# Spam Companion - working title

*Ivan Trendafilov*


4th Year Project Report

Computer Science

School of Informatics

University of Edinburgh

2012

# Abstract

A really good abstract.

# Acknowledgements

I would like to thank my supervisor, Charles Sutton, for his advice and support throughout this project. His suggestions and feedback were invaluable to building the system to its current state.

# Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Ivan Trendafilov*)

HI MUM.

# Table of Contents

# Chapter 1

# Introduction

The document structure should include:

- The title page in the format used above.
- An optional acknowledgements page.
- The table of contents.
- The report text divided into chapters as appropriate.
- The bibliography.

Commands for generating the title page appear in the skeleton file and are self explanatory. The file also includes commands to choose your report type (project report, thesis or dissertation) and degree. These will be placed in the appropriate place in the title page.

The default behaviour of the class is to produce documents typeset in 12 point, and appropriate for doubled sided printing (all new chapters appearing on the first clear right-hand page). Regardless of the formatting system you use, it is recommended that you submit your thesis printed (or copied) double sided.

**NB** please note that the report should be printed single-spaced. Previously advertised policy of printing in double space has changed as of November 24th 1999 and is no longer valid. Space recommendations are revised as follows: the dissertation should be around 40 sides in single space printing. The page limit is 60 sides in single space printing. Appendices are in addition to the above and you should place detail here which may be too much or not strictly necessary when reading the relevant section.

## 1.1 Using Sections

Divide your chapters in sub-parts as appropriate.

## 1.2 Citations

Note that citations (like [**?**] or [**?**]) can be generated using `bibtex` or by creating `thebibliography` environment. This makes sure that the table of contents includes an entry for the bibliography. Of course you may use any other method as well.

## 1.3 Class Options

The only class option available is `parskip`. It alters the paragraph formatting so that each paragraph is separated by a vertical space, and there is no indentation at the start of each paragraph. This option is used in the current document. See `documentclass` in the skeleton file for usage.

## 1.4 Restrictions

The class does not allow the use of `listoffigures` or `listoftables`.

# Chapter 2

# Background

This chapter details the functionality and implementation of the response generation component. As emphasized in sections [SECTION], [SECTION], response generation depends on the outputs of the information extraction, classification, and identity generation components.

# Chapter 3

# Architecture overview

This chapter details the functionality and implementation of the response generation component. As emphasized in sections [SECTION], [SECTION], response generation depends on the outputs of the information extraction, classification, and identity generation components.

# Chapter 4

# Collector

In this chapter we outline our system's mechanism for collecting new messages. We first set up the context and motivation of the problem. Then, we explore the implementation of our solution. Lastly, we describe three specific modules which implement the collection interface.

## 4.1 Motivation

The collector component provides the entry point for messages into the system. There are two parts to this problem. First of all, the component provides the required functionality to retrieve replies from all current conversation threads. Second, it also provides a mechanism to obtain new instances of the AFF scam and pass it to the conversational agent for further procesisng. Both mechanism ensure full automation, so no user interaction is required for system operation.

The solution to the first problem is trivial – it is briefly described in [SECTION]. In contrast, the problem of obtaining new AFF email messages is more unusual. One common approach is to plant a set of target email addresses in online guestbooks and forums of web sites with high PageRank score. These pages are frequently crawled by email harvester bots. The extracted email addresses are then added to spammers' mailing lists and eventually start receiving spam. Whilst this approach ensures a constant flow of new spam messages, it suffers from two major drawbacks. First of all, it requires manual entry, which is against our principle of full automation. Second, it attracts all types of spam, not advance fee fraud scams specifically. Instead, it is more effective for us to build our own crawler and scraper for AFF scams, as discussed in [SECTION].

## 4.2 Implementation

The implementation of the collector component can be described as a variation of the classical producer-consumer pattern, for $N$ producers and a single consumer, using a FIFO queue. The consumer module defines a uniform, thread-safe interface for accepting messages. Similarly, a set of producer modules define methods for obtaining new messages from various sources. The set of producers run concurrently and implement the consumer interface.

Communication between the consumer and the producers is accomplished through the file system. When a producer starts to fetch a message, it creates a unique temporary buffer on the file system. Once the message is fully downloaded, its contents are flushed to the buffer. Finally, the file is renamed with an extension, which signals it is ready to be consumed. This is safe, because the rename operation is atomic on POSIX compliant operating systems. Concurrently, the consumer spinwaits for new incoming files. If multiple new files are available, it processes them in FIFO order. This approach allows the implementation to be extended easily with new producer modules, without having to change the consumer or any of the existing producers.

[FIGURE of collectors]

## 4.3 Producer modules

For our prototype system, we have implemented three producer modules. These fall into two categories: collectors of new scam instances and collectors of replies to existing threads.

### 4.3.1 419eater.com crawler

The 419eater.com crawler is an example of a producer which collects new scam instances from the aforementioned web site. 419eater.com is a popular online scam baiting community. Its members use a message board to exchange various information – scammer names, techniques, etc. Most notably, the message board features a section titled: “Surplus 419 scam letters” which receives between 20–25 new AFF scam posts per day. This section is a good source of new scam instances.

The producer crawls the section to build an index of available posts. It then crawls each post and scrapes the message content by matching tags in the HTML parse tree. This is illustrated in the algorithm in [FIGURE].

$index \leftarrow$ index of all posts
$visited \leftarrow$ index of crawled posts
$messages \leftarrow \emptyset$
**for all** $post \in index \setminus visited$ **do**
    $tree \leftarrow$ HTML parse tree
    **for all** $node \in tree$ **do**
        **if** $\exists text \in node$ **then**
            $messages \leftarrow text \cup messages$
        **end if**
    **end for**
    $visited \leftarrow post \cup visited$
**end for**

### 4.3.2 Forwarded messages collector

The forwarded messages collector was a commonly requested feature in project meetings and is another example of a producer designed to gather new scam instances. Users can configure their email client to forward the contents of their spam folder to an email account monitored by this provider. The collector will then automatically fetch all messages, strip any personally identifiable information and deliver them to the agent for further processing. Depending on the outcome of the classification task, the agent may initiate a series of new conversation threads.

### 4.3.3 Replies collector

The replies collector interfaces with the identity generation component to fetch new messages from all email accounts maintained by the agent. This is a very simple proces – the collector obtains the current list of email accounts, iterates over the list and downloads any new messages from the associated POP3 server and account. Notably, the collector also observes a limit to the number of consecutive connections allowed to the same POP3 server. This is currently set conservatively to one connection within a 15-minute period, in order to avoid triggering any potential anti-abuse measures.

# Chapter 5

# Information extraction

## 5.1 Overview

The goal of the information extraction component is to obtain useful information from incoming messages. The main tasks performed by this component are named-entity recognition, email extraction, HTML removal, header parsing, quoted text removal and relationship disambiguation. Because this component is the first processing step after entry into the system, it is equipped with methods to deal with potentially dirty data [SECTION]. Once relevant information is extracted, the result is passed as input to the response generation component and is used to generate a convicing, human-like response. For example, the outputs of the named-entity recognition (NER) task might be used to compose a personalized greeting in the beginning of the message – e.g., "*Hello John*".

## 5.2 Implementation

In order to extract the information we are interested in, we apply a series of transformations to each incoming message. These steps are summarized in the algorithm below:

1. remove HTML
2. remove any quoted messages
3. cleanse headers
4. extract *From, Reply-To, Subject, To* headers
5. extract message body
6. perform NER on the message body
7. extract all emails from the message body

8. compute relationships between emails and named-entities

The implementation of steps1–4 and how we deal with dirty data is discussed in [SECTION]. Similarly, in [SECTION] we outline our approach to named-entity recognition and the computation of the named-entity, email pairs.

### 5.2.1 Dealing with dirty data

As we explained in [CHAPTER], a web crawler is one of the main sources of new email messages for our system. Whilst this aids automation, it also creates a number of problems. First, because messages are posted by human forum participants, they rarely conform to MIME or any other RFC memoranda. The lack of standartization makes it impossible to rely on a standard, compliant parsers to extract the headers and payload of our messages. Under MIME, the body part of a message is separated from the headers via a blank line [REF]. This is often omitted in human posts. Furthermore, email clients and servers attach custom headers to processed email messages, so there is no standard set of headers we can match against. Finally, some of the messages downloaded by the crawler are not emails at all, so we also need a way to discard them.

We observe that correct headers follow a set pattern: *"Header: content"*. Furthermore, there is at most one header per line and headers are placed on consecutive lines in the message. Using that observation, we propose a best-effort algorithm to address the problem above. We use a vocabulary $V$ of 144 verified headers (compiled ahead of time). Then, for each incoming message $M$, we use a regular expression to match the contents of the message against the standard set of headers $S$, such that $S = \{Subject, To, From, Reply\text{-}To\}$. If we fail to match at least three elements of $S$, we discard the message, as it is not a valid email. If we find a match, we then iterate over all lines in the message to find $L_{vmax}$, defined to be the last line that contains a header $\in V$. $L_{vmax}$ allows us to split the message into two segment – headers and body. Lastly, we iterate over all lines in the headers segment and match each line against our header heuristic. If there is a match, we augment $V$, which gradually improves the performance of the algorithm.

Another problem with parsing incoming email messages is HTML. Popular web-based email services regularly use HTML to format outgoing messages. As these messages are multi-part, some are encoded in both *text/plain* and *text/html*, whilst others only offer *text/html*. For messages available only in HTML format, we construct

a HTML parse tree and remove all nodes that do not contain text. Finally, we strip all tags and convert any ampersand character codes to ASCII, where possible – e.g., *&* to *&*.

### 5.2.2 Named-entity recognition and emails

# Chapter 6

# Classification

## 6.1 Overview

The classification component serves two purposes. First, it provides a Python wrapper interface to an instance of the Stanford Classifier [REF] – a Java implementation of a Maximum Entropy classifier. Second, it provides two trained models to solve text categorization problems. The first model – the scam variation classifier – helps us place an incoming message in one of 22 recognized classes of advance fee fraud scams. The second model – the PQ classifier – is a binary classifier that recognizes instances of personal questions in the body of a message. Both classifiers are trained on a corpus of AFF messages we have constructed specifically for this project. Their outputs are used to generate relevant responses, as shown in [CHAPTER].

## 6.2 The 419 corpus

In this section we introduce the 419 corpus – a collection of $32,000$ advance fee fraud email messages in 22 classes. We have constructed this corpus from freely available data on antifraudintl.org over the course of a week using a crawler, a scraper and a set of preprocessing techniques. It takes up 134 MB of disk space uncompressed and the data span a period of 5 years – between January 2007 and January 2012. Each message contains the following information:

- A minimal set of headers – *Subject*, *From* and/or *Reply-To* address.
- An extended set of headers – these vary, but may contain *Return-Path*, *Received*, *Date*, *X-Originating-IP*, etc.

- A message body – the contents of the email message
- Scam class – e.g. *Lottery*, *Romance*, etc. See table [TABLE].

The original data has been preprocessed heavily to construct the corpus. Using the information extraction techniques in [CHAPTER], we have discarded over $11,000$ malformed messages – i.e., messages that do not contain the minimal set of headers, a message body or are not emails. This represents 25.6% of the original data. In addition, we have grouped similar categories together (e.g., *Romance scam* and *Russian romance scam*) and reduced the number of possible classes from 31 to 22. Because the data has been labelled by third parties – the moderators and users of antifraudintl.org, we have randomly sampled 380 messages to measure the labelling error rate. We define this rate as the percentage of email messages assigned a wrong class over the total number of messages in the sample. With 95% confidence, we estimate a 0.79% labelling error rate in the data. Whilst this is not insignificant, given the total size of the corpus, it is unlikely that classification accuracy will be significantly degraded due to the error rate.

32000 922 Romance 263179 870 ATM Card 546786 638 Western Union & Moneygram 204352 628 Misc 192875 603 Widow 555 Church and Charity 5473 Next of kin 454 Military 444 Refugees 4347 Lottery 405 Delivery company 389 Mystery shopper 382 Employment 3659 Government 276 Commodities 2615 Business 1779 Banking 1742 Compensation 1625 Dying people 1608 Fake cheques 1582 Loans 1104 Orphans

## 6.3 Methodology

In this section we briefly describe the maximum entropy model. We then outline our feature selection and methodology for training the classifiers.

### 6.3.1 Maximum Entropy model

The Maximum Entropy model is a statistical classification model which seeks to optimize for the probability distribution with the maximum entropy, subject to the constraints of the training set . It is a an exponential model of the following form:

$$p(a|b) = \frac{\prod_{j=1}^{k} \alpha_j^{f_i(a,b)}}{\sum_a \prod_{j=1}^{k} \alpha_j^{f_j(a,b)}} \tag{6.1}$$

where $p(a|b)$ denotes the probability of predicting $a$, given $b$, $\alpha_j$ is the estimated weight of feature $f_i$, and $\sum_a \prod_{j=1}^{k} \alpha_j^{f_j(a,b)}$ is the normalization factor.

By combining evidence into a bag of features, the Maximum Entropy model allows us to represent knowledge about a problem in the form of contextual predicates. Also, in contrast to Naïve Bayes, another commonly used classifier for text categorization [REFERENCE Mitchell, 1997)], the Maximum Entropy model does not assume conditional independence between observed evidence. This allows us to incorporate overlapping features, such as whole words and corresponding n-grams, and typically achieve better probability estimates than possible with Naïve Bayes. [REF]

### 6.3.2 Feature selection

Optimal feature selection for supervised learning problems is a challenging task. Exhaustive enumeration of all feature sets is computationally infeasible, so popular approaches often rely on greedy algorithms such as forward selection and backwards elimination. Nevertheless, building the best possible text categorization model is not the main goal of this project, so we have opted to use n-gram based features, which have shown satisfactory performance in the literature [REF][REF]. The tables [FIGURE], [FIGURE] illustrate the feature selection for the scam variation classifier and the PQ classifier, respectively.

Scam variation classifier - N(-)Grams (length 1–4) - Prefix and Suffix NGrams - lowercase words - lowercase ngrams PQ classifier: - lowercase words - use ngrams - yes - no prefix or suffix ngrams

### 6.3.3 Training the classifiers

We have trained both classifiers on data from the 419 corpus. The scam variation classifier is trained on a 80

To measure the accuracy of our predictive models, we do cross-validation using a 80:20 split. This translates to 6,400 and 100 messages in the validation sets for the scam variation and PQ classifiers, respectively. The results are available in [CHAPTER, SECTION].

# Chapter 7

# Response generation

This chapter details the functionality and implementation of the response generation component. As emphasized in sections [SECTION], [SECTION], response generation depends on the outputs of the information extraction, classification, and identity generation components.

We begin by introducing a bucketing algorithm for grouping messages into conversation threads. Next, we discuss strategies for generating engaging replies. In section [SECTION], we go into detail on how responses are generated and provide examples. Finally, in [SECTION], we outline briefly the mechanism to send out emails anonymously.

## 7.1 Bucketing algorithm

The first step in generating a response to an incoming message is to determine whether the message belongs to a conversation thread. We define a thread as a logical unit of all messages related to a single instance of a scam. Furthermore, it is important to note that a single instance may involve multiple actors. Formally, given an incoming message $M$ and a set of threads $T = \{T_1, T_2, ..., T_n\}$, determine if $\exists M \in T$.

This is a challenging problem. A naïve approach assumes that each conversation thread can contain at most two actors – the agent and the scammer. Therefore, keeping track of the From header is sufficient to maintain conversation state. Whilst this is true in some instances, we showed in [SECTION, Background] that many scams involve multiple actors. Another approach is to compute the thread with the largest word overlap for each incoming message. This is a better solution, but it makes the assumption that the message body always contains a quoted response. Due to the nature of AFF

scams, this is also ineffective.

The bucketing algorithm combines ideas from the aforementioned approaches with one important observation – the scammer always notifies the target in advance when he is being transferred to a third party – e.g. "*please contact the prize remittance manager, John Smith at john.smith@domain.com*". Therefore, an effective way to keep track of threads is to keep track of the mentioned emails in each message. In order to do this, we attach a bucket to each thread and collect email addresses. Once a new message comes in, we try to match it to a bucket. If successful, we update the bucket with any new email addresses. Otherwise, we create a new thread. This is illustrated by the pseudocode in [FIGURE].

[FIGURE - pseudocode]

Let us assume we have threads $T_1, T_2, T_3$ with corresponding buckets $B_1, B_2, B_3$, such that $B_1 = \{E_1, E_2\}$, $B_2 = \{E_3\}$, $B_3 = \{E_5\}$ where $E_n$ is an email address. Message $M_1$ enters the system with a candidate bucket $B_c = \{E_3, E_6\}$. Then, for $B_n$ in $B$, $\arg\max |B_n \cup B_c| = \{1\}$. Therefore, we attach $M_1$ to $T_2$ and update $B_2 \coloneqq B_2 \cup B_c$. Message $M_2$ enters the system with a candidate bucket $B_c = \{E_{10}\}$. For $B_n$ in $B$, $\arg\max |B_n \cup B_c| = \{0\}$. Therefore, we create a new thread $T_4$ with $B_4 \coloneqq B_c \cup \emptyset$.

## 7.2 Strategies

The agent employs a set of strategies designed to occupy the scammer for as long as possible. These strategies can be divided into two categories – cooperative and non-cooperative. Cooperative strategies are mainly used to gain the scammer's confidence and are fulfilled by responding positively to requests. Non-cooperative strategies are used throughout each conversation thread and aim to deflect questions, drive the conversation to a different topic, or elicit extra work from the scammer.

### 7.2.1 Cooperative strategies

In our implementation, we use three different cooperative strategies.

The first strategy is to always express interest in any proposed scheme in the beginning of a conversation thread. The initial responses are always enthusiastic and reaffirm the premise set up by the scammer. For example, if the scammer's initial email claims we have won the lottery, we do our best to pretend that we did. This strategy helps create the impression that the agent is a viable target and encourages the

scammer to spend time replying back.

Another cooperative strategy involves responding to requests for personal information. As we discussed in [CHAPTER], scammers commonly request personal information as a way to establish the target's trust. If the target gives out personal details, it is considered much more valuable to the scammer, as it is very likely to cooperate with other future requests. Therefore, a useful strategy for our agent is to respond positively to these requests. If asked, our agent responds to these requests with personal details obtained from the identity generation component. This helps build up the scammer's perception that the agent is a viable target and makes him more invested in the conversation.

The final cooperative strategy is reaffirmation. Reaffirmation is used throughout each conversation thread to restate the agent's interest in the scammer's proposition. It is often used in conjunction with non-cooperative strategies to express that we are still interested in the scheme, despite any setbacks we might have introduced. For example, an example of reaffirmation is claiming – "*I look forward to working with you to process my winning*" at the end of the message, whilst asking many extra questions in the message body.

### 7.2.2 Non-cooperative strategies

We use four main non-cooperative strategies in the implementation of our agent.

The first non-cooperative strategy is deflecting questions. As we discussed in [SECTION], answering certain types of questions is beneficial, as it helps establish trust. However, complying with other types of questions can be dangerous or impossible. One example are requests for photo identification. Cooperating with these is clearly a bad idea. Instead, we choose to compose excuses – e.g. "*I am sorry, but I am bad with computers. Could you tell me how to put a photo in this letter?*" Excuses are usually effective at bypassing these questions altogether. Alternatively, the scammer has to spend the time to write a mini tutorial on how to work with email attachments.

Another non-cooperative strategy is asking questions about exceptional circumstances. As we observe in [SECTION], answering questions is a challenging NLP task. Instead, it is better to ask questions and attempt to drive the conversation. These questions are closely related to the variation of the scam in play and are designed to elicit extra work from the scammer. For example, in the context of a lottery scam, the agent might ask if the winnings are subject to any tax or whether they can be paid out

in Australian dollars.

Stories are the third non-cooperative strategy. They are employed in later stages of a conversation and are context independent. The main purpose of stories is to take up the scammer's time by having him read a large chunk of text, following a few initially promising exchanges. Because of this, each story is between 350 and 600 words long. At the end of a story, the scammer is asked whether he can relate to the situation described in the story. By requesting a comment, we validate that he has read the story.

Lastly, our final non-cooperative strategy is to prompt the scammer to resend messages. This behavior is elicited by composing replies that claim the scammer's previous message has been accidentally deleted, is garbled, or has never been received. This strategy has a single goal – to force the scammer to look through his inbox, find the correct message, and resend it. It is a simple way to get the scammer to do extra work.

## 7.3 Composing a response

Responses are generated from the outputs of the information extraction, identity generation, classification tasks and current conversation state via a multi-layer finite state machine. The process works as follows: the top layer FSM builds a template with placeholders for lower-level probabilistic finite state machines (PFSMs). The lower-level PFSMs emit text or another layer of PFSMs. At the bottom layer, all PFSMs generate text. This approach can also be described as top-down text generation.

### 7.3.1 Understanding the context

Understanding natural language is a very challenging task. Fortunately, advance fee fraud scams tend to follow a set pattern within a relatively narrow domain ([CHAPTER]). This makes it possible for us to use techniques such as machine learning, pattern matching rules and the conversation state to generate convincing replies.

Two maximum entropy classifiers are central to our effort to understand the contents of an incoming message. The the scam type classifier helps determine which one of the 22 known AFF variations is currently in play. For example, if we are dealing with an instance of a lottery scam, we will use the corresponding lottery PFSMs and text snippets to generate that part of the response. The second classifier helps determine whether the scammer asks us to provide any personal information. If so, similarly as before, we will use another set of corresponding PFSMs to generate that information.

Rules are the second method we use to understand the context of a message. Rules are specific to each AFF variation and allow us to look for patterns that are strong signals for conversation state. In the aforementioned lottery scam, we recognize four distinct states – *initial, claim form approved, payment approved, fee request*. Knowing the current state allows us to generate a reply with information specific to that state and creates the impression of a natural response.

Finally, it is not always possible to understand the context of a message through a classifier or rules. Where this is not possible, we use the current state of the thread, as determined by the bucketing algorithm, and compose a reply using a non-cooperative strategy. For example, if the current state shows we are in the beginning of a thread, we will use the asking questions PFSM. In later stages, we will pick randomly between prompting the scammer to resend the message or the story strategy.

### 7.3.2 Text snippets

We use text snippets in the final states of the PFSMs to generate large paragraphs of text. These text snippets are written ahead of time and are stored in a hierarchical tree data structure. Top-level nodes represent specific scenarios – *photo request*, *lottery*, etc. Scenarios which describe scam variations have another set of mid-level nodes which list all recognized states of the variation. Finally, at the bottom of the tree, leaves contain a set of text snippets associated with their parent nodes. Text snippets are diverse, but usually consist of 1–3 sentences, with placeholder variables for conversation-specific information (e.g. the agent's identity). For current system prototype, we have defined 37 nodes and 169 text snippets. Extending the collection of text snippets is easy and requires no changes to the implementation – new snippets can be saved as text files in the file system.

Further to adding context, text snippets also provide variability to the generated responses. This is accomplished through redundant tree leaves. These leaves have similar sentiment, but are phrased in different ways. Let us illustrate this. Consider the example scenario of answering questions for personal information. The response paragraph is constructed from text snippets of the following bottom-level tree nodes: intro, name, age, occupation, location, postcode, contact details, closing. These nodes have 5, 7, 8, 5, 4, 2, 6, 6 leaves respectively. A PFSM crawls the tree and randomly selects a single leaf from each parent node. The outputs are then compiled into the final text paragraph. In this instance, the aforementioned process allows for $8 \times 7 \times$

$6^2 \times 5^2 \times 4 \times 2 = 403,200$ distinct paragraph compositions. As such, the probability of generating a duplicate response is very low. We use this approach to generate each paragraph of the outgoing message.

### 7.3.3 Finite state machines

Several layers of FSMs and PFSMs determine how each response is generated. We start by looking at the top-level FSM. We move on to a simple PFSM used for generating greetings. Finally, we illustrate the PFSMs that generate the content body.

The top-level FSM defines the high-level structure of our message – a letter. The Greeting, Body and Signature states represent a set of PFSMs, whilst Quoted_Text simply produces a quoted version of the incoming message. The FSM is illustrated below.


The greeting generation automata is an example of a simple bottom-level PFSM. Its task is to accept the name of the scammer and produce a greeting, as illustrated in [FIGURE]. The transitions between the has_name state and the text snippets have probability $p = \frac{1}{3}$.

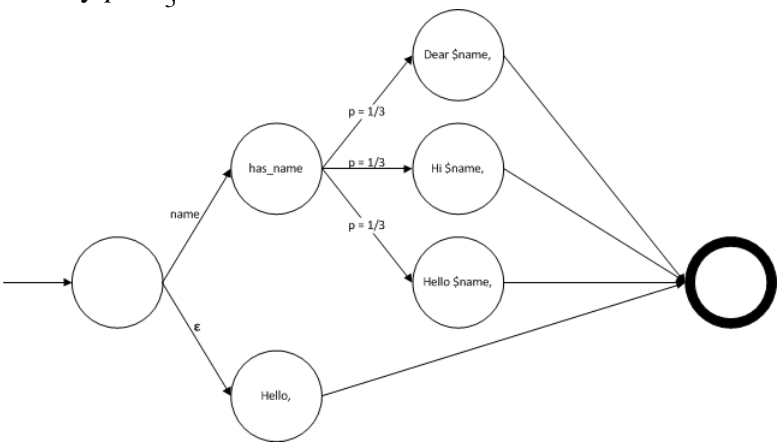
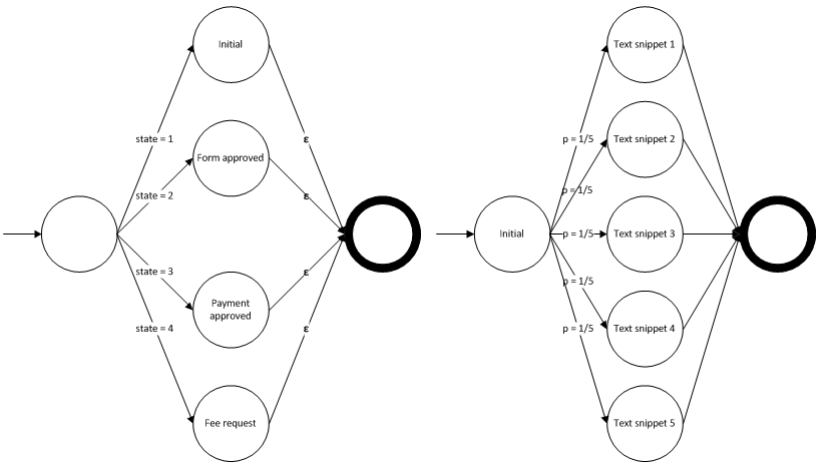



Figure 3 (missing) illustrates the PFSM which generates the content body of a message. It takes in as inputs the results of classifier, information extraction and identity generation tasks. Its main purpose is to select strategies and create placeholders for lower-level PFSMs. Strategies are selected based on the available information and conversation state. Please note, for clarify of the figure, we have omitted some of the notation.

Figure (a) illustrates a lower-level FSM for the lottery scam variation. The scam




(a) Lottery FSM (b) Snippet FSM

is modelled by four distinct states. Figure (b) illustrates a bottom-level PFSM for the same scam, after a transition to the initial state.

We follow the same approach when modelling other scam variations and conversation strategies.

### 7.3.4 An example message

[FIGURE]

### 7.3.5 Sending the reply

As we described in [SECTION], anonymous email accounts are used to send and receive emails. In order to ensure consistency, each thread has an associated identity and email account. Once a message is generated, it is transmitted via a SSL connection to the SMTP server

Through experimentation, we have established that sending 40 messages with less than a second delay between messages results in the termination of the associated email account by the provider. To mitigate this, we observe a random backoff period between consecutive SMTP connections. This is currently set to a minimum of 70 and a maximum of 170 seconds between connection attempts.

# Chapter 8

# Evaluation

This chapter details the functionality and implementation of the response generation component. As emphasized in sections [SECTION], [SECTION], response generation depends on the outputs of the information extraction, classification, and identity generation components.

# Chapter 9

# Conclusions

This chapter details the functionality and implementation of the response generation component. As emphasized in sections [SECTION], [SECTION], response generation depends on the outputs of the information extraction, classification, and identity generation components.