerabilities. Some works also extract formal models from white-box [49] or black-box [43] cellular implementations. In contrast, Hermes generates a formal model from the specifications. Further, LTEFuzz [51] generates test cases from security properties. DoLTEst [69] provides a comprehensive negative testing framework for UE implementations using properties and security-relevant states. Moreover, several works test baseband implementations [35, 50, 61, 82]. These works focus on testing cellular systems but do not build formal models, which is the primary objective of this work.

10 Discussion and Limitations

❏ **Underspecifications, ambiguities, and conflicts.** Hermes does not deal with identifying underspecifications (e.g., implicit reject, unspecified reject cause, etc. [69]), and we consider this orthogonal to this work. For ambiguities stemming from directives, e.g., "should", "may", etc., in cellular specifications, Hermes utilizes uniform random variables, and resulting security analysis can consider all possibilities equally likely, uncovering potential security flaws. For ambiguities or inconsistencies originating from table information or information from multiple specification documents, we consider these as our future work as we currently do not incorporate table data into the generated FSM or perform security analysis on FSMs from multiple documents.

❏ **Multiple documents.** One of the challenges of extracting FSMs for cellular systems is that the protocols are described across multiple documents. Although Hermes has the capabilities to handle text from multiple documents and extract the corresponding transitions constituting the full FSM, we restrict our experiments to single documents to facilitate comparison with prior works [67]. In the future, we plan to incorporate multiple documents and evaluate Hermes.

❏ **Manual efforts.** In `Keyword Extractor`, after automatically extracting the keywords from a document and assigning types to most of them, the user needs to assign types for the rest. This effort is significantly less than manually extracting and typing *all* the keywords from scratch. For new functionalities or operations, the user needs to define the syntax and semantics for new DSL rules. Furthermore, security analysis on the extracted models requires manual efforts to define security properties and to analyze model checker outputs. In case a spurious counterexample is identified, relevant transitions are manually traced and fixed according to the specifications. Further, to check the feasibility of the counterexamples, we manually model them in ProVerif [18].

❏ **Utility of Hermes generated imperfect FSM.** In the absence of 3GPP-defined standard formal models, prior research focuses on manually modeling the specifications and verifying security properties [14, 29, 40, 42, 64, 71]. However, due to frequent updates, changing the formal models demands strenuous efforts. Conversely, although imperfect, Hermes-generated FSMs can significantly reduce this effort. When the formal model checker provides any output, we manually verify them by consulting the specifications and detecting any errors. In case of errors, we track the relevant transitions in the FSM and correct them according to the specification. Compared to manually building the FSM from scratch, which may also entail errors [40], this effort is reasonable.

❏ **No gold FSM.** If the standard body, i.e., 3GPP provides a gold FSM and it is sufficiently detailed for comprehensive formal analysis, it could potentially provide better results than hand-crafted or automatically extracted FSMs. However, 3GPP does not provide any such FSM for cellular protocols. In fact, 3GPP plans against that since constructing such a gold FSM is extremely tedious, often error-prone, and may cause inconvenience in handling interoperabilities among stakeholders as the specifications leave many details to the developers' discretion.

11 Conclusion and Future Work

We present Hermes, a novel framework that automatically extracts FSMs from cellular specifications. Its `NEUTREX`, leveraging `CellulaRoBERTa`, provides large improvements to capture transition components in natural language compared to existing works. The transitions generated by Hermes achieve 81-87% accuracy, and security analysis on the extracted FSMs enables us to identify 3 new attacks and 19 previous attacks exploiting design flaws. For future work, we will extend Hermes to detect underspecifications and conflicting sentences in cellular specifications. We will also incorporate multiple documents while extracting FSMs with Hermes.

Acknowledgements

We thank the anonymous reviewers and the shepherd for their feedback and suggestions. We also thank GSMA for cooperating with us during the responsible disclosure. This work has been supported by the NSF under grants 2145631, 2215017, and 2226447, the Defense Advanced Research Projects Agency (DARPA) under contract number D22AP00148, and the NSF and Office of the Under Secretary of Defense– Research and Engineering, ITE 2326898, as part of the NSF Convergence Accelerator Track G: Securely Operating Through 5G Infrastructure Program.

References

[1] *3GPP Standard.* www.3gpp.org.

[2] *3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Specification drafting rules (Release 17).* [Online]. Available: http://www.3gpp.org/dynareport/21801.htm.

[3] *5G; Non-Access-Stratum (NAS) protocol for 5G System (5GS); Stage 3 (3GPP TS 24.501 version 15.7.0 Release 15).* [Online]. Available: http://www.3gpp.org/dynareport/24501.htm.

[4] *5G; Non-Access-Stratum (NAS) protocol for 5G System (5GS); Stage 3 (3GPP TS 24.501 version 16.5.1 Release 16)*. [Online]. Available: http://www.3gpp.org/dynareport/24501.htm.

[5] *5G; Non-Access-Stratum (NAS) protocol for 5G System (5GS); Stage 3 (3GPP TS 24.501 version 17.8.0 Release 17)*. [Online]. Available: http://www.3gpp.org/dynareport/24501.htm.

[6] *5G; NR; Radio Resource Control (RRC); Protocol specification (3GPP TS 38.331 version 17.0.0 Release 17)*. [Online]. Available: http://www.3gpp.org/dynareport/38331.htm.

[7] *5G; Security architecture and procedures for 5G System (3GPP TS 33.501 version 17.5.0 Release 17)*. [Online]. Available: http://www.3gpp.org/dynareport/33501.htm.

[8] Hermes. https://github.com/SyNSec-den/hermes-spec-to-fsm.

[9] Hugging face – the ai community building the future. https://huggingface.co/.

[10] *Universal Mobile Telecommunications System (UMTS); LTE; 5G; Non-Access-Stratum (NAS) protocol for Evolved Packet System (EPS); Stage 3 (3GPP TS 24.301 version 12.6.0 Release 12)*. [Online]. Available: http://www.3gpp.org/dynareport/24301.htm.

[11] *Universal Mobile Telecommunications System (UMTS); LTE; 5G; Non-Access-Stratum (NAS) protocol for Evolved Packet System (EPS); Stage 3 (3GPP TS 24.301 version 16.8.0 Release 16)*. [Online]. Available: http://www.3gpp.org/dynareport/24301.htm.

[12] Myrto Arapinis, Loretta Mancini, Eike Ritter, Mark Ryan, Nico Golde, Kevin Redon, and Ravishankar Borgaonkar. New privacy issues in mobile telephony: Fix and verification. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, 2012.

[13] Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. Abstract meaning representation for sembanking. In *Proceedings of the 7th linguistic annotation workshop and interoperability with discourse*, pages 178–186, 2013.

[14] David Basin, Jannik Dreier, Lucca Hirschi, Saša Radomirovic, Ralf Sasse, and Vincent Stettler. A formal analysis of 5g authentication. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery.

[15] Ramzi Bassil, Imad H Elhajj, Ali Chehab, and Ayman Kayssi. Effects of signaling attacks on lte networks. In *2013 27th International Conference on Advanced Information Networking and Applications Workshops*, pages 499–504. IEEE, 2013.

[16] Evert Willem Beth. *Semantic Entailment and Formal Derivability*. Noord-Hollandsche, 1955.

[17] Steven Bird. Nltk: The natural language toolkit. In *Proceedings of the COLING/ACL on Interactive Presentation Sessions*, COLING-ACL '06, USA, 2006. Association for Computational Linguistics.

[18] Bruno Blanchet et al. Modeling and verifying security protocols with the applied pi calculus and proverif. *Foundations and Trends® in Privacy and Security*, 1(1-2):1–135, 2016.

[19] Leonard Bolc and Tomasz Strzalkowskl. Transformation of natural language into logical formulas. In *Coling 1982: Proceedings of the Ninth International Conference on Computational Linguistics*, 1982.

[20] Ravishankar Borgaonkar, Lucca Hirschi, Shinjo Park, and Altaf Shaik. New privacy threat on 3g, 4g, and upcoming 5g aka protocols. *Proceedings on Privacy Enhancing Technologies*, 2019(3):108–127, 2019.

[21] Jin Cao, Maode Ma, Hui Li, Ruhui Ma, Yunqing Sun, Pu Yu, and Lihui Xiong. A survey on security aspects for 3gpp 5g networks. *IEEE Communications Surveys & Tutorials*, 22(1):170–195, 2020.

[22] Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. The nuxmv symbolic model checker. In *Computer Aided Verification*. Springer International Publishing, 2014.

[23] Danqi Chen and Christopher D Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*.

[24] Yi Chen, Di Tang, Yepeng Yao, Mingming Zha, XiaoFeng Wang, Xiaozhong Liu, Haixu Tang, and Dongfang Zhao. Seeing the forest for the trees: Understanding security hazards in the 3GPP ecosystem through intelligent analysis on change requests. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, 2022.

[25] Yi Chen, Luyi Xing, Yue Qin, Xiaojing Liao, XiaoFeng Wang, Kai Chen, and Wei Zou. Devils in the guidance: Predicting logic vulnerabilities in payment syndication services through automated documentation analysis. In *Proceedings of the 28th USENIX Conference on Security Symposium*, SEC'19, page 747–764, USA, 2019. USENIX Association.

[26] Yi Chen, Yepeng Yao, XiaoFeng Wang, Dandan Xu, Chang Yue, Xiaozhong Liu, Kai Chen, Haixu Tang, and Baoxu Liu. Bookworm game: Automatic discovery of lte vulnerabilities through documentation analysis. In *2021 IEEE Symposium on Security and Privacy (SP)*, 2021.

[27] Merlin Chlosta, David Rupprecht, Thorsten Holz, and Christina Pöpper. Lte security disabled: Misconfiguration in commercial networks. In *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*. Association for Computing Machinery, 2019.

[28] Merlin Chlosta, David Rupprecht, Christina Pöpper, and Thorsten Holz. 5g suci-catchers: still catching them all? In *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 359–364, 2021.

[29] Cas Cremers and Martin Dehnel-Wild. Component-based formal analysis of 5g-aka: Channel assumptions and session confusion. In *26th Annual Network and Distributed System Security Symposium, NDSS, 2019*. The Internet Society.

[30] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[31] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, June 2019.

[32] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[33] Li Dong and Mirella Lapata. Coarse-to-fine decoding for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

[34] Li Dong and Mirella Lapata. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2016.

[35] Nico Golde and Daniel Komaromy. Breaking band: reverse engineering and exploiting the shannon baseband. *Recon 2016, Recon*, 2016.

[36] Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Luke Benson, Lucy Sun, Ekaterina Zubova, Yujie Qiao, Matthew Burtell, David Peng, Jonathan Fan, Yixin Liu, Brian Wong, Malcolm Sailor, Ansong Ni, Linyong Nan, Jungo Kasai, Tao Yu, Rui Zhang, Shafiq Joty, Alexander R. Fabbri, Wojciech Kryscinski, Xi Victoria Lin, Caiming Xiong, and Dragomir Radev. Folio: Natural language reasoning with first-order logic. *arXiv preprint arXiv:2209.00840*.

[37] James Henderson. Inducing history representations for broad coverage statistical parsing. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 103–110, 2003.

[38] John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. Introduction to automata theory, languages, and computation. *Acm Sigact News*, 32(1):60–65, 2001.

[39] Huawei and HiSilicon. *CHANGE REQUEST: Prevention of deletion of allowed CAG list due to man in middle attack*. [Online].

[40] Syed Rafiul Hussain, Omar Chowdhury, Shagufta Mehnaz, and Elisa Bertino. Lteinspector: A systematic approach for adversarial testing of 4g LTE. In *25th Annual Network and Distributed System Security Symposium, NDSS , 2018*. The Internet Society.

[41] Syed Rafiul Hussain, Mitziu Echeverria, Omar Chowdhury, Ninghui Li, and Elisa Bertino. Privacy attacks to the 4g and 5g cellular paging protocols using side channel information. *Proceedings 2019 Network and Distributed System Security Symposium, NDSS*, 2019.

[42] Syed Rafiul Hussain, Mitziu Echeverria, Imtiaz Karim, Omar Chowdhury, and Elisa Bertino. 5greasoner: A property-directed security and privacy analysis framework for 5g cellular network protocol. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery.

[43] Syed Rafiul Hussain, Imtiaz Karim, Abdullah Al Ishtiaq, Omar Chowdhury, and Elisa Bertino. Noncompliance as deviant behavior: An automated black-box noncompliance checker for 4g lte cellular devices. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery.

[44] Abdullah Al Ishtiaq, Sarkar Snigdha Sarathi Das, Syed Md Mukit Rashid, Ali Ranjbar, Kai Tu, Tianwei Wu, Zhezheng Song, Weixuan Wang, Mujtahid Akon, Rui Zhang, and Syed Rafiul Hussain. Hermes: Unlocking security analysis of cellular network protocols by synthesizing finite state machines from natural language specifications. *arXiv preprint arXiv:2310.04381*.

[45] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. Learning a neural semantic parser from user feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2017.

[46] Samuel Jero, Maria Leonor Pacheco, Dan Goldwasser, and Cristina Nita-Rotaru. Leveraging textual specifications for grammar-based fuzzing of network protocols. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 9478–9483, 2019.

[47] Ran Ji and Jianmin Ji. Transferring knowledge from structure-aware self-attention language model to sequence-to-sequence semantic parsing. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 3164–3174, October 2022.

[48] Georgios Kambourakis, Constantinos Kolias, Stefanos Gritzalis, and Jong Hyuk Park. Dos attacks exploiting signaling in umts and ims. *Computer Communications*, 34(3):226–235, 2011.

[49] Imtiaz Karim, Syed Rafiul Hussain, and Elisa Bertino. Prochecker: An automated security and privacy analysis framework for 4g lte protocol implementations. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pages 773–785, July 2021.

[50] Eunsoo Kim, Dongkwan Kim, CheolJun Park, Insu Yun, and Yongdae Kim. Basespec: Comparative analysis of baseband software and cellular specifications for l3 protocols. *Proceedings 2021 Network and Distributed System Security Symposium, NDSS*, 2021.

[51] Hongil Kim, Jiho Lee, Eunkyu Lee, and Yongdae Kim. Touching the untouchables: Dynamic security analysis of the lte control plane. In *2019 IEEE Symposium on Security and Privacy (SP)*, May 2019.

[52] Nikita Kitaev, Steven Cao, and Dan Klein. Multilingual constituency parsing with self-attention and pre-training. *arXiv preprint arXiv:1812.11760*, 2018.

[53] Nikita Kitaev and Dan Klein. Constituency parsing with a self-attentive encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, July 2018.

[54] Jayant Krishnamurthy, Pradeep Dasigi, and Matt Gardner. Neural semantic parsing with type constraints for semi-structured tables. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

[55] Patrick PC Lee, Tian Bu, and Thomas Woo. On the detection of signaling dos attacks on 3g/wimax wireless networks. *Computer Networks*, 53(15):2601–2616, 2009.

[56] Wai Kay Leong, Aditya Kulkarni, Yin Xu, and Ben Leong. Unveiling the hidden dangers of public ip locationes in 4g/lte cellular data networks. In *ACM Workshop on Mobile Computing Systems and Applications (HotMobile). ACM*, 2014.

[57] Vladimir I Levenshtein et al. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10.

[58] Zhiqiang Lin, Xuxian Jiang, Dongyan Xu, and X. Zhang. Automatic protocol format reverse engineering through context-aware monitored execution. In *Network and Distributed System Security Symposium, NDSS*, 2008.

[59] Jiangming Liu and Yue Zhang. In-order transition-based constituent parsing. *Transactions of the Association for Computational Linguistics*, 5:413–424, 2017.

[60] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[61] Dominik Maier, Lukas Seidel, and Shinjo Park. Basesafe: Baseband sanitized fuzzing through emulation. In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '20, NY, USA, 2020. Association for Computing Machinery.

[62] Kenneth L McMillan. The smv system. In *Symbolic Model Checking*, pages 61–85. Springer, 1993.

[63] Benoit Michau and Christophe Devine. How to not break lte crypto. In *ANSSI Symposium sur la sécurité des technologies de l'information et des communications (SSTIC)*, 2016.

[64] Rhys Miller, Ioana Boureanu, Stephan Wesemeyer, and Christopher J. P. Newton. The 5g key-establishment stack: In-depth formal verification and experimentation. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, 2022.

[65] OPPO, Huawei, HiSilicon, Vodafone, Apple, and Deutsche Telekom. *CHANGE REQUEST: Contradictory requirements on update of local NAS COUNT*. [Online].

[66] Maria Lenore Pacheco, Max von Hippel, Ben Weintraub, Dan Goldwasser, and Cristina Nita-Rotaru. Rfcnlp/rfcnlp: Open-source code for rfcnlp paper. https://github.com/RFCNLP/RFCNLP.

[67] Maria Lenore Pacheco, Max von Hippel, Ben Weintraub, Dan Goldwasser, and Cristina Nita-Rotaru. Automated attack synthesis by extracting finite state machines from protocol specification documents. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022.

[68] Rahul Pandita, Xusheng Xiao, Wei Yang, William Enck, and Tao Xie. Whyper: Towards automating risk assessment of mobile applications. In *Proceedings of the 22nd USENIX Conference on Security*, SEC'13.

[69] CheolJun Park, Sangwook Bae, BeomSeok Oh, Jiho Lee, Eunkyu Lee, Insu Yun, and Yongdae Kim. DoLTEst: In-depth downlink negative testing framework for LTE devices. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, August 2022.

[70] Shinjo Park, Altaf Shaik, Ravishankar Borgaonkar, and Jean-Pierre Seifert. White rabbit in mobile: Effect of unsecured clock source in smartphones. In *Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '16.

[71] Aleksi Peltonen, Ralf Sasse, and David Basin. A comprehensive formal analysis of 5g handover. In *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '21.

[72] Hoifung Poon. Grounded unsupervised semantic parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 933–943, August 2013.

[73] Hoifung Poon and Pedro Domingos. Unsupervised semantic parsing. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 1–10, Singapore, August 2009.

[74] Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. Stanza: A python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*.

[75] Satoshi Sekine and Michael Collins. Evalb bracket scoring program. http://www.cs.nyu.edu/cs/projects/proteus/evalb, 1997.

[76] Altaf Shaik, Jean-Pierre Seifert, Ravishankar Borgaonkar, N. Asokan, and Valtteri Niemi. Practical attacks against privacy and availability in 4g/lte mobile communication systems. In *23rd Annual Network and Distributed System Security Symposium, 2016*. The Internet Society.

[77] Jeremy Singer-Vine. jsvine/pdfplumber: Plumb a pdf for detailed information about each char, rectangle, line, et cetera — and easily extract text and tables. https://github.com/jsvine/pdfplumber.

[78] Hrituraj Singh, Milan Aggrawal, and Balaji Krishnamurthy. Exploring neural models for parsing natural language into first-order logic. *arXiv preprint arXiv:2002.06544*, 2020.

[79] Richard Socher, John Bauer, Christopher D Manning, and Andrew Y Ng. Parsing with compositional vector grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 455–465, 2013.

[80] Fabian van den Broek, Roel Verdult, and Joeri de Ruiter. Defeating imsi catchers. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, 2015.

[81] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017.

[82] Ralf-Philipp Weinmann. Baseband attacks: Remote exploitation of memory corruptions in cellular protocol stacks. In *Proceedings of the 6th USENIX Conference on Offensive Technologies*, WOOT'12, 2012.

[83] René Witte, Qiangqiang Li, Yonggang Zhang, and Juergen Rilling. Text mining and software engineering: an integrated source code and document analysis approach. *IET Softw.*, 2:3–16, 2008.

[84] Edmund Wong, Lei Zhang, Song Wang, Taiyue Liu, and Lin Tan. Dase: Document-assisted symbolic execution for improving automated software testing. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 620–631, May 2015.

[85] Jane Yen, Tamás Lévai, Qinyuan Ye, Xiang Ren, Ramesh Govindan, and Barath Raghavan. Semi-automated protocol disambiguation and code generation. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21. Association for Computing Machinery.

[86] Pengcheng Yin, Chunting Zhou, Junxian He, and Graham Neubig. StructVAE: Tree-structured latent variable models for semi-supervised semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

[87] Chuan Yu and Shuhui Chen. On effects of mobility management signalling based dos attacks against lte terminals. In *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC)*, pages 1–8, 2019.

[88] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018.

[89] Luke Zettlemoyer and Michael Collins. Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, Prague, Czech Republic. Association for Computational Linguistics.

[90] Luke S. Zettlemoyer and Michael Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, Virginia, USA. AUAI Press.

[91] Juan Zhai, Yu Shi, Minxue Pan, Guian Zhou, Yongxiang Liu, Chunrong Fang, Shiqing Ma, Lin Tan, and Xiangyu Zhang. C2s: Translating natural language comments to formal program specifications. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020. Association for Computing Machinery.

[92] Shiyu Zhang, Juan Zhai, Lei Bu, Mingsong Chen, Linzhang Wang, and Xuandong Li. Automated generation of ltl specifications for smart home iot using natural language. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, March 2020.

[93] Yu Zhang, Houquan Zhou, and Zhenghua Li. Fast and accurate neural crf constituency parsing. *arXiv preprint arXiv:2008.03736*, 2020.

[94] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.

A Evaluation Metric To Compare FSMs

For a paragraph in the specification, we have a set of ground truth transitions $\mathsf{S_{GT}}$ and a set of inference transitions $\mathsf{S_I}$. The Boolean expression present as the condition can be complex, and inferring quantitative accuracy from it is non-trivial. Thus, to make the conditions in $\mathsf{S_{GT}}$ and $\mathsf{S_I}$ consistent, we convert them to disjunctive normal form (DNF). Moreover, in this DNF form, if there is a disjunction, we split it into multiple conditions, leading to multiple transitions with the same actions. This further ensures that the conditions from $\mathsf{S_{GT}}$ and $\mathsf{S_I}$ are mapped consistently. We also append the start state to conditions and the end state to actions to take them into account while computing the transition accuracy. After these conversions, we have revised sets of transitions $\mathsf{S'_{GT}}$ and $\mathsf{S'_I}$. For each transition $T_{GT} \in \mathsf{S'_{GT}}$, we compare it with each transition $T_I \in \mathsf{S'_I}$. We then obtain the following match scores:

Conditions. The modified conditions in $T_{GT} \in \mathsf{S'_{GT}}$ and $T_I \in \mathsf{S'_I}$ only have terms connected with conjunctions. We count the number of terms in the ground truth condition c_{GT} of T_{GT} accurately present in condition c_I of T_I. We finally divide this count with the number of terms present in c_{GT} to obtain a *match score* for each transition T_I. Thus, the match score is $\frac{|m_{GT,I}|}{|c_{GT}|}$, where $|m_{GT,I}|$ is the number of matched terms, and $|c_{GT}|$ is the number of terms in c_{GT}. We finally compute the average of the highest match scores for each $T_I \in \mathsf{S'_I}$ for all $T_{GT} \in \mathsf{S'_{GT}}$ and report as the condition accuracy of $\mathsf{S_I}$. The computation is discussed in more detail with an example in the full version of this paper [44].

Actions. For each action $a_{GT} \in T_{GT}$, we consider the inference transitions T_I where $a_{GT} \in T_I$. We again compare the condition expressions for all the obtained T_I, and report the average of the highest match scores (similarly computed as the condition accuracy above) as the action accuracy of $\mathsf{S_I}$. An example of this computation is provided in the full version of this paper [44].