FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Plugin zur Interaktion mit IMAP-Servern durch ACTHEX

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Software & Information Engineering

eingereicht von

Andreas Schmidt
Matrikelnummer 01526918

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: O.Univ.Prof. Dipl.-Ing. Dr.techn. Thomas Eiter
Mitwirkung: Dipl.-Ing. Dr.techn. Peter Schüller

Wien, 20. Jänner 2019

Andreas Schmidt	Thomas Eiter

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Plugin for Interacting with an IMAP Server from ACTHEX

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Software & Information Engineering

by

Andreas Schmidt
Registration Number 01526918

to the Faculty of Informatics

at the TU Wien

Advisor: O.Univ.Prof. Dipl.-Ing. Dr.techn. Thomas Eiter
Assistance: Dipl.-Ing. Dr.techn. Peter Schüller

Vienna, 20th January, 2019

Andreas Schmidt	Thomas Eiter

Erklärung zur Verfassung der Arbeit

Andreas Schmidt
Kasernenstraße 14, 7000 Eisenstadt

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 20. Jänner 2019

Andreas Schmidt

Kurzfassung

Der ACTHEX-Formalismus erweitert Answer Set Programming um die Formalisierung sogenannter *externer Atome* und *Aktionen*. Diese bieten Zugriff auf externe Wissensbasen und die Möglichkeit, Umgebungen durch die Ausführung von Aktionen aktiv zu beeinflussen. In dieser Arbeit wird ein Plugin zur Interaktionen mit IMAP-Servern, zum Zwecke der Integration von Informationen aus Emails in ACTHEX-Programme und der Manipulation von Emails und Mailboxen mithilfe von *Aktionen*, entwickelt. Zu diesem Zweck wird eine Problemstellung formuliert, das entwickelte Plugin dokumentiert und getestet, sowie dessen Anwendung durch Beispiele demonstriert.

Abstract

The ACTHEX-formalism extends answer set programming by introducing so-called *external atoms* and *action atoms* that provide access to external knowledge resources and the possibility of actively influencing environments through the execution of actions. In this work a plugin is created for enabling interaction with IMAP servers in order to integrate information contained in email messages into ACTHEX programs and manipulate messages and mailboxes through actions. For this purpose, a problem statement is formulated, the developed plugin is documented and tested and possible applications are demonstrated through examples.

Contents

CHAPTER 1

INTRODUCTION

1.1 Motivation

Answer Set Programming (ASP) is in the interest of research of various research groups all over the world for over 20 years now. It describes a certain kind of programming languages following the declarative logic programming paradigm, that is to devise a set of logical formulas that specify a problem that should be solved and then let a so-called answer set solver find solutions to it [Lif02].

ASP was developed in order to be able to solve problems in an intuitive way [BET11]. Especially solving computationally complex search problems, like the well-known graph coloring problem, benefits from the advantages and the performance of ASP. Planning and reasoning over knowledge bases are areas where ASP can play to its strengths [Lif02].

Due to its versatile application purposes, the field of ASP has been evolving and gave reasons to make an effort on the development of formalisms that allow the access of external knowledge resources from answer set programs.

One of such formalisms is the HEX language, an extension of ASP that allows the access of external knowledge bases [EFI+16]. In order to further expand the possible application purposes of HEX and to enhance its potential, an extension called ACTHEX was developed with the intention of creating the possibility of letting answer set programs actively influence their environment through the execution of actions [BEFI10].

An implementation of the HEX formalism is DLVHEX [EMRS15, EGI+18], which is a reasoning system for answer set programs with external knowledge bases that allows various ways of integrating external resources like the semantic web or knowledge resources provided by Python or C++ programs. Another HEX solver is HEXLITE, a partial implementation of the HEX language, which is intended to be easy to use and available via several package management systems [Sch18]. HEXLITE implements also parts of the ACTHEX language and provides an interface for Python plugins.

Motivated by the promising capabilities of HEX and ACTHEX and its versatile application purposes, this work is going to continue the efforts of integrating external knowledge resources into ASP by attempting to integrate the information that is contained in email messages and elaborate how ASP can be used in this context. Potential applications are filtering messages and the organization of messages and mailboxes with rules in form of answer set programs.

1.2 Problem statement

Although ASP provides a variety of application purposes, generally ASP languages are restricted in their use, because of their limited ability of accessing external knowledge bases, and have consequently difficulties in prevailing against other more flexible programming paradigms. To this end, HEX and ACTHEX have been developed, attempting to provide an extension for ASP by the ability of accessing external knowledge bases and actively influencing their environment.

The capability of ACTHEX to allow the integration of external knowledge and to influence a program's environment opens a broad variety of possible applications. Yet there has not been made any effort in supporting the access to email messages from ACTHEX, in order manipulate emails and integrate knowledge contained in email messages, even though emails may offer valuable information for different application purposes.

1.3 Aim of this work

The aim of this work is the continuation of ongoing research works on the topic of the extension of ASP towards the integration of external knowledge resources. To that end, this work suggests how ASP can be used for the purpose of processing email messages accessed via IMAP and develops a plugin for ACTHEX, which facilitates the access and the processing of emails through answer set programs. As a result of the thereof gained insights and experience, this work is going to reflect about the potentials of such an extension considering the further improvement of the capabilities of ASP.

1.4 Methodological approach

The methods used in this work are the assessment of the possible advantages, the combination of ASP and email messages could offer, and the elaboration of a way this can be achieved. As a practical approach, a plugin for ACTHEX, that allows the integration of information contained in email messages, is going to be developed. In order to examine the results, subsequently possible applications of answer set programs with support of email messages are going to be demonstrated in example programs and the actual advantages and disadvantages of the developed system are going to be discussed in comparison with other approaches to filter and manage email messages.

1.5 Structure of the work

This work is structured as follows:

Chapter 2 analyzes the state of the art in ASP and gives a formal introduction to ACTHEX and IMAP.

Chapter 3 describes how the functionality of IMAP can be integrated into ACTHEX by stating a selection of the IMAP functionality that should be made accessible from ACTHEX.

Chapter 4 suggests how this selection can be implemented in a plugin for ACTHEX and how possible application scenarios that make use of this plugin can be realized, through the demonstration of example programs.

Chapter 5 reflects about the gained insights and discusses aspects, related to the context of ASP and IMAP that could not be approached as part of this work.

Chapter 6 finally summarizes the results of prior work and how they can be used and extended in future research works.

CHAPTER 2

STATE OF THE ART

2.1 Answer Set Programming

Answer Set Programming (ASP) [BET11, SW18] is a programming paradigm that belongs to the declarative logic programming paradigm and is based on the idea of non-monotonic reasoning. The declarative programming paradigm is - opposed to the imperative paradigm, where a program is a sequence of instructions that is executed by the processor - a form of programming, where a program is the specification of a space of potential solutions and constraints on the solutions of a problem. The search for the actual solutions to this problem is then performed automatically. The logic programming paradigm is a form of declarative programming and based on the idea of using logic as a programming language. Here, a program is a set of logical formulas that represent a problem which should be solved. The first logic programming language was Prolog, which was developed in the 1970s. Other logic programming formalisms, like ASP, emerged from Prolog and also often use a similar syntax to that of Prolog [Kow74] [EIK09].

The idea of ASP goes back to Gelfond and Lifschitz who introduced its semantics (called the stable model semantics) in 1988 [GL88]. A variety of languages based on the ASP paradigm were developed in the past for various different application purposes and uses [EIK09].

2.2 ACTHEX

One of those ASP languages is HEX, a language which was introduced by Eiter et al. 2005 [EIST05, EFI$^+$16], extending ASP by a formalism to access external knowledge bases. HEX defines so-called external atoms in addition to the ASP language introduced by Gelfond and Lifschitz. The ASP language ACTHEX, in turn, extends HEX by the ability of actively influencing the environment of an ACTHEX program by executing actions. The execution of actions is determined by presence of certain atoms in a program's answer set. To that end, ACTHEX defines so-called action atoms.

Example 1. The following is a valid ACTHEX program [BEFI10]:
$evening \lor morning.$
$\#robot[turnAlarm, on]\{c, 2\} \leftarrow evening.$
$\#robot[turnAlarm, off]\{c, 2\} \leftarrow morning.$
$\#robot[move, all]\{b, 1\} \leftarrow \&getFuel[](high).$
$\#robot[move, left]\{b, 1\} \leftarrow \&getFuel[](low).$ ◇

In the following, the syntax and semantics of ACTHEX is described:

2.2.1 Syntax

Definition 1. *An* ACTHEX *program P is a finite set of rules of the form*

$$\alpha_1 \lor ... \lor \alpha_k \leftarrow \beta_{k+1}, ..., \beta_m, not\beta_{m+1}, ..., not\beta_n$$

with $k \geq 0$, $n \geq m \geq 0$, atoms or action atoms $\alpha_1, ..., \alpha_k$ and atoms or external atoms $\beta_{k+1}, ..., \beta_n$. For each rule r, $H(r) = \{\alpha_1, ..., \alpha_k\}$, $B^+(r) = \{\beta_{k+1}, ..., \beta_m\}$ and $B^-(r) = \{not\beta_{m+1}, ..., not\beta_n\}$ are defined.

Definition 2. *An external atom is an atom of the form*

$$\&g[Y_1, ..., Y_n](X_1, ..., X_m)$$

with terms $Y_1, ..., Y_n, X_1, ..., X_m$ and an external predicate name $\&g$ with fixed lengths $in(\&g) = n$ and $out(\&g) = m$ [EIST05].

Definition 3. *An action atom is an atom of the form*

$$\#g[Y_1, ..., Y_n]\{o, r\}[w : l]$$

with terms $Y_1, ..., Y_n$, an action predicate name $\#g$ with fixed length $in(\#g) = n$, action option $o \in \{b, c, c_p\}$ and positive integers or variables r (action precedence), w (action weight) and l action level [BEFI10].

The following *safety condition* must hold: every variable in a rule r must also appear in some atom in $B^+(r)$.

2.2.2 Semantics

The *Herbrand base* HB_P of an answer set program P is the set of all ground atoms and external atoms that can be obtained by replacing the variables in the predicates of P by the ground terms that can be formed from the constants and functions occurring in P. Accordingly, the grounding $grnd(r)$ of a rule r is the set of all rules that can be obtained by replacing the variables of r by the ground terms that can be formed from the constants and functions occurring in P. The grounding of P is $grnd(P) = \bigcup_{r \in P} grnd(r)$.

Definition 4. *An interpretation I of an* ACTHEX *program P is a subset of HB_P.*

An ACTHEX program P acts in an *external environment* E, which is assumed to be a collection of data structures.

For each external predicate name $\&g$, there is a function $f_{\&g}$ that assigns each tuple $(I, y_1, ..., y_n, x_1, ..., x_m)$, with $I \subseteq HB_P$, $in(\&g) = n$, $out(\&g) = m$ and constants $y_1, ..., y_n, x_1, ..., x_m$, either 0 or 1.

For each action predicate name $\#g$, there is a function $f_{\#g}$ that assigns each tuple $(E, I, y_1, ..., y_n)$, with *external environment* E, $I \subseteq HB_P$, $in(\#g) = n$ and constants $y_1, ..., y_n$, a new *external environment* E'.

An interpretation I is called a model of an ACTHEX program P, iff $I \models r$ for every ground rule $r \in grnd(P)$.

For an interpretation I and a ground rule r, $I \models r$, iff $I \models H(r)$ if $I \models B(r)$.

For an interpretation I, $I \models H(r)$, iff there is an $a \in H(r)$ such that I is a model of a.

For an interpretation I, $I \models B(r)$, iff for every $a \in B^+(r)$, $I \models a$, and for every $a \in B^-(r)$, $I \not\models a$.

An interpretation I is called a model of an atom $a \in HB_P$, denoted $I \models a$, iff $a \in I$.

Definition 5. *The FLP-reduct fP^I of an* ACTHEX *program P with respect to an interpretation I is the set of all $r \in grnd(P)$, such that $I \models B(r)$.*

I is called an answer set of P iff no answer set I' of P exists, with $I' \subset I$ [FLP04].

Intuitively, functions that are associated with external atoms and action atoms model access to external knowledge bases, respectively model the execution of the action that influence a program's environment [BEFI10]. For existing ACTHEX solvers, such as HEXLITE, the implementation of these functions are provided in a plugin that also defines the external atoms and action atoms themselves. ACTHEX has first been described in [BEFI10] and the first implementation and certain formal aspects have been made more concrete in [FGI+13].

2.2.3 Iterative evaluation of ACTHEX programs

A basic concept of ACTHEX is to iteratively evaluate an ACTHEX program multiple times in a loop and execute actions after each iteration based on the evaluated answer sets. This process is repeated until a certain action atom, indicating the end of the loop, is executed or until no answer set is found. This behaviour relates to the idea of influencing a program's environment by the execution of actions, in that external atoms integrate knowledge from the environment into the program and actions modify the environment, leading to different knowledge integrated by the external atoms after the execution of actions. This idea can be used in various applications where an answer set program should be used to perform actions itself and also reason from the modified environment after the execution of actions. A formal definition of the semantics and the iterative evaluation of ACTHEX programs is given by Eiter et al. 2012 [EFF12].

2.3 IMAP

The Internet Message Access Protocol (IMAP) specifies rules for communication between a server and a client for the purpose of enabling the client to access email messages stored on the server's computer. The messages are organized in mailboxes that can be viewed as folders, like the folders in a file system, and can also be structured hierarchically, meaning that a mailbox can contain other mailboxes. Messages are identified by *Message IDs*, which are unique within their mailbox. A message on an IMAP server is therefore identified by its *Message ID* and the mailbox path it is contained in. Usually, server and client are connected through the internet and *TCP/IP* is used as the underlying network protocol. An IMAP server stores the messages and mailboxes of its users and the users access them via IMAP. As opposed to the *Post Office Protocol (POP)*, the messages stay on the server after they got accessed by a client and can therefore be accessed again later (e.g. from another IMAP client). As a consequence, messages can be fetched from the server, without changing the server's state, but the manipulation of messages or mailboxes is persistent for all clients.

IMAP specifies messages that the client and the server are allowed to send to each other, as well as their intended behaviour. The messages sent from the client to the server are commands that either query information, the client wants to acquire from the server or instruct the server to perform certain manipulations on email messages or mailboxes. The messages sent from the server to the client are usually answers to the client's commands.

For security reasons, IMAP connections are often established based on encrypted *Secure Sockets Layer (SSL)* connections. Nowadays, most commercial email providers only allow encrypted connections to their IMAP servers.

IMAP was originally introduced in the 1980s by Mark Crispin to replace *POP* [MM00]. The latest version, on which the implementation of the IMAP plugin for ACTHEX is based, is IMAP Version 4, published in 2003 [Cri03]. Today, IMAP is the state of the art protocol for communication between mail clients and mail servers, which is the reason for its application in this work.

2.4 Message Filtering

At the moment, there are different methods to manipulate email messages automatically. A common application is the filtering of messages, for example to sort out spam mail or to classify messages based on predefined criteria and to move them in particular mailboxes.

While formerly, separate programs (so called *Mailfilters* or *Milters*) were used for message filtering (e.g. *procmail* or *Apache SpamAssassin*), today, filtering is often done by

Mail Delivery Agents themselves, which deliver messages to a user's mail account. In general, message filtering is mostly applied server-sided, although software like *Apache SpamAssassin* can also be used on the client side (e.g. with *Mozilla Thunderbird*).

Example 2. This example illustrates how *procmail* can be instructed to delete all messages from *unwanted@example.com*. The following code has to be written into a configuration file of *procmail*:

```
:0
* From: unwanted@example.com
/dev/null
```

◇

Present systems for message filtering commonly use the message filtering language *Sieve*, which was developed in 2001 and allows users to define their own filter rules. However, *Sieve* is not very expressive and only allows rather primitive filter rules [GS08]. This motivates the approach of developing a plugin for ACTHEX that allows for managing email messages and mailboxes in a more sophisticated way.

Example 3. This example shows how to move messages from *user1@example.com* into another mailbox using *Sieve*:

```
if address :is ["From", "To"] "user1@example.com" {
   fileinto "INBOX.mails-from-user1";
}
else {
  keep;
}
```

◇

CHAPTER 3

DESIGN OF THE ACTHEX IMAP INTERFACE

This chapter describes the design of the plugin for ACTHEX that facilitates the interaction with IMAP. The plugin consists of a set of external atoms and action atoms that provide certain parts of the functionality of IMAP. In the following a suggestion of IMAP functionalities that shall be accessible from ASP is given, followed by a selection of IMAP commands that cover these functionalities. Subsequently, considerations of how these IMAP commands can be mapped onto ACTHEX using external atoms and action atoms are defined.

3.1 Selection of IMAP commands

IMAP provides functionality to access email accounts, to access and manipulate mailboxes and to query and manipulate email messages. However, the entire functionality provided by IMAP also contains additional functions that exceed what is needed in order to allow reasoning over email messages in ASP and interaction between answer set programs and IMAP servers to access and manipulate mailboxes and messages. Since the IMAP plugin for ACTHEX is created for this purpose, the functionality of IMAP is required only partially. Therefore, a selection of IMAP commands whose functionality should be covered by this plugin has to be made. Such a selection has to be a compact set of functionalities that meet the requirements stated above in order be useful for future applications.

In order to start interaction with an IMAP server, a client needs to send a *LOGIN* command to the server. *LOGIN* takes the mail account's user name and password as arguments. Accordingly, an action atom that takes a hostname, a port, a user name and a password as an input list is needed. In order to enhance the capabilities of the

IMAP plugin, the possibility of establishing an encrypted or non-encrypted connection should be supported. Thus, an additional flag specifying whether the connection should be encrypted or not has to be in the input list.

One main requirement for the IMAP plugin is the integration of information that is accessible via IMAP and contained in emails in a format that allows to reason new knowledge from the accessed information in ACTHEX programs. IMAP specifies the commands *LIST*, *SEARCH* and *FETCH*:

LIST takes two filter arguments and commands the server to return a list of mailbox names that match the given filters. The first argument specifies the mailbox in which to search for mailboxes, the second argument the mailbox name with wildcards.

FETCH returns a message's content.

SEARCH takes a filter criterion and commands the server to return a list of message IDs that match the given filters. In order to be able to apply *SEARCH*, a mailbox has to be selected before, using the *SELECT* command. The words that can be used as filter criteria for the *SEARCH* command are defined in *Augmented Backus-Naur Form* in Figure 3.1. Using the logical operators *or*, *not* and conjunction, which is expressed by multiple filter criteria separated by empty spaces, also more complex filters can be expressed. A detailed description of the syntax and semantics is given in "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1" [Cri03].

Besides the integration of knowledge provided by IMAP into answer set programs, the IMAP plugin should be able to perform manipulations on email messages and mailboxes via IMAP. Commands for this purpose are *STORE*, *COPY*, *APPEND*, *EXPUNGE*, *CREATE*, *DELETE* and *RENAME*. *STORE* commands the IMAP server to set or unset one of the flags "\Seen", "\Answered", "\Flagged", "\Deleted", "\Draft" or "\Recent" or any other flag accepted by the server for a given message. *COPY* commands the IMAP server to copy a given message to another mailbox. *APPEND* commands the IMAP server to add a specified message into a given mailbox. *EXPUNGE* commands the IMAP server to delete all message in a given mailbox that have a "\Deleted" flag. *CREATE* commands the IMAP server to create a specified mailbox. *DELETE* commands the IMAP server to delete a specified mailbox. *RENAME* commands the IMAP server to rename a specified mailbox.

3.2 Selection of External Atoms and Action Atoms

The specified atoms are intended to map those parts of the functionality of IMAP onto ACTHEX that appear to be relevant in the context of ASP. The selection of external atoms and action atoms is a minimal set that comprises all of the IMAP functionalities that seems to be useful when reasoning over emails and is made with respect to the existing IMAP commands and their intended usage. For a more detailed specification of the external atoms and action atoms see Chapter 4.

```
ListOfFilters  =  Filter " " ListOfFilters

Filter         =  ("(" Filter ")")
                  | ID
                  | "all"

                  | "answered" | "deleted" | "draft"
                  | "flagged" | "new" | "old" | "recent"
                  | "seen" | "unseen" | "unanswered"
                  | "undeleted" | "undraft" | "unflagged"
                  | ("keyword" Flag) | (unkeyword" Flag)

                  | ("on" Date) | ("sentbefore" Date)
                  | ("senton" Date) | ("sentsince" Date)
                  | ("since" Date)

                  | ("bcc" String) | ("body" String)
                  | ("from" String) | ("header" Field String)
                  | ("subject" String) | ("text" String)
                  | ("to" String)

                  | ("larger" N) | ("smaller" N)

                  | ("not" ListOfFilters)
                  | ("or" ListOfFilters ListOfFilters)

ID             =  1⋆DIGIT

String         =  "′" 1⋆VCHAR "′"

Date           =  1⋆2DIGIT "-" ("Jan" | "Feb" | "Mar" | "Apr"
                  | "May" | "Jun" | "Jul" | "Aug" | "Sep"
                  | "Oct" | "Nov" | "Dec") "-" 1⋆4DIGIT

Field          =  1⋆VCHAR

Flag           =  1⋆VCHAR

N              =  1⋆DIGIT
```

Figure 3.1: Syntax of possible filter criteria for the *SEARCH* command in *Augmented Backus-Naur Form.*

The IMAP commands mentioned above are considered to provide a large and useful portion of the relevant functionality of IMAP and will thus be part of the IMAP plugin for ACTHEX. The following selection of external atoms and action atoms is a minimal set that comprises all of these functionalities with respect to the existing IMAP commands and their intended usage and the characteristics of ASP. Therefore, some considerations are required for how the selected IMAP commands can best be mapped onto external atoms and action atoms. Some commands can be used as external atoms and action atoms as they are, others need to be restructured or combined to one atom.

Because the IMAP plugin should be able to handle multiple connections to different IMAP servers at a time, a term that identifies for which connection an external atom should be evaluated or for which connection an action should be performed is needed. This term will be denoted as *ConnectionID* in the following.

The establishment of a connection to an IMAP server consists of building up a connection to the server via TCP/IP and sending a *LOGIN* command. Due to the fact that these steps are always being performed together and in this order, they can be combined to one action atom. An appropriate action atom is

- $\#login[\mathit{ConnectionID},\ Hostname,\ Port, Username,\ Password, \mathit{SSL}]$.

Here, *ConnectionID* has to be provided as a term of the *#login*-action in the answer set program. This may seem unintuitive at first glance, because usually a client gets a connection ID assigned by a server and not the other way round. On closer consideration, however, it appears to be necessary, because action atoms do not allow for an output list of terms (as opposed to external atoms).

In order to be able to reason from the fact whether a connection with a given *ConnectionID* has already been established or not, and to perform actions before or after a connection exists, an appropriate external atom like

- $\&logged\_in[\mathit{ConnectionID}]$

is needed.

The IMAP command *LIST* can be mapped onto the external atom

- $\&mailbox[\mathit{ConnectionID},\ Path, Name](Mailbox)$

that takes the same filter arguments in its input list as the *LIST* command.

As stated above, with *SEARCH*, IMAP provides a quite easy way of filtering and fetching message IDs based on a message's properties. A filter argument with the same syntax as the filter argument of *SEARCH* can be used in ACTHEX programs relatively easy by concatenating single parts of the string that form the filter argument. Accordingly, the external atom

- $\&filter[ConnectionID,\ Mailbox,\ Filter](MessageID)$

where *Filter* denotes a filter argument with the syntax of the filter arguments of *SEARCH* is suitable for integrating message IDs matching specified filters into ACTHEX programs.

Depending on the arguments, the *FETCH* command returns a list of header information of a message or its content. In order to represent a message's header information, like its subject, in a useful way and to be able to reason from messages' header information or their content, we define the external atoms

- $\&msg\_header[ConnectionID,\ Mailbox,\ MessageID](Type,\ Value)$

and

- $\&msg\_body[ConnectionID,\ Mailbox,\ MessageID](Body)$.

To keep these atoms simple and adapt them to the characteristics of answer set programs, *MessageID* denotes a single message ID and not a sequence of message IDs (other than the *FETCH* command's argument). A sequence of messages on which *FETCH* should be applied, is represented by multiple &*msg_body* atoms (respectively &*msg_header* atoms) in ASP.

STORE allows to add, remove or replace a given list of flags for a given sequence of message IDs. Due to the fact that it is hard to work with lists in ASP in general, the configurability of the *STORE* command can be simplified, resulting in the external atom

- $\#set\_flag[ConnectionID,\ Mailbox,\ MessageID,\ Flag]$

which allows to add or remove one flag for one message. To reproduce the IMAP functionality of executing *STORE* for a sequence of message IDs, multiple #*set_flag* actions can be triggered at the same time.

COPY also takes a sequence of message IDs and will also better be mapped onto an external atom that takes only a single message ID. An appropriate action atom is

- $\#copy\_msg[ConnectionID,\ Mailbox,\ MessageID,\ NewMailbox]$.

APPEND takes a message that should be added to the given mailbox as a single string, which contains the components the message consists of, namely headers and text. Though building large strings like this, which are composed of many substrings, is feasible in ASP, it leads to programs with a lot of rules and is therefore unsuitable for ASP. Due to this, it seems better to adapt the action atom corresponding to *APPEND*. The action atom

- $\#create\_msg[ConnectionID,\ Mailbox,\ Sender,\ Subject,\ Recipient,\ CC,\ BCC,\ Text]$

does not consider all possible message headers, but is confined to those which seem to be rather relevant.

The IMAP commands *EXPUNGE*, *CREATE*, *DELETE* and *RENAME* can be mapped onto action atoms mostly as they are and result in

- $\#expunge\_msgs[ConnectionID,\ Mailbox]$,
- $\#create\_mailbox[ConnectionID, Mailbox]$,
- $\#delete[ConnectionID,\ Mailbox]$ and
- $\#rename\_mailbox[ConnectionID, Mailbox,\ NewMailbox]$

respectively.

IMAP does not provide a command for creating a reply to a given message. However, it seems to be useful to have such an action atom as part of the IMAP plugin for ACTHEX. Accordingly, we define the action atom

- $\#reply[ConnectionID,\ Mailbox,\ MessageID,\ Text]$.

Another action that is not covered by an IMAP command, is to move a message from one mailbox to another. Usually this can be achieved by copying and deleting a message. For the IMAP plugin for ACTHEX, although, a seperate *move* action makes sense because it is not always easy to determine the execution of multiple actions in ACTHEX and *move* is a relatively often used action, especially when an ACTHEX program is used for classification of messages and to move them to certain mailboxes. An adequate action atom would be

- $\#move\_msg[ConnectionID,\ Mailbox,\ MessageID,\ NewMailbox]$.

CHAPTER 4

IMPLEMENTATION OF THE IMAP PLUGIN

This chapter specifies the IMAP plugin developed as part of this work and, subsequently, illustrates the use of the plugin in possible application scenarios by some examples. The implementation of the IMAP plugin is available at http://www.kr.tuwien.ac.at/research/systems/dlvhex/imapplugin.html and http://github.com/hexhex/imapplugin.

4.1 Specification

In the following, a specification of the external atoms and action atoms stated above is given.

***#login*[*ConnectionID*, *Hostname*, *Port*, *Username*, *Password*, *SSL*]**

ConnectionID	term identifying an IMAP server to connect to.
Hostname	string containing an IMAP server's hostname.
Port	integer representing an IMAP server's port.
Username	string containing an email account's user name.
Password	string containing an email account's password.
SSL	flag representing whether to establish an encrypted connection or not.

Establishes a connection to the IMAP server with hostname *Hostname* and port *Port* and logs in to the account with user name *Username* and password *Password*.

Other atoms may use term *ConnectionID* in order to make use of the connection established by this atom.
If *SSL* is 0, the connection is unencrypted. Otherwise, the connection is encrypted.
If there already exists a connection using the same *ConnectionID*, the prior connection

gets cut.
If *Hostname*, *Username* or *Password* is not a string or *Port* is not an integer, an error message is printed and the evaluation of the ACTHEX program is aborted.
If the establishment of a connection to the IMAP server specified by *Hostname* and *Port* fails, an error message is printed and the evaluation of the ACTHEX program is aborted.
If the login to the email account specified by *Username* and *Password* fails, an error message is printed and the evaluation of the ACTHEX program is aborted.

Example 4. Let P be the following ACTHEX program:

$\#login[connection1, \text{"}example.com\text{"}, 143, \text{"}user1@example.com\text{"}, \text{"}passw0rd\text{"}, 0].$

In an environment where a mail account with user name $user1@example.com$ and password $passw0rd$ exists on an IMAP-server with hostname "*example.com*" and port 143, the first two answer sets and actions are:

- Evaluation of the answer sets of P, resulting in an empty answer set $\{\}$.
- Execution of the $\#login$-action, causing the establishment of a connection to the specified IMAP server and the login.
- Evaluation of the answer sets of P, again resulting in an empty answer set $\{\}$.
- Execution of the $\#login$-action.

◇

***&logged_in*[*ConnectionID*]**

ConnectionID term identifying an IMAP server connection.

Evaluates to true iff there already exists a connection with the given *ConnectionID*.

Example 5. Let P be the following ACTHEX program:

$\#login[connection1, \text{"}example.com\text{"}, 143, \text{"}user1@example.com\text{"}, \text{"}passw0rd\text{"}, 0] \leftarrow$
$\quad not\ \&logged\_in[connection1].$
$login\_successful \leftarrow \&logged\_in[connection1].$
$-login\_successful \leftarrow not\ \&logged\_in[connection1].$

Here the $-$ in $-login\_successful$ refers to *strong negation* as usual in ASP.

In an environment where an appropriate IMAP-server exists, the first answer sets and actions of P are:

- Evaluation of the answer sets of P, resulting in the answer set $\{-login\_successful\}$.
- Execution of the $\#login$-action, causing the establishment of a connection to the specified IMAP server and the login.

- Evaluation of the answer sets of P, resulting in the answer set $\{login\_successful\}$.

◇

&mailbox[ConnectionID, Path, Name](Mailbox)

ConnectionID	term identifying an IMAP server connection.
Path	string containing a path in which to search for mailboxes.
Name	string containing a mailbox name with wildcards.
Mailbox	string containing a matching mailbox name.

Returns string *Mailbox* containing the name of each mailbox matching name *Name* in path *Path*.

If *Path* or *Name* is not a string, an error message is printed, but the evaluation of the ACTHEX program is being continued.
A mailbox matches *Name* iff its name equals *Name*, where * in *Name* is replaced with any string and % in *Name* is replaced with any string except the mailbox structure's hierarchy delimiter.
If path *Path* does not exist or no mailbox matches *Mailbox*, *&mailbox[ConnectionID, Path, Name](Mailbox)* does not evaluate to true.
If there is no IMAP server connection identified by *ConnectionID* (e.g. because no *#login*-action was performed before or the connection was lost), *&mailbox[ConnectionID, Path, Name](Mailbox)* does not evaluate to true.
If the evaluation of *&mailbox[ConnectionID, Path, Name](Mailbox)* fails, an error message is printed, but the evaluation of the ACTHEX program is being continued.

Example 6. Let P be the following ACTHEX program:

#login[account1, "example.com", 143, "user1@example.com", "passw0rd"].
mailbox(M) ← &mailbox[account1, "INBOX", "Junk"](M).

In an environment where an appropriate IMAP-server exists with a mailbox *INBOX* containing a mailbox *Junk E−Mail*, the first answer sets and actions of P are:

- Evaluation of the answer sets of P, resulting in an empty answer set $\{\}$.
- Execution of the *#login*-action, causing the establishment of a connection to the specified IMAP server and login.
- Evaluation of the answer sets of P, resulting in an answer set $\{mailbox(\text{"INBOX/Junk E−Mail"})\}$.

◇

&filter[ConnectionID, Mailbox, Filter](MessageID)

ConnectionID	term identifying an IMAP server connection.
Mailbox	string containing a mailbox name.
Filter	string containing a filter.
MessageID	integer representing a message ID.

Returns the ID *MessageID* for each message in mailbox *Mailbox* that matches the specified filter *Filter*.

Filter follows the same syntax as the filter argument of the IMAP command *SEARCH*, which is stated in 3.1, but with double quotes replaced by single quotes.
If *Mailbox* or *Filter* is not a string, an error message is printed, but the evaluation of the ACTHEX program is being continued.
If no mailbox with name *Mailbox* exists, $\&filter$[*ConnectionID*, *Mailbox*, *Filter*](*MessageID*) does not evaluate to true.
If there is no IMAP server connection identified by *ConnectionID* (e.g. because no *#login*-action was performed before or the connection was lost), $\&filter$[*ConnectionID*, *Mailbox*, *Filter*](*MessageID*) does not evaluate to true.
If the evaluation of $\&filter$[*ConnectionID*, *Mailbox*, *Filter*](*MessageID*) fails, an error message is printed, but the evaluation of the ACTHEX program is being continued.

Example 7. Let *P* be the following ACTHEX program:

#login[*account1*, "*example.com*", *143*, "*user1@example.com*", "*passw0rd*"].
message_id(*ID*) ←
  $\&filter[account1, "INBOX", "or\ from\ 'user2.example.com'\ deleted"](ID)$.

In an environment where an appropriate IMAP-server exists, the first answer sets and actions of *P* are:

- Evaluation of the answer sets of *P*, resulting in an empty answer set {}.
- Execution of the *#login*-action, causing the establishment of a connection to the specified IMAP server and login.
- Evaluation of the answer sets of *P*, resulting in an answer set {*message_id*(2), *message_id*(3)}.

◇

***&msg_header*[*ConnectionID*, *Mailbox*, *MessageID*](*Type*, *Value*)**

ConnectionID	term identifying an IMAP server connection.
Mailbox	string containing a mailbox name.
MessageID	integer representing a message ID.
Type	term describing a type of header information.
Value	string containing the according header information.

Returns string *Value* containing the header information according to type *Type* of the message with ID *MessageID* in mailbox *Mailbox*.

If *Mailbox* or *Value* is not a string or *MessageID* is not an integer, an error message is printed, but the evaluation of the ACTHEX program is being continued.
If no mailbox with name *Mailbox* or no message with ID *MessageID* in mailbox *Mailbox* exists, &*msg_header*[*ConnectionID*, *Mailbox*, *MessageID*](*Type*, *Value*) does not evaluate to true.
If there is no IMAP server connection identified by *ConnectionID* (e.g. because no #*login*-action was performed before or the connection was lost), &*msg_header*[*ConnectionID*, *Mailbox*, *MessageID*](*Type*, *Value*) does not evaluate to true.
If the evaluation of &*msg_header*[*ConnectionID*, *Mailbox*, *MessageID*](*Type*, *Value*) fails, an error message is printed, but the evaluation of the ACTHEX program is being continued.

Example 8. Let P be the following ACTHEX program:

#*login*[*account1*, "*example.com*", *143*, "*user1@example.com*", "*passw0rd*"].
header(T, V) $\leftarrow$ &*msg_header*[*account1*, "*INBOX*", *3*](T, V).

In an environment where an appropriate IMAP-server exists with a mailbox *INBOX* containing a message with ID 3 and subject *some−subject*, from *user*2@*example.com*, to *user*1@*example.com* and *user*3@*example.com*, the first answer sets and actions of P are:

- Evaluation of the answer sets of P, resulting in an empty answer set $\{\}$.
- Execution of the #*login*-action, causing the establishment of a connection to the specified IMAP server and login.
- Evaluation of the answer sets of P, resulting in an answer set {*header*(*subject*, "*some−subject*"), *header*(*from*, "*user*2@*example.com*"), *header*(*to*, "*user*1@*example.com*"), *header*(*to*, "*user*3@*example.com*")}.

◇

&*msg_body*[*ConnectionID*, *Mailbox*, *MessageID*](*Body*)

ConnectionID	term identifying an IMAP server connection.
Mailbox	string containing a mailbox name.
MessageID	integer representing a message ID.
Body	string containing the requested message's body.

Returns string *Body* containing the body (without possible attachments) of the message with ID *MessageID* in mailbox *Mailbox*.

If *Mailbox* is not a string or *MessageID* is not an integer, an error message is printed, but the evaluation of the ACTHEX program is being continued.
If no mailbox with name *Mailbox* or no message with ID *MessageID* in mailbox *Mailbox*

exists, &*msg_body*[*ConnectionID*, *Mailbox*, *MessageID*](*Body*) does not evaluate to true.
If there is no IMAP server connection identified by *ConnectionID* (e.g. because no #*login*-action was performed before or the connection was lost), &*msg_body*[*ConnectionID*, *Mailbox*, *MessageID*](*Body*) does not evaluate to true.
If the evaluation of &*msg_body*[*ConnectionID*, *Mailbox*, *MessageID*](*Body*) fails, an error message is printed, but the evaluation of the ACTHEX program is being continued.

Example 9. Let P be the following ACTHEX program:

#*login*[*account1*, "*example.com*", *143*, "*user1@example.com*", "*passw0rd*"].
body(*B*) ← &*msg_body*[*account1*, "*INBOX*", *3*](*B*).

In an environment where an appropriate IMAP-server exists with a mailbox *INBOX* containing a message with ID 3 and *some−text* in its body, the first answer sets and actions of P are:

- Evaluation of the answer sets of P, resulting in an empty answer set {}.
- Execution of the #*login*-action, causing the establishment of a connection to the specified IMAP server and login.
- Evaluation of the answer sets of P, resulting in an answer set {*body*("*some−text*")}.

◇

#*set_flag*[*ConnectionID*, *Mailbox*, *MessageID*, *Flag*]

ConnectionID	term identifying an IMAP server connection.
Mailbox	string containing a mailbox name.
MessageID	integer representing a message ID.
Flag	string representing a flag.

Sets the specified flag for the message with ID *MessageID* in mailbox *Mailbox*.

If *Mailbox* or *Flag* is not a string or *MessageID* is not an integer, an error message is printed and the evaluation of the ACTHEX program is aborted.
Flag contains either a flag (without "/") accepted by the server, eventually with a leading *not*. If *Flag* is starts with *not*, the flag is removed from the message's list of flag. Otherwise, it is added to the message's list of flags.
If no mailbox with name *Mailbox* or no message with ID *MessageID* in mailbox *Mailbox* exists, an error message is printed and the evaluation of the ACTHEX program is aborted.
If there is no IMAP server connection identified by *ConnectionID* (e.g. because no #*login*-action was performed before or the connection was lost), no action is performed.
If the execution of #*set_flag*[*ConnectionID*, *Mailbox*, *MessageID*, *Flag*] fails, an error message is printed and the evaluation of the ACTHEX program is aborted.

Example 10. Let P be the following ACTHEX program:

#login[account1, "example.com", 143, "user1@example.com", "passw0rd"]{1}.
#set_flag[account1, "INBOX", 3, "Deleted"]{2}.
#set_flag[account1, "INBOX", 3, "not Seen"]{2}.

In an environment where an appropriate IMAP-server exists with a mailbox *INBOX* containing a message with ID 3, the first answer sets and actions of P are:

- Evaluation of the answer sets of P, resulting in an empty answer set {}.
- Execution of the *#login*-action, causing the establishment of a connection to the specified IMAP server and login.
- Execution of the *#set_flag*-actions, marking the specified message as "\Deleted" and unmarking the specified message as "\Seen".

◇

#copy_msg[ConnectionID, Mailbox, MessageID, NewMailbox]

ConnectionID	term identifying an IMAP server connection.
Mailbox	string containing a mailbox name.
MessageID	integer representing a message ID.
NewMailbox	string containing a mailbox name.

Copies the message with ID *MessageID* in mailbox *Mailbox* to mailbox *NewMailbox.*

If *Mailbox* or *NewMailbox* is not a string or *MessageID* is not an integer, an error message is printed and the evaluation of the ACTHEX program is aborted.
If no mailbox *Mailbox* or *NewMailbox* or no message with ID *MessageID* in mailbox *Mailbox* exists, an error message is printed and the evaluation of the ACTHEX program is aborted.
If there is no IMAP server connection identified by *ConnectionID* (e.g. because no *#login*-action was performed before or the connection was lost), no action is performed.
If the execution of *#copy_msg[ConnectionID, Mailbox, MessageID, NewMailbox]* fails, an error message is printed and the evaluation of the ACTHEX program is aborted.

Example 11. Let P be the following ACTHEX program:

#login[account1, "example.com", 143, "user1@example.com", "passw0rd"]{1}.
#copy_msg[account1, "INBOX", 3, "INBOX/Junk E−Mail"]{2}.

In an environment where an appropriate IMAP-server exists with a mailbox *INBOX* containing a mailbox *Junk E−Mail* and a message with ID 3, the first answer sets and actions of P are:

- Evaluation of the answer sets of P, resulting in an empty answer set {}.
- Execution of the *#login*-action, causing the establishment of a connection to the specified IMAP server and login.

- Execution of the *#copy_msg*-action, causing copying the specified message to *INBOX/Junk E−Mail*.

◇

***#create_msg*[*ConnectionID*, *Mailbox*, *Sender*, *Subject*, *Recipient*, *CC*, *BCC*, *Text*]**

ConnectionID	term identifying an IMAP server connection.
Mailbox	string containing a mailbox name.
Sender	string containing a message's sender.
Subject	string containing a message's subject.
Recipient	function with name *recipient* and an arbitrary number of strings as terms.
CC	function with name *cc* and an arbitrary number of strings as terms.
BCC	function with name *bcc* and an arbitrary number of strings as terms.
Text	string containing a message's text.

Appends the message specified by *Sender*, *Subject*, *Recipient*, *CC*, *BCC*, *Test* to mailbox *Mailbox*.

If *Mailbox*, *Sender*, *Subject* or *Test* is not a string, an error message is printed and the evaluation of the ACTHEX program is aborted.
If no mailbox with name *Mailbox* exists, an error message is printed and the evaluation of the ACTHEX program is aborted.
If *Recipient* is not a function with name *recipient* and strings as terms, an error message is printed and the evaluation of the ACTHEX program is aborted.
If *CC* is not a function with name *cc* and strings as terms, an error message is printed and the evaluation of the ACTHEX program is aborted.
If *CC* is not a function with name *bcc* and strings as terms, an error message is printed and the evaluation of the ACTHEX program is aborted.
If there is no IMAP server connection identified by *ConnectionID* (e.g. because no *#login*-action was performed before or the connection was lost), no action is performed.
If the execution of *#create_msg*[*ConnectionID*, *Mailbox*, *Sender*, *Subject*, *Recipient*, *CC*, *BCC*, *Text*] fails, an error message is printed and the evaluation of the ACTHEX program is aborted.

Example 12. Let P be the following ACTHEX program:

#login[*account1*, "*example.com*", *143*, "*user1@example.com*", "*passw0rd*"]{*1*}.
#create_msg[*account1*, *INBOX*, "*user1@example.com*", "*some−subject*",
"*recipient*("*user2@example.com*", "*user3@example.com*"), *cc*(), *bcc*(), "*some text*"]
{*2*}.

In an environment where an appropriate IMAP-server exists with a mailbox *INBOX*, the first answer sets and actions of P are:

- Evaluation of the answer sets of P, resulting in an empty answer set $\{\}$.
- Execution of the *#login*-action, causing the establishment of a connection to the specified IMAP server and login.
- Execution of the *#create_msg*-action, causing the creation of a new message in *INBOX* with sender *user1@example.com*, subject *some−subject*, recipients *user2@example.com* and *user3@example.com* and text *some text*.

◇

***#move_msg*[*ConnectionID*, *Mailbox*, *MessageID*, *NewMailbox*]**

ConnectionID	term identifying an IMAP server connection.
Mailbox	string containing a mailbox name.
MessageID	integer representing a message ID.
NewMailbox	string containing a mailbox name.

Copies the message with ID *MessageID* from mailbox *Mailbox* to mailbox *NewMailbox* and sets the "\Deleted" flag of the message with ID *MessageID* in mailbox *Mailbox*.

If *Mailbox* or *NewMailbox* is not a string or *MessageID* is not an integer, an error message is printed and the evaluation of the ACTHEX program is aborted.
If no mailbox with name *Mailbox* or *NewMailbox* or no message with ID *MessageID* in mailbox *Mailbox* exists, an error message is printed and the evaluation of the ACTHEX program is aborted.
If there is no IMAP server connection identified by *ConnectionID* (e.g. because no *#login*-action was performed before or the connection was lost), no action is performed.
If the execution of *#move_msg*[*ConnectionID*, *Mailbox*, *MessageID*, *NewMailbox*] fails, an error message is printed and the evaluation of the ACTHEX program is aborted.

Example 13. Let P be the following ACTHEX program:

#login[*account1*, "*example.com*", *143*, "*user1@example.com*", "*passw0rd*"]{*1*}.
#move_msg[*account1*, "*INBOX*", *3*, "*INBOX/Junk E−Mail*"]{*2*}.

In an environment where an appropriate IMAP-server exists with a mailbox *INBOX* containing a mailbox *Junk E−Mail* and a message with ID 3, the first answer sets and actions of P are:

- Evaluation of the answer sets of P, resulting in an empty answer set $\{\}$.
- Execution of the *#login*-action, causing the establishment of a connection to the specified IMAP server and login.
- Execution of the *#move_msg*-action, causing copying the specified message to *INBOX/Junk E−Mail*.

◇

***#expunge_msgs*[*ConnectionID*, *Mailbox*]**

ConnectionID	term identifying an IMAP server connection.
Mailbox	string containing a mailbox name.

Expunges all messages with a "\Deleted" flag in mailbox *Mailbox*.

If *Mailbox* is not a string, an error message is printed and the evaluation of the ACTHEX program is aborted.
If no mailbox with name *Mailbox* exists, an error message is printed and the evaluation of the ACTHEX program is aborted.
If there is no IMAP server connection identified by *ConnectionID* (e.g. because no *#login*-action was performed before or the connection was lost), no action is performed.
If the execution of *#expunge_msgs*[*ConnectionID*, *Mailbox*] fails, an error message is printed and the evaluation of the ACTHEX program is aborted.

Example 14. Let P be the following ACTHEX program:

#login[*account1*, *"example.com"*, *143*, *"user1@example.com"*, *"passw0rd"*]{*1*}.
#expunge_msgs[*account1*, *"INBOX"*]{*2*}.

In an environment where an appropriate IMAP-server exists with a mailbox *INBOX* containing messages, the first answer sets and actions of P are:

- Evaluation of the answer sets of P, resulting in an empty answer set {}.
- Execution of the *#login*-action, causing the establishment of a connection to the specified IMAP server and login.
- Execution of the *#expunge_msgs*-action, resulting in an empty *INBOX*.

◇

***#create_mailbox*[*ConnectionID*, *Mailbox*]**

ConnectionID	term identifying an IMAP server connection.
Mailbox	string containing a new mailbox name.

Creates mailbox *Mailbox*.

If *Mailbox* is not a string, an error message is printed and the evaluation of the ACTHEX program is aborted.
If there is no IMAP server connection identified by *ConnectionID* (e.g. because no *#login*-action was performed before or the connection was lost), no action is performed.
If the execution of *#create_mailbox*[*ConnectionID*, *Mailbox*] fails, an error message is printed and the evaluation of the ACTHEX program is aborted.

Example 15. Let P be the following ACTHEX program:

#login[*account1*, *"example.com"*, *143*, *"user1@example.com"*, *"passw0rd"*]{*1*}.
#create_mailbox[*account1*, *"INBOX/Mailbox2"*]{*2*}.

In an environment where an appropriate IMAP-server exists with a mailbox *INBOX*, the first answer sets and actions of P are:

- Evaluation of the answer sets of P, resulting in an empty answer set {}.
- Execution of the *#login*-action, causing the establishment of a connection to the specified IMAP server and login.
- Execution of the *#create_mailbox*-action, causing the creation of *Mailbox2* in *INBOX*.

◇

***#delete_mailbox*[*ConnectionID*, *Mailbox*]**

ConnectionID	term identifying an IMAP server connection.
Mailbox	string containing a mailbox name.

Deletes mailbox *Mailbox*.

If *Mailbox* is not a string, an error message is printed and the evaluation of the ACTHEX program is aborted.
If no mailbox with name *Mailbox* exists, an error message is printed and the evaluation of the ACTHEX program is aborted.
If there is no IMAP server connection identified by *ConnectionID* (e.g. because no *#login*-action was performed before or the connection was lost), no action is performed.
If the execution of *#delete_mailbox*[*ConnectionID*, *Mailbox*] fails, an error message is printed and the evaluation of the ACTHEX program is aborted.

Example 16. Let P be the following ACTHEX program:

#login[*account1*, *"example.com"*, *143*, *"user1@example.com"*, *"passw0rd"*]{*1*}.
#delete_mailbox[*account1*, *"INBOX"*]{*2*}.

In an environment where an appropriate IMAP-server exists with a mailbox *INBOX*, the first answer sets and actions of P are:

- Evaluation of the answer sets of P, resulting in an empty answer set {}.
- Execution of the *#login*-action, causing the establishment of a connection to the specified IMAP server and login.
- Execution of the *#delete_mailbox*-action, causing the deletion of *INBOX*.

◇

***#rename_mailbox*[*ConnectionID*, *Mailbox*, *NewMailbox*]**

ConnectionID	term identifying an IMAP server connection.
Mailbox	string containing a mailbox name.
NewMailbox	string containing a mailbox name.

Renames mailbox *Mailbox* to *NewMailbox*.

If *Mailbox* or *NewMailbox* is not a string, an error message is printed and the evaluation of the ACTHEX program is aborted.
If no mailbox with name *Mailbox* exists, an error message is printed and the evaluation of the ACTHEX program is aborted.
If there is no IMAP server connection identified by *ConnectionID* (e.g. because no *#login*-action was performed before or the connection was lost), no action is performed.
If the execution of *#rename_mailbox*[*ConnectionID*, *Mailbox*, *NewMailbox*] fails, an error message is printed and the evaluation of the ACTHEX program is aborted.

Example 17. Let P be the following ACTHEX program:

#login[*account1*, ”*example.com*”, *143*, ”*user1@example.com*”, ”*passw0rd*”]{*1*}.
#rename_mailbox[*account1*, ”*Mailbox1*”, ”*Mailbox2*”]{*2*}.

In an environment where an appropriate IMAP-server exists with a mailbox *Mailbox1*, the first answer sets and actions of P are:

- Evaluation of the answer sets of P, resulting in an empty answer set {}.
- Execution of the *#login*-action, causing the establishment of a connection to the specified IMAP server and login.
- Execution of the *#rename_mailbox*-action, causing *Mailbox* to be renamed to *Mailbox2*.

◇

***#reply*[*ConnectionID*, *Mailbox*, *MessageID*, *Text*]**

ConnectionID	term identifying an IMAP server connection.
Mailbox	string containing a mailbox name.
MessageID	integer representing a message ID.
Text	string containing a text to be replied.

Creates a draft of a message containing *Text* as text and the subject with prefixed "Re: " of the message with ID *MessageID* in mailbox *Mailbox* as subject. The created message has the sender of the message with ID *MessageID* in mailbox *Mailbox* as its recipient, the mail address of the logged in user as its sender and the same values in cc as the message with ID *MessageID* in mailbox *Mailbox*.

If *Mailbox* or *Text* is not a string or *MessageID* is not an integer, an error message is printed and the evaluation of the ACTHEX program is aborted.
If no mailbox with name *Mailbox* or no message with ID *MessageID* in mailbox *Mailbox* exists, an error message is printed and the evaluation of the ACTHEX program is aborted.
If there is no IMAP server connection identified by *ConnectionID* (e.g. because no *#login*-action was performed before or the connection was lost), no action is performed.
If the execution of *#reply*[*ConnectionID*, *Mailbox*, *MessageID*, *Text*] fails, an error message is printed and the evaluation of the ACTHEX program is aborted.

Example 18. Let P be the following ACTHEX program:

#login[*account1*, "*example.com*", *143*, "*user1@example.com*", "*passw0rd*"]{*1*}.
#reply[*account1*, "*INBOX*", *3*, "*Text to be replied*"]{*2*}.

In an environment where an appropriate IMAP-server exists with a mailbox *INBOX* containing a message with ID 3, subject *example subject*, sender *user*2@*example.com* and recipients *user*1@*example.com* and *user*3@*example.com*, the first answer sets and actions of P are:

- Evaluation of the answer sets of P, resulting in an empty answer set {}.
- Execution of the *#login*-action, causing the establishment of a connection to the specified IMAP server and login.
- Execution of the *#reply*-action, causing the creation of a message with subject *Re*: *example subject*, sender *user*1@*example.com* and recipients *user*2@*example.com* and *user*3@*example.com* with a "\Draft" flag.

◇

4.2 Application Scenarios

In the following, the use of the IMAP plugin specified above is illustrated by some examples that show possible application scenarios.

Example 19. The following ACTHEX program copies all messages of mailbox A that contain the word *IMAP* in their subject, to mailbox B:

#login[*imap_server_connection_1*, "*example.com*", *143*, "*user1@example.com*", "*password123*", *0*] ← not &*logged_in*[*imap_server_connection_1*].

message_to_be_copied(*ID*) ←
&*filter*[*imap_server_connection_1*, "*A*", "*subject 'IMAP'*"](*ID*).

#copy_msg[*imap_server_connection_1*, "*A*", *ID*, "*B*"] ←
message_to_be_copied(*ID*).

The above program exemplifies ACTHEX' iterative evaluation: The first iteration results in an empty answer set {}, because &*logged_in* does not evaluate to true, since the IMAP server connection has not been established yet, and no *message_to_be_copied* atom is true, again since the IMAP server connection has not been established yet. The empty answer set then triggers the execution of the #*login* action. The answer set resulting from the second iteration contains a *message_to_be_copied* atom for every message with subject *IMAP* in mailbox *A*. Because &*logged_in* does not evaluate to true anymore, the #*login* is not executed a second time, but for every *message_to_be_copied* atom, the #*copy_msg* action is executed. ◇

Example 20. The following ACTHEX program creates a mailbox for every sender, from whom > 5 messages exist in mailbox *A*, and moves all messages from this sender to the newly created mailbox:

$\#login[imap\_server\_connection\_1,\ "example.com",\ 1143,\ "user1@example.com",$
$\quad "password123",\ 0] \leftarrow \text{not } \&logged\_in[imap\_server\_connection\_1].$

$msg(F,\ ID) \leftarrow$
$\quad \&filter[imap\_server\_connection\_1,\ "A",\ "all"](ID),$
$\quad \&msg\_header[imap\_server\_connection\_1,\ "A",\ ID](from,\ F).$

$new\_mailbox(F) \leftarrow \#count\{ID : msg(F,\ ID)\} > 5,\ msg(F,\ \_).$

$\#create\_mailbox[imap\_server\_connection\_1, F)\{1\} \leftarrow$
$\quad new\_mailbox(F),$
$\quad \&concat["MB",\ F](X).$

$\#copy\_msg[imap\_server\_connection\_1,\ M,\ ID,\ X)\{2\} \leftarrow$
$\quad new\_mailbox(F),$
$\quad \&filter[imap\_server\_connection\_1,\ M,\ Y](ID),$
$\quad \&mailbox[imap\_server\_connection\_1,\ "*",\ "*"](M),$
$\quad \&concat["from\ '",\ F,\ "'"](Y),$
$\quad \&concat["MB",\ F](X).$

The first evaluation iteration of the above program is used for creating a connection with the IMAP server. (Rule 1)

In subsequent evaluations, the actual creation of mailboxes and moving of messages is done: for every message in mailbox *A* an atom *msg*(*F*, *ID*) with the message's sender *F* and the message ID *ID* is generated and for every sender *F* that is contained in > 5 *msg* atoms, a *new_mailbox* atom is generated using the #*count* atom. (Rules 2 and 3)

According to the *new_mailbox* atoms contained in the answer sets, new mailboxes are then created and messages are copied there. (Rule 4 and 5)

Since there is no rule that stops the iterative evaluation, the program will be evaluated repeatedly and with every evaluation a new mailbox is created for every *new_mailbox* atom and messages are copied there. If stopping the iterative evaluation is desired after the mailboxes have been created once, an according rule has to be added to the program. $\#acthexStop\{3\} \leftarrow new\_mailbox(F)$ for example will cause the evaluation of the program to stop if a *new_mailbox* atom is contained an answer set. Since *#acthexStop* has a higher *action precedence* than *#create_mailbox* and *#copy_msg*, they are executed before the evaluation is stopped. ◇

Example 21. The following ACTHEX program deletes every message from each sender, from whom a message exists that has already been deleted:

$\#login[imap\_server\_connection\_1, "example.com", 1143, "user1@example.com",$
$\quad "password123", 0] \leftarrow \text{not}\&logged\_in[imap\_server\_connection\_1].$

$delete(F) \leftarrow$
$\quad \&filter[imap\_server\_connection\_1, M, "deleted"](ID),$
$\quad \&mailbox[imap\_server\_connection\_1, " * ", " * "](M),$
$\quad \&msg\_header[imap\_server\_connection\_1, M,\ ID](from, F).$

$\#set\_flag[imap\_server\_connection\_1, M,\ ID, "Deleted"] \leftarrow$
$\quad \&filter[imap\_server\_connection\_1, M, "all"](ID),$
$\quad \&mailbox[imap\_server\_connection\_1, " * ", " * "](M),$
$\quad \&msg\_header[imap\_server\_connection\_1, M, ID](from, F), delete(F).$

The above program performs a login after the first evaluation iteration and then deletes every message from each sender, from whom a message exists that has already been deleted. Like examples 19 and 20, the program makes use of the fact that the external atoms and action atom for interacting with IMAP servers do not appear in answer sets (respectively are not executed) before a login was performed. This way, the interaction with the IMAP server, is always started with a login, without the need of additional control structures (like checking if *&logged_in[imap_server_connection_1]* evaluates to true). ◇

CHAPTER 5

TESTING

In the process of the implementation of the IMAP Plugin for ACTHEX, it is necessary to test the developed software in order to verify its correct behaviour. The basic test strategy is to run an ACTHEX solver in a controllable test environment with the IMAP plugin and an ACTHEX program, that contains the external atoms and action atoms provided by the plugin, and afterwards verify if the plugin operated as expected. For this purpose, test cases that test certain aspects of the plugin are created. The selection of test cases is made with the intention of achieving a complete code coverage, meaning that all parts of the software's source code are covered by a test case [AO08]. Besides the selection of test cases, a test setup that provides such a controllable environment needs to be established. Such a test setup consists of several components and subsystems that are interfacing each other and allow the execution of the test cases. Figure 5.1 provides a schematic illustration of the architecture of the test setup. An implementation of this test setup is available at http://www.kr.tuwien.ac.at/research/systems/dlvhex/imapplugin.html and http://github.com/hexhex/imapplugin.

In the following sections, the architecture of the test setup is described:

5.1 Test Script

In order to test the IMAP plugin, the test setup needs to include a script that is responsible for the execution of the test cases, including the assurance of certain preconditions regarding the other components of the test script, the actual execution of the ACTHEX programs that test a certain functionality of the IMAP plugin and the verification of the expected test results.

In particular, the test script ensures that the IMAP server is set up correctly, which requires the creation of user accounts and mailboxes, the insertion of messages into these mailboxes, clearing the IMAP server's logfile and starting the IMAP server. In order to

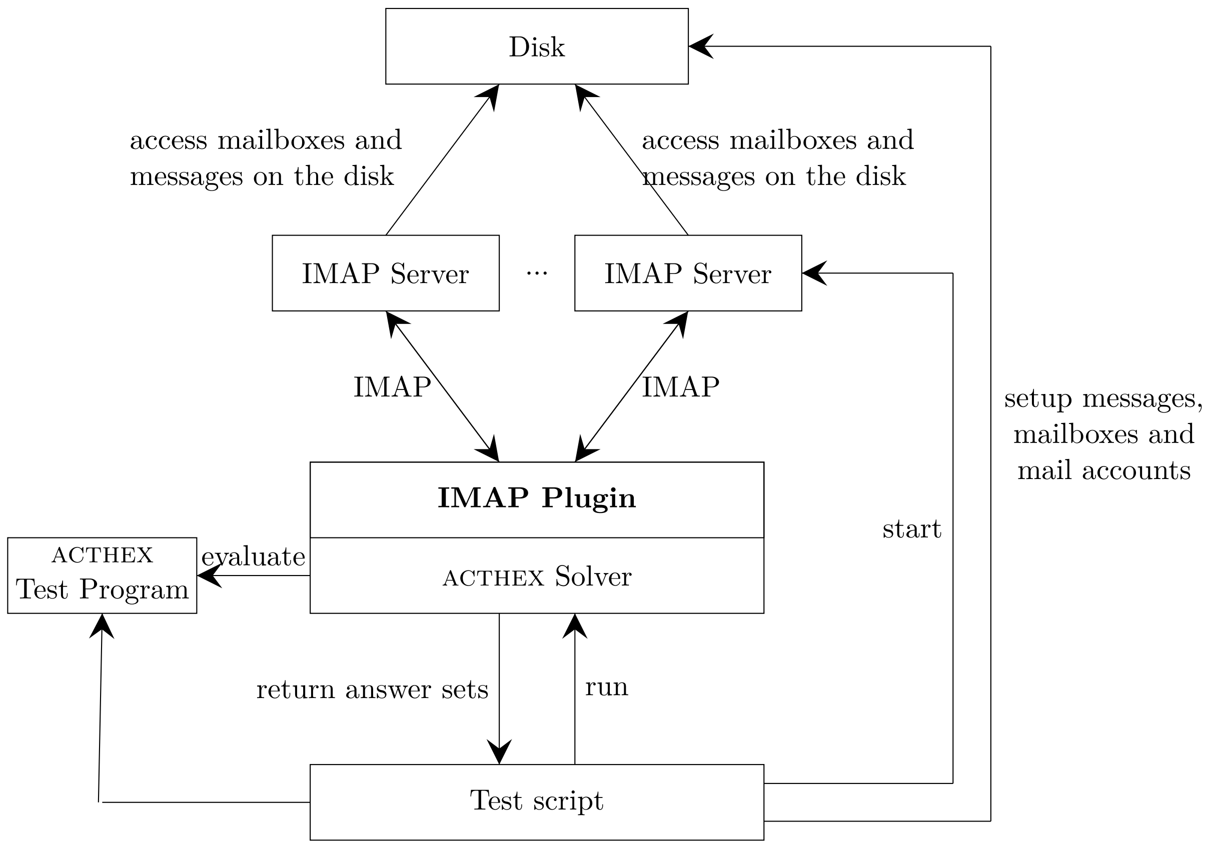
Disk
access mailboxes and messages on the disk
access mailboxes and messages on the disk
IMAP Server
···
IMAP Server
IMAP
IMAP
setup messages, mailboxes and mail accounts
IMAP Plugin
start
ACTHEX Test Program
evaluate
ACTHEX Solver
return answer sets
run
Test script

Figure 5.1: Architecture of the test setup

be able to fully control these aspects, the test script needs access to the IMAP server's underlying data storage. This can be achieved, by using an IMAP server that runs on the same machine as the rest of the test setup. Therefore, an IMAP server implementation that satisfies the requirements for this application, has to be chosen.

After a test run, the test script asserts that the expected postconditions are satisfied in order to verify the correct behaviour of the IMAP plugin. For this purpose, it inspects the generated answer sets, the state of the IMAP server (e.g. by reading its logfile) and the output produced by the answer set solver.

5.2 ACTHEX Solver

The ACTHEX solver used in the test setup is HEXLITE, which is an implementation of parts of HEX, also containing a partial implementation of ACTHEX. HEXLITE is an open-source software, developed by Peter Schüller as an easy-to-use and readily available HEX solver. It provides an interface for Python programs that act as plugins and implement the evaluation of external atoms and action atoms for ACTHEX programs. The advantage of using HEXLITE in the test setup is that it supports plugins written in Python, a very common programming language. Another advantage is that it is available through various package management systems for different operating systems and therefore easy to install [Sch18].

5.3 IMAP Server

In order to verify the correct behaviour of the IMAP plugin, it is substantial to allow the access to an IMAP server for test purposes. Since there are a lot of free IMAP servers available over the internet, like Google Gmail, it seems obvious to simply use an already available IMAP server. In fact, to ensure controllable and repeatable test results that are independent from outside influences, direct access to the IMAP server's user accounts, mailboxes and messages is required. The IMAP server, therefore, has to be operated on the local machine of the test setup. For this purpose an appropriate IMAP server implementation has to be chosen.

A fairly popular one is *Courier IMAP*, which is published under the General Public License (GPL) and therefore allows unrestrained use for the purposes of this work. It provides versatile configuration options that allow many different application scenarios, especially the use as an IMAP server in a test setup like this. *Courier IMAP* can be configured to only accept Transport Layer Security (TLS) encrypted connections as well as only unencrypted connections. This property provides us with the possibility to establish unencrypted or encrypted connections. It furthermore supports the operating system's user accounts as well as various database technologies for managing the user authentication [Var00].

Another IMAP server implementation which is considered in the decision of the IMAP server to use for the test setup is *Dovecot*, which is also licensed with GPL. According to the documentation of *Dovecot*, it can also be configured to only accept TLS encrypted as well as only unencrypted connections to IMAP clients and also supports several technologies for the management of users, including various database management systems or the user authentication of the operating system via *passwd* files. *Dovecot*, furthermore, can be configured to be operated by a user without root privileges, which is a major advantage for the use in a test setup [Sir02].

Both IMAP server implementations would possibly meet the requirements for this test setup. Since Dovecot has the advantage of supporting operation without root privileges, it is the IMAP server of choice and will be used in this work.

5.4 Implementation of the Test Setup

An implementation of this test setup is available at http://github.com/hexhex/imapplugin, together with a collection of 62 test cases that verify the correct behaviour of the IMAP plugin. A list of these test cases is available in

Appendix A. The test cases use the Python test framework *unittest*, which allows for the execution of test runs with large sets of test cases and provides statistical information about test runs. All test cases are divided into the four phases *setup*, *execute*, *verify* and *teardown* and provide detailed information logged to the console. In the following, a part of the console input and output, when running the test setup, is given:

```
$ python runtests.py /
runtests      -    INFO - load test module testlogin
runtests      -    INFO - load test module testfilter
runtests      -    INFO - load test module testmailbox
runtests      -    INFO - load test module testmsgheader
runtests      -    INFO - load test module testmsgbody
runtests      -    INFO - load test module testsetflag
runtests      -    INFO - load test module testcopymsg
runtests      -    INFO - load test module testcreatemsg
runtests      -    INFO - load test module testmovemsg
runtests      -    INFO - load test module testexpungemsgs
runtests      -    INFO - load test module testcreatemailbox
runtests      -    INFO - load test module testdeletemailbox
runtests      -    INFO - load test module testrenamemailbox
runtests      -    INFO - load test module testreply
...
.testfilter    -    INFO - run test filter_no_messages
testutil      -    INFO - set server state
testutil      -    INFO - start dovecot
testutil      -    INFO - dovecot now running with PID 6334
testutil      -    INFO - run hexlite with program
testcases/filter_success.hex, plugin path ('../',) and plugins
('imapplugin',)
testutil      -    INFO - kill dovecot
.testfilter    -    INFO - run test filter_wrong_connection_id
testutil      -    INFO - set server state
testutil      -    INFO - start dovecot
testutil      -    INFO - dovecot now running with PID 6345
testutil      -    INFO - run hexlite with program
testcases/filter_wrong_connection_id.hex, plugin path ('../',)
and plugins ('imapplugin',)
testutil      -    INFO - kill dovecot
.testfilter    -    INFO - run test filter_wrong_mailbox
testutil      -    INFO - set server state
testutil      -    INFO - start dovecot
testutil      -    INFO - dovecot now running with PID 6355
testutil      -    INFO - run hexlite with program
testcases/filter_wrong_mailbox.hex, plugin path ('../',) and
```

```
plugins ('imapplugin',)
testutil        -       INFO - kill dovecot
...
```

The characteristics of the test cases are illustrated in the following exemplary test case:

```
def filter_success(self):
    """
    setup the imap server,
    start the imap server,

    run an acthex program that contains filter atoms,

    assert that the evaluation of the acthex program results in the
    expected output,
    assert that the expected answer set is evaluated
    """

    logging.info("run␣test␣filter_success")

    # setup
    set_server_state(([("user1@example.com", "passw0rd123"),],
        [("INBOX", "wanted_sender", "wanted_subject", ["recipient11",
            "recipient21"], [], ["bcc11", "bcc12", "bcc13"], ""),
        ("INBOX", "wanted_sender", "subject1", ["recipient21",
            "wanted_recipient"], [], ["bcc2", ], ""),
        ("INBOX", "sender3", "wanted_subject", ["recipient31",
            "recipient32"], ["cc3"], [], ""),
        ("INBOX", "wanted_sender", "wanted_subject", ["recipient41",
            "wanted_recipient"], [], [], ""),
        ("INBOX", "wanted_sender", "subject5", ["recipient5"],
            ["cc51", "cc52"], [], "")
        ]))
    pid = start_server()

    # execute
    acthex_program = TEST_DIR + "filter_success.hex"
    plugin_paths = (PLUGIN_DIR,)
    plugins = (IMAP_PLUGIN,)

    out, err = run_hexlite(acthex_program, plugin_paths, plugins)

    # verify
    assert out != None, "unexpected␣hexlite␣stdout␣output␣"
```

```
        + str(out)
    assert err == None, "unexpected␣hexlite␣stderr␣output␣"
        + str(err)

    answersets = get_answersets(out)
    assert len(answersets) >= 1
        and len(answersets[-1].split(",")) == 3,
            "unexpected␣answer␣set␣" + str(answersets)

    # teardown
    kill_server(pid)
```

The above test case tests the correct behaviour of the action atom *#filter*. For this purpose, in the *setup* phase messages are created and stored on the IMAP server. Afterwards, the IMAP server is started. In the *execute* phase the following ACTHEX program is run with HEXLITE:

$\#login[test\_server,\ "localhost",\ 10143,\ "user1@example.com",$
$"passw0rd123",\ 0]\{1\} \leftarrow \text{not}\ \&logged\_in[test\_server].$

$found(ID) \leftarrow$
$\&logged\_in[test\_server],$
$\&filter[test\_server,\ "INBOX",\ "from'wanted\_sender'\ (or\ to\ 'wanted\_recipient'$
$subject\ 'wanted\_subject')"](ID).$

$\#acthexStop\{2\} \leftarrow \&logged\_in[test\_server].$

In the *verify* phase the output of HEXLITE and the evaluated answer sets are verified and in the *teardown* phase the IMAP server is terminated.

5.4.1 Results

The implementation of the test setup described above validates the implementation of the IMAP plugin and the correctly working interaction between ACTHEX programs and IMAP servers. It furthermore demonstrates possible usages of the plugin and how knowledge from email messages can be integrated into ASP.

CHAPTER 6

DISCUSSION

The IMAP plugin for ACTHEX developed as part of this work, provides an interface between the ASP language ACTHEX and IMAP, in order to allow the integration of external knowledge provided in email messages into ACTHEX programs and to allow influencing email accounts through the execution of actions. As a result arises the possibility of managing email accounts and processing email messages (e.g. manipulating messages based on knowledge that was reasoned from information provided in other messages) with answer set programs.

This chapter focuses on the impact of such a plugin on ASP and its relevance among other methods for manipulating email messages beyond ASP. Especially, attention should be turned on a critical examination of the developed plugin and the results of this work.

Although the IMAP plugin supports both SSL-encrypted and unencrypted connections to IMAP servers, the server authentication through the IMAP plugin has a substantial disadvantage regarding security, which is the fact that mail account credentials (esp. passwords) have to be put in the ACTHEX programs as plain text. This circumstance constrains the practical relevance of the IMAP plugin significantly and makes it unusable for many practical applications. A possibility to encounter this problem could be to include the option of authenticating using an authorisation protocol like *OAuth*, which is also supported by many commercial IMAP servers.

Another disadvantage of the IMAP plugin is the fact that it is based only on a selection of the entire functionality of IMAP and, thus, is restricted in its capabilities, regarding applications where parts of IMAP that are missing in this plugin would be needed. This selection had to be made, because of the fact that the entire IMAP functionality would have been too comprehensive for the scope of this work. IMAP feature that are not covered by this work are the identification of messages by *UIDs* (identifiers that are unique across mailboxes), the *SUBSCRIBE* or the *UNSUBSCRIBE* command, any IMAP extensions such as the "Internet Message Access Protocol - SORT and THREAD

Extensions" [MC08] or any *experimental commands.* (Note that this enumeration is not exhaustive.)

Despite the limitations and downsides stated above, the IMAP plugin for ACTHEX has also some practical relevance. Thinking of special applications or of the assessment of the basic idea of the interaction with IMAP servers from answer set programs, this work contains quite interesting practical outcomes. Besides that, the IMAP plugin for ACTHEX can take advant?age of the benefits of ASP, like its advantages at solving computationally complex problems. Due to the expressiveness of ASP in general and ACTHEX in particular, furthermore, ACTHEX programs are often relatively compact and easy to understand.

To demonstrate possible applications of the IMAP plugin in practice, the message filtering with *procmail* and *Sieve*, demonstrated in Examples 2 and 3, is going to be implemented as an ACTHEX program:

Example 22. The following ACTHEX program implements the behaviour of the *procmail* filter rule in example 2. It deletes all messages from *unwanted@example.com*:

$\#login[connection\_1,\ "example.com",\ 143,\ "user1@example.com",$
$\quad "password123",\ 0] \leftarrow \text{ not } \&logged\_in[connection\_1].$

$\#set\_flag[connection\_1,\ "INBOX",\ ID,\ "Deleted"]\{1\} \leftarrow$
$\quad \&filter[connection\_1,\ "INBOX",\ "from\ 'unwanted@example.com'"](ID).$

$\#acthexStop\{1\} \leftarrow \&logged\_in[connection\_1].$ ◇

Example 23. The following ACTHEX program implements the behaviour of the *Sieve* filter rule in example 2. It moves all messages from *user1@example.com* into mailbox *INBOX.mails-from-user1.*

$\#login[connection\_1,\ "example.com",\ 143,\ "user1@example.com",$
$\quad "password123",\ 0] \leftarrow \text{ not } \&logged\_in[connection\_1].$

$\#move\_msg[connection\_1,\ "INBOX",\ ID,\ "INBOX.mails-from-user1"]\{1\} \leftarrow$
$\quad \&mailbox[connection\_1,\ "*",\ "*"](M),$
$\quad \&filter[connection\_1,\ M, "all"](ID),$
$\quad \&msg\_header[connection\_1,\ M, ID]("from",\ "user1@example.com").$ ◇

Compared to the message filtering Examples 2 and 3 done with *procmail* and *Sieve*, the above examples obviously contain more code. Having to write more code to produce the same behaviour seems to be a disadvantage at first sight, but the ACTHEX formalism also comes with more expressiveness than other message filtering systems. Especially the explicit representation of knowledge e.g. the representation of relationships between entities like messages, senders or recipients are not possible with other message filters.

Example 24. The following ACTHEX program identifies messages that were sent as part of one mail communication (also known as a *thread*) and represents them as the relationship $thread(A, B)$ were A, B are pairs ($mailbox$, $message\ ID$). Messages identified as part of a *thread* could then be possibly treated in a particular way by an approiate ACTHEX program. Such a complex relationship between can only barely or even not at all be implemented with e.g. *procmail* or *Sieve*.

The multiple rules with *thread* atoms in their head are necessary in order to cover all possible cases when messages are considered as part of a *thread*. N.B., this program does not cover all cases in which email messages would usually be considered as part of a *thread*. Besides, recent versions of IMAP have an extension that contains functionality to organize messages in *threads* by themselves [MC08].

$\#login[connection\_1,\ "example.com",\ 143,\ "user1@example.com",$
$\quad "password123",\ 0]\ \leftarrow \text{not}\ \&logged\_in[connection\_1].$

$mailbox(M)\ \leftarrow\ \&mailbox[connection\_1,\ "*",\ "*"](M).$

$msg(M,\ ID)\ \leftarrow\ \&filter[connection\_1,\ M,\ "all"](ID),\ mailbox(M).$

$header(M,\ ID,\ T,\ V)\ \leftarrow\ \&msg\_header[connection\_1,\ M,\ ID](T,\ V),\ msg(M,\ ID).$

$thread((M1,\ ID1),\ (M2,\ ID2))\ \leftarrow$
$\quad header((M1,\ ID1),\ subject,\ S1),\ header((M2,\ ID2),\ subject,\ S2),$
$\quad \&concat("Re:\ ",\ S1)[RS],\ S2\ =\ RS,$
$\quad header((M1,\ ID1),\ from,\ F),\ header((M2,\ ID2),\ to,\ T),\ T = F.$

$thread((M1,\ ID1),\ (M2,\ ID2))\ \leftarrow$
$\quad header((M1,\ ID1),\ subject,\ S1),\ header((M2,\ ID2),\ subject,\ S2),$
$\quad \&concat("Re:\ ",\ S1)[RS],\ S2\ =\ RS,$
$\quad header((M1,\ ID1),\ to,\ T),\ header((M2,\ ID2),\ from,\ F),\ T = F.$

$thread((M1,\ ID1),\ (M2,\ ID2))\ \leftarrow$
$\quad header((M1,\ ID1),\ subject,\ S1),\ header((M2,\ ID2),\ subject,\ S2),\ S1\ =\ S2,$
$\quad header((M1,\ ID1),\ from,\ F),\ header((M2,\ ID2),\ to,\ T),\ T = F.$

$thread((M1,\ ID1),\ (M2,\ ID2))\ \leftarrow$
$\quad header((M1,\ ID1),\ subject,\ S1),\ header((M2,\ ID2),\ subject,\ S2),$
$\quad \&concat("Re:\ ",\ S1)[RS],\ S2\ =\ RS,$
$\quad header((M1,\ ID1),\ from,\ F1),\ header((M2,\ ID2),\ from,\ F2),\ F1 = F2.$

$thread((M1,\ ID1),\ (M2,\ ID2))\ \leftarrow$
$\quad header((M1,\ ID1),\ subject,\ S1),\ header((M2,\ ID2),\ subject,\ S2),\ S1\ =\ S2,$
$\quad header((M1,\ ID1),\ from,\ F1),\ header((M2,\ ID2),\ from\ F2),\ F1 = F2.$

$thread((M1,\ ID1),\ (M2,\ ID2))\ \leftarrow$
$\quad header((M1,\ ID1),\ subject,\ S1),\ header((M2,\ ID2),\ subject,\ S2),$
$\quad \&concat("Re:\ ",\ S1)[RS],\ S2\ =\ RS,$
$\quad header((M1,\ ID1),\ to,\ T1),\ header((M2,\ ID2),\ to,\ T2),\ T1 = T2.$

$thread((M1,\ ID1),\ (M2,\ ID2))\ \leftarrow$
$\quad header((M1,\ ID1),\ subject,\ S1)\ header((M2,\ ID2),\ subject,\ S2),\ S1\ =\ S2,$
$\quad header((M1,\ ID1),\ to,\ T1),\ header((M2,\ ID2),\ to\ T2),\ T1 = T2.$

◇

Another aspect in favour of the IMAP plugin regarding Examples 2 and 3 is that *procmail* or *Sieve* rules are always applied to incoming messages, whereas ACTHEX programs filtering email messages can be run at any time and especially also be applied on messages that have already been received in the past.

CHAPTER 7

SUMMARY AND FUTURE WORK

This chapter summarizes the results emerging from this work.

7.1 Summary

The idea of ACTHEX is the extension of traditional ASP by external atoms and action atoms, and the concept of the iterative program evaluation. This work considers ASP in the context of email and uses ACTHEX to work out an approach of interacting with IMAP servers from answer set programs. A basic concept of ACTHEX, which is the iterative program evaluation, is utilized in order to repeatedly access information from messages stored on IMAP servers, perform reasoning and then interact with IMAP servers based on the reasoning. The interaction between ACTHEX programs and IMAP servers is facilitated by ACTHEX' extensibility that allows the access to external environments through plugins. On this basis, a plugin for ACTHEX that comprises the parts of the IMAP functionality that have been considered to be most relevant in this context, is developed. The process of the development of this plugin also involves considerations for the structuring of the plugin (i.e. how the IMAP functions are mapped onto external atoms and action atoms) in order to maximize its usability and for the validation of its expected behaviour. The IMAP plugin has versatile applications purposes and facilitates the integration of knowledge from email messages into answer set programs, which can then be used e.g. to manipulate email messages and mailboxes on IMAP servers.

7.2 Future Work

The results discussed in Chapter 6 point out some of the drawbacks and needs for improvement of the IMAP plugin, but also show that it can benefit from the advantages of ASP.

From the results of these considerations various possible approaches for improvements and advances emerge that can be covered by future works.

A promising application of the IMAP plugin seems to be message filtering systems such as spam filters. For this purpose also the integration of machine learning into ASP in the context of email could be an interesting approach. Employing machine learning techniques, like *inductive logic programming*, on ASP has already lead to promising research results [EK16, CRL12]. A possible topic for future works could therefore be automated classification of email messages based on *inductive logic programming*, that could for example be aimed at the development of intelligent spam filters or intelligent agents for the creation of automatic or machine-aided responses to messages.

Appendix A

In the following, the names of the Python functions that are implemented as part of the test setup are listed with a short explanation of what is tested. Each function contains one test case and tests a particular part of the functionality of the implementation of the IMAP plugin that was developed as part of this work. The list is structured according to the external atoms and action atoms that are object of the test cases.

- *#login*:

  1. `login_success`: positive test
  2. `login_username_no_string`: username is no string
  3. `login_with_already_used_connection_id`: connection ID already in use
  4. `login_wrong_hostname`: non-existing host name
  5. `not_logged_in`: positive test for &*logged_in*
  6. `login_wrong_password`: wrong password

- &*filter*:

  7. `filter_success`: positive test
  8. `filter_no_messages`: no messages on the IMAP server
  9. `filter_wrong_connection_id`: non-existing connection ID
  10. `filter_wrong_mailbox`: non-exiting mailbox name
  11. `filter_invalid_filter`: invalid filter string

- &*mailbox*:

  12. `mailbox`: positive test
  13. `mailbox_wrong_connection_id`: non-existing connection ID
  14. `mailbox_no_mailboxes`: no mailboxes on the IMAP server
  15. `mailbox_invalid_path`: invalid mailbox path
  16. `mailbox_name_pattern`: invalid mailbox name pattern

- &*msg_header*:

  17. `msg_header_success`: positive test
  18. `msg_header_wrong_connection_id`: non-existing connection ID
  19. `msg_header_wrong_mailbox`: non-existing mailbox name
  20. `msg_header_wrong_msg_id`: non-existing message ID

21. `msg_header_different_mailboxes`: positive test if headers can be fetched from messages of multiple mailboxes

- *&msg_body*:

22. `msg_body_success`: positive test
23. `msg_body_no_text`: positive test with a message without a text
24. `msg_body_wrong_connection_id`: non-existing connection ID
25. `msg_body_wrong_mailbox`: non-existing mailbox name
26. `msg_body_wrong_message_id`: non-existing message ID

- *#set_flag*:

27. `set_deleted`: positive test for setting the *Deleted* flag
28. `set_not_deleted`: positive test for setting a *not Deleted* flag
29. `set_draft_seen`: positive test for setting a *Draft* and a *Seen* flag
30. `set_deleted_wrong_connection_id`: non-existing connection ID
31. `set_deleted_wrong_mailbox`: non-existing mailbox name
32. `set_deleted_wrong_message_id`: non-existing message ID

- *#copy_msg*:

33. `copy_msg`: positive test
34. `copy_msg_wrong_mailbox`: non-existing mailbox name
35. `copy_msg_wrong_connection_id`: non-existing connection ID

- *#create_msg*:

36. `create_msg`: positive test
37. `create_msg_wrong_connection_id`: non-existing connection ID
38. `create_msg_wrong_mailbox`: non-existing mailbox name
39. `create_msg_invalid_recipients`: invalid recipient
40. `create_msg_subject_no_string`: invalid subject

- *#move_msg*:

41. `move_msg`: positive test
42. `move_msg_deleted`: positive test for a message with a *Deleted* flag
43. `move_msg_wrong_connection_id`: non-existing connection ID
44. `move_msg_wrong_mailbox`: non-existing mailbox name
45. `move_msg_wrong_message_id`: non-existing message ID

- *#expunge_msgs*:
  46. `expunge_msgs`: positive test
  47. `expunge_msgs_wrong_connection_id`: non-existing connection ID
  48. `expunge_msgs_wrong_mailbox`: non-existing mailbox name
- *#create_mailbox*:
  49. `create_mailbox`: positive test
  50. `create_mailbox_already_exists`: already existing mailbox
  51. `create_mailbox_wrong_connection_id`: non-existing connection ID
  52. `create_mailbox_empty_string` invalid mailbox name
- *#delete_mailbox*:
  53. `delete_mailbox` positive test
  54. `delete_mailbox_wrong_connection_id`: non-existing connection ID
  55. `delete_mailbox_not_existing`: non-existing mailbox name
- *#rename_mailbox*:
  56. `rename_mailbox`: positive test
  57. `rename_mailbox_wrong_connection_id`: non-existing connection ID
  58. `rename_mailbox_not_existing`: non-existing mailbox name
  59. `rename_mailbox_already_exists`: already existing mailbox name
- *#reply*:
  60. `reply`: positive test
  61. `reply_wrong_connection_id`: non-existing connection ID
  62. `reply_wrong_message_id`: non-existing message ID

Bibliography

[AO08] Paul Ammann and Jeff Offutt. *Introduction to Software Testing.* Cambridge University Press, New York, NY, USA, 1 edition, 2008.

[BEFI10] Selen Basol, Ozan Erdem, Michael Fink, and Giovambattista Ianni. HEX programs with action atoms. In *ICLP (Technical Communications)*, volume 7 of *LIPIcs*, pages 24–33. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

[BET11] Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, December 2011.

[Cri03] Mark Crispin. INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1. RFC 3501, March 2003.

[CRL12] Domenico Corapi, Alessandra Russo, and Emil Lupu. Inductive logic programming in answer set programming. In Stephen H. Muggleton, Alireza Tamaddoni-Nezhad, and Francesca A. Lisi, editors, *Inductive Logic Programming*, pages 91–97, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[EFF12] Thomas Eiter, Cristina Feier, and Michael Fink. Simulating production rules using ACTHEX. In *Correct Reasoning - Essays on Logic-Based AI in Honour of Vladimir Lifschitz*, pages 211–228, 2012.

[EFI$^+$16] Thomas Eiter, Michael Fink, Giovambattista Ianno, Thomas Krennwallner, Christoph Redl, and Peter Schüller. A model building framework for answer set programming with external computations. *Theory and Practice of Logic Programming*, 16(4):418?464, 2016.

[EGI$^+$18] Thomas Eiter, Stefano Germano, Giovambattista Ianni, Tobias Kaminski, Christoph Redl, Peter Schüller, and Antonius Weinzierl. The DLVHEX system. *KI*, 32(2-3):187–189, 2018.

[EIK09] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. *Answer Set Programming: A Primer*, pages 40–110. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[EIST05] Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits. A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, pages 90–96, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc.

[EK16] Thomas Eiter and Tobias Kaminski. Exploiting contextual knowledge for hybrid classification of visual objects. In Loizos Michael and Antonis Kakas, editors, *Logics in Artificial Intelligence*, pages 223–239, Cham, 2016. Springer International Publishing.

[EMRS15] Thomas Eiter, Mustafa Mehuljic, Christoph Redl, and Peter Schüller. User guide: dlvhex 2.X. Technical Report INFSYS RR-1843-15-05, Vienna University of Technology, Institute for Information Systems, September 2015.

[FGI$^+$13] Michael Fink, Stefano Germano, Giovambattista Ianni, Christoph Redl, and Peter Schüller. ActHEX: Implementing HEX programs with action atoms. In Pedro Cabalar and Tran Cao Son, editors, *Logic Programming and Nonmonotonic Reasoning*, pages 317–322, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[FLP04] Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. Recursive aggregates in disjunctive logic programs: Semantics and complexity. In Jóse Júlio Alferes and João Leite, editors, *Logics in Artificial Intelligence*, pages 200–212, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[GL88] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert Kowalski, Bowen, and Kenneth, editors, *Proceedings of International Logic Programming Conference and Symposium*, pages 1070–1080. MIT Press, 1988.

[GL91] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3):365–385, Aug 1991.

[GS08] Philip Guenther and Tim Showalter. Sieve: An Email Filtering Language. RFC 5228, January 2008.

[Kow74] Robert A. Kowalski. Predicate logic as programming language. In *IFIP Congress*, pages 569–574, 1974.

[Lif02] Vladimir Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1):39 – 54, 2002. Knowledge Representation and Logic Programming.

[MC08] Ken Murchison and Mark Crispin. Internet Message Access Protocol - SORT and THREAD Extensions. RFC 5256, June 2008.

[MM00] Dianna Mullet and Kevin Mullet. *Managing IMAP*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1st edition, 2000.

[Sch17] Peter Schüller. HEXLite Python-based solver for a fragment of HEX. `https://github.com/hexhex/hexlite`, 2017. Accessed: 2018-07-30.

[Sch18] Peter Schüller. The Hexlite solver - lightweight and efficient evaluation of HEX programs. *Workshop on Trends and Applications of Answer Set Programming*, 2018. to appear.

[Sir02] Timo Sirainen. Dovecot. `https://www.dovecot.org`, 2002. Accessed: 2018-07-25.

[SW18] Torsten Schaub and Stefan Woltran. Answer set programming unleashed! *KI - Künstliche Intelligenz*, 32(2):105–108, Aug 2018.

[Var00] Sam Varshavchik. Courier IMAP. `http://www.courier-mta.org/imap/`, 2000. Accessed: 2018-07-25.