Automatic Repair and Recovery for Omnibase: Robust Extraction of Data from Diverse Web Sources

by

Erica L. Cooper

S.B., Massachusetts Institute of Technology (2009)

Submitted to the Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

June 2010

Author ....
Department of Electrical Engineering and Computer Science
June 1, 2010

Certified by.... ....
Boris Katz
Principal Research Scientist
Thesis Supervisor

Accepted by ....
Christoper J. Terman
Professor of Electrical Engineering
Chairman, Department Committee on Graduate Theses

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

Automatic Repair and Recovery for Omnibase: Robust Extraction of Data from Diverse Web Sources

by

Erica L. Cooper

Submitted to the
Department of Electrical Engineering and Computer Science

June 1, 2010

In partial fulfillment of the requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In order to make the best use of the multitude of diverse, semi-structured sources of data available on the internet, information retrieval systems need to reliably access the data on these different sites in a manner that is robust to changes in format or structure that these sites might undergo. An interface that gives a system uniform, programmatic access to the data on some web site is called a *web wrapper*, and the process of inferring a wrapper for a given website based on a few examples of its pages is known as *wrapper induction*. A challenge of using wrappers for online information extraction arises from the dynamic nature of the web—even the slightest of changes to the format of a web page may be enough to invalidate a wrapper. Thus, it is important to be able to detect when a wrapper no longer extracts the correct information, and also for the system to be able to recover from this type of failure. This thesis demonstrates improved error detection as well as methods of recovery and repair for broken wrappers for START, a natural-language question-answering system developed by Infolab at MIT.

Thesis Supervisor: Boris Katz
Title: Principal Research Scientist

Chapter 1

Introduction

In order to make the best use of the multitude of diverse, semi-structured sources of data available on the internet, information retrieval systems need to reliably access the data on these different sites in a manner that is robust to changes in format or structure that these sites might undergo. An interface that gives a system uniform, programmatic access to the data on some web site is called a *web wrapper*, and the process of inferring a wrapper for a given website based on a few examples of its pages is known as *wrapper induction*. A challenge of using wrappers for online information extraction arises from the dynamic nature of the web—even the slightest of changes to the format of a web page may be enough to invalidate a wrapper. Thus, it is important to be able to detect when a wrapper no longer extracts the correct information, and also for the system to be able to recover from this type of failure.

START [1, 2] is a natural-language question answering system that combines information from over 80 different online sources to answer users' questions. As these sources all have completely different hierarchies and page formats, START accesses them through Omnibase [3], which provides a uniform interface for START by using a collection of wrapper scripts that describe how to find particular pieces of information on the pages of a given website. At present, these scripts are mostly created and repaired manually. This thesis addresses the issue of Omnibase wrapper script failures caused by website format changes, by enabling automatic detection of such errors, and by implementing methods of recovery and repair from them, in order to improve the robustness of START and Omnibase to the changes that web sources often undergo.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 2

Background

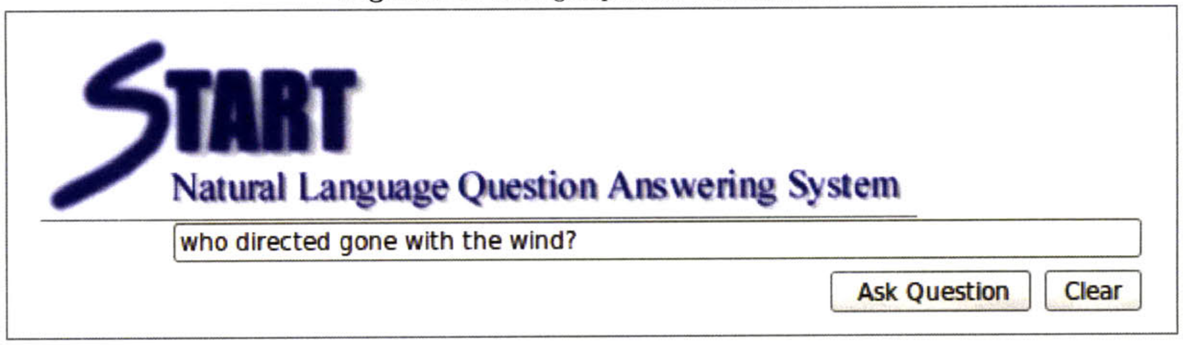
2.1 The START Question-Answering System

START is a natural-language question-answering system developed by the Infolab group at MIT. Users can search for information with START by using the intuitive form of natural language questions. START combines over 80 different online databases of information, including Wikipedia, the Internet Movie Database, the World Factbook, the Nobel Prize website, and many others. Rather than simply returning a list of documents which the user must then click through to try and locate the relevant information, START returns the relevant information itself, in the form of a sentence or a paragraph or whatever is appropriate to the query, thus making it easier for users to quickly find exactly the information they need. Since its first appearance online in 1993, START has answered millions of user queries on a wide variety of topics.

START parses users' questions by breaking them into ternary expressions, which consist of a subject, an object, and a relation between them. Semantic annotations, which are manually-created sentences or phrases that describe the semantic content of the piece of information that they are annotating, are similarly parsed into nested ternary expressions, and START matches ternary expressions for queries against those in its knowledge base to find what knowledge is relevant to the questions. START can access information in various formats, including text and pictures.

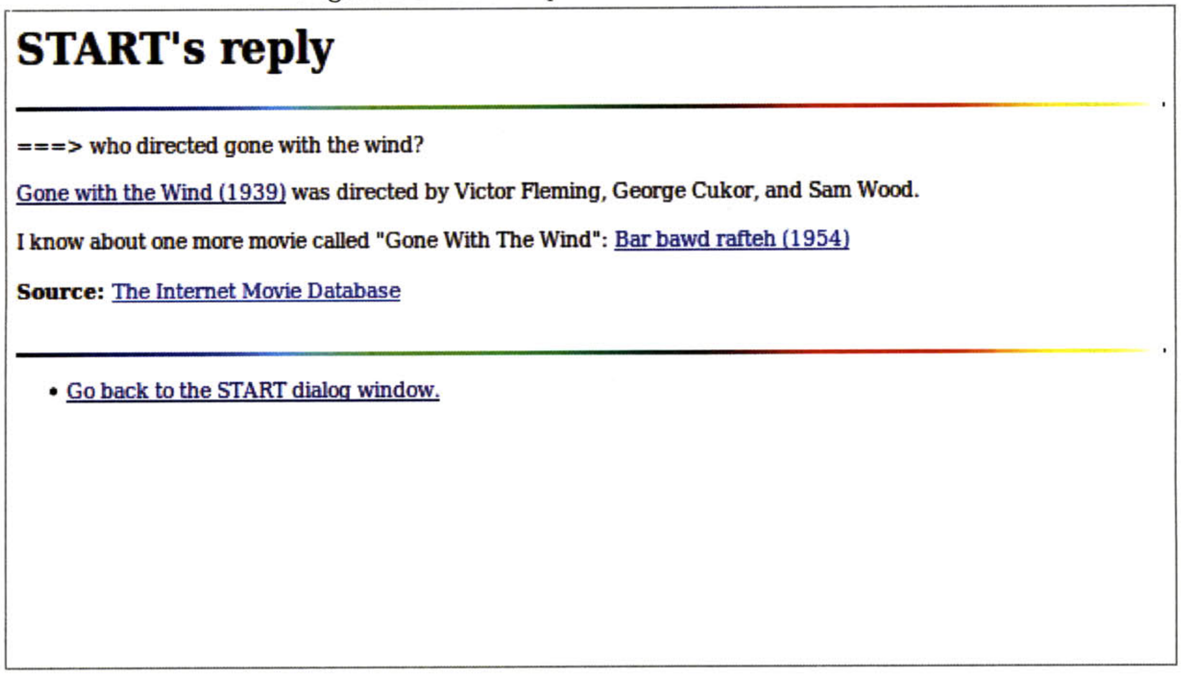
People can ask START questions such as, "What does an emu look like?" and "Who directed Gone with the Wind?". START can handle "nested" questions as well, such as "What is the population of the largest country in South America?", by first determining that the largest country in South America is Brazil, and then by finding out its population.

Figure 2-1: Asking a question with START.



who directed gone with the wind?

Figure 2-2: An example of an answer from START.



START's reply

===> who directed gone with the wind?

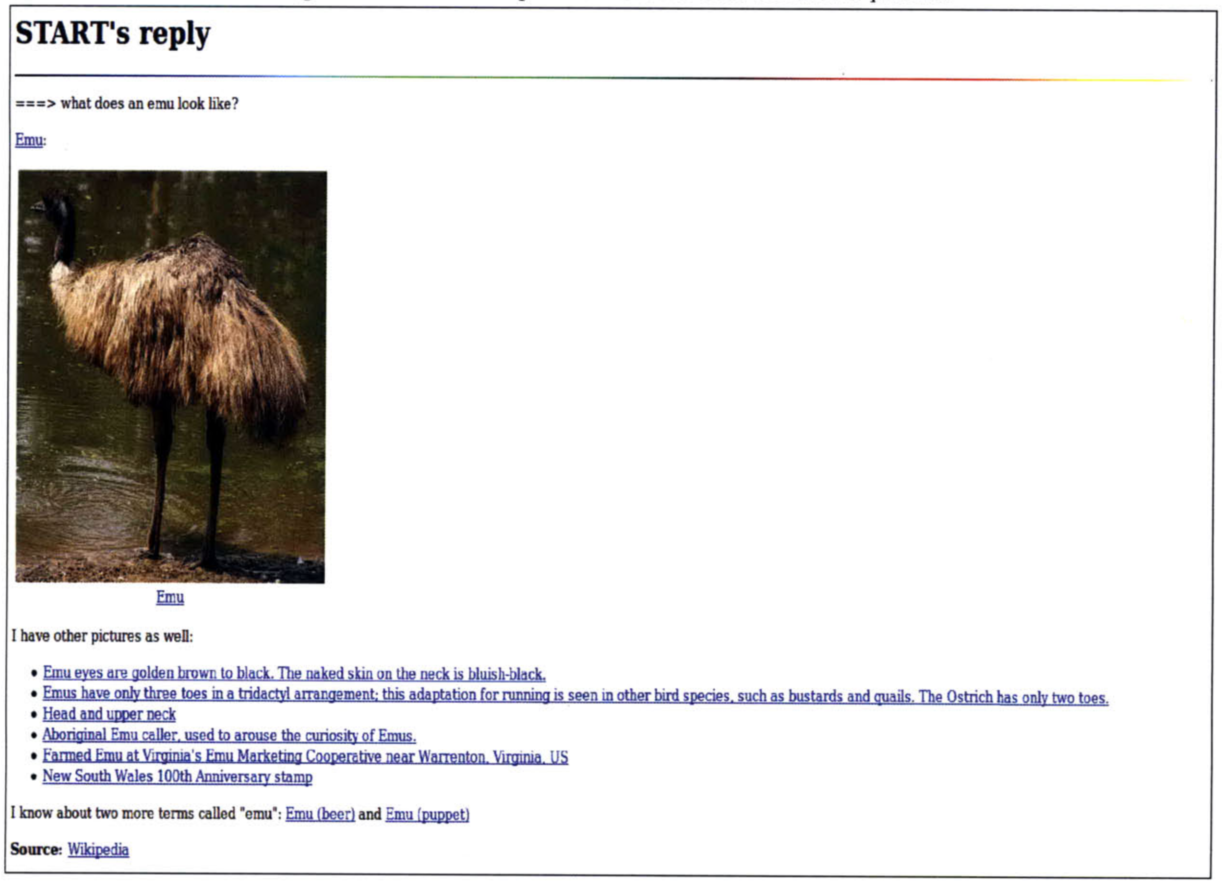
Gone with the Wind (1939) was directed by Victor Fleming, George Cukor, and Sam Wood.

I know about one more movie called "Gone With The Wind": Bar bawd rafteh (1954)

Source: The Internet Movie Database

- Go back to the START dialog window.

Figure 2-3: An example of an answer that includes a picture.



START's reply

===> what does an emu look like?

Emu:

Emu

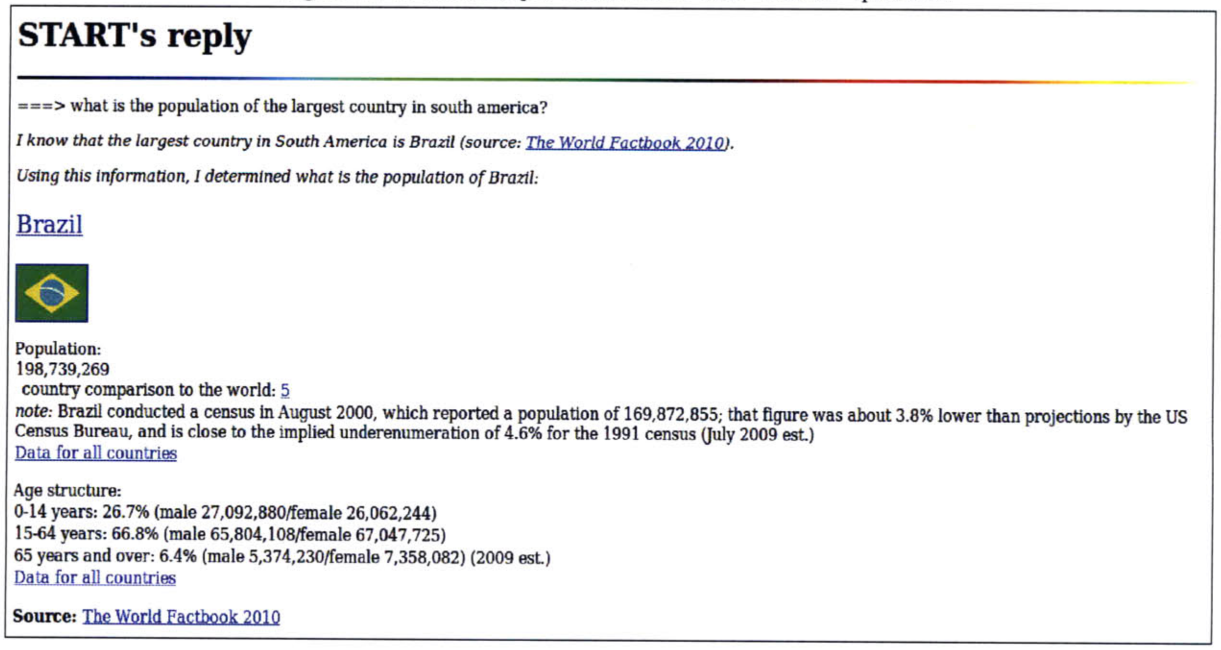
I have other pictures as well:

- Emu eyes are golden brown to black. The naked skin on the neck is bluish-black.
- Emus have only three toes in a tridactyl arrangement; this adaptation for running is seen in other bird species, such as bustards and quails. The Ostrich has only two toes.
- Head and upper neck
- Aboriginal Emu caller, used to arouse the curiosity of Emus.
- Farmed Emu at Virginia's Emu Marketing Cooperative near Warrenton, Virginia, US
- New South Wales 100th Anniversary stamp

I know about two more terms called "emu": Emu (beer) and Emu (puppet)

Source: Wikipedia

Figure 2-4: An example of an answer to a "nested" question.



START's reply

===> what is the population of the largest country in south america?

I know that the largest country in South America is Brazil (source: The World Factbook 2010).

Using this information, I determined what is the population of Brazil:

Brazil

Population:
198,739,269
country comparison to the world: 5
note: Brazil conducted a census in August 2000, which reported a population of 169,872,855; that figure was about 3.8% lower than projections by the US Census Bureau, and is close to the implied underenumeration of 4.6% for the 1991 census (July 2009 est.)
Data for all countries

Age structure:
0-14 years: 26.7% (male 27,092,880/female 26,062,244)
15-64 years: 66.8% (male 65,804,108/female 67,047,725)
65 years and over: 6.4% (male 5,374,230/female 7,358,082) (2009 est.)
Data for all countries

Source: The World Factbook 2010

2.2 Omnibase

Omnibase [3] is the interface between START and the internet, providing the ability to extract relevant information from the different web sources. Omnibase consists of a collection of wrapper scripts that extract this data and put it into a consistent format so that it can be accessed by START, using an Object-Property-Value data model. The data from websites is turned into relations of objects (also known as symbols), properties of those objects, and the specific values the properties take on—for example, an Omnibase query would be made with the `get` function, taking the form `(get "NOBEL-PERSON" "Albert Einstein" "BIOGRAPHY")`, which would look on the Nobel Prize website (http://www.nobelprize.org) to retrieve a paragraph about Einstein's life. The `BIOGRAPHY` script looks like this:

```
# nobel-person BIOGRAPHY

(lambda (symbol)
  (let ((bio-code (get-local "nobel-person" symbol "BIO-CODE")))
    (if (string? bio-code)
        (match-between
         "<p>"
         "<p class=\"smalltext\">"
         (get-page
          (string-append
           (get "nobel-person" symbol "URL")
           (string-append
            bio-code
            ".html"))
          ':cache-policy 'prefer_live
          ':cache-tag "nobel"))
        (throw 'attribute-value-not-found
               ':reply (string-append symbol " does not have a biography on the Nobel site.")))))
```

This script uses a mix of "local data" (the code for Albert Einstein's page on the Nobel Prize website, which is a number stored locally in the database), regular expressions to locate the biography on the webpage, and other scripts, namely the `URL` script which is another attribute for the `nobel-person` class.

As of now, the scripts are written by hand for each new web source that is added, and, since they are specific to the format of the source, they must also be corrected by hand if the format of the website changes enough to cause the script to no longer work. In addition, when a script fails in that way, the failure is not immediately detected—one would detect such an error either by asking START a question and seeing it respond with an error, or by looking at the START logs, which show which questions users asked and the responses they got back from START. Some errors are currently also detected by an automatic test suite, but the tests do not always take into account all of the ways in which the format of a response can change, but still be correct, or match the test, yet still be wrong.

2.3 Wrapster

Wrapster [4] facilitates the creation of wrapper scripts for finding the attribute values for a particular symbol. Wrapster automatically creates a set of wrapper scripts for a web source. It takes as input a few examples of pages from the web source that are associated with a few representative symbols. It then generates a wrapper by separating the pages into regions, and then clustering similar regions across the different example pages to determine which regions these pages have in common—these are likely choices for attributes. The user can use the Wrapster web interface to edit the wrappers by giving the regions more meaningful names and by fine-tuning the automatically-generated regular expressions for extracting attributes. The web interface also has the capability to interactively test a wrapper that a user is editing—a user can select an attribute and a symbol, see the value that the wrapper extracts, and decide whether that value is correct or the wrapper requires further editing. Finally, when the user has decided that the wrapper is finished, he or she can import the scripts into Omnibase, which will also automatically create test cases for the wrapper from the examples that the wrapper was initially generated from.

2.4 Hap-Shu

Hap-Shu [5] is a language that provides abstractions for creating wrapper scripts. Rather than specifying particular regular expressions for locating attributes, a user can instead use Hap-Shu to locate titles on a webpage, and name the title associated with each particular attribute as the means of extracting the value for that attribute. Hap-Shu scripts are more robust to format changes than regular expression-based scripts because it can find elements on the page that are titles, regardless of their format. Hap-Shu is particularly useful for highly-structured pages that have titles for each section of information.

2.5 Web Cache

Omnibase has a web caching library for storing the live-fetched text of webpages locally on the filesystem. This has the advantage of making it faster to retrieve pages that are requested frequently, as well as providing a fail-safe for network connection problems or website downtime. The caching library was designed to be extensible; it can also be used to cache the results of computations that are not necessarily live fetches of pages from the web.

2.5.1 Safeurl

`safeurl` is the cache library's abstraction for getting the HTML of a page, either from the web or from the local cache, depending upon the cache policy. It has some conveniences built in, such as the ability to specify a random delay between live fetches to avoid spamming some site with many requests all at once.

2.5.2 Cache Policies

Depending on the type of information on the pages being cached, a user might want to use the cache in different ways. We implemented four different cache policies.

Prefer-Live

The `'prefer_live` cache policy option says that when a user requests a page, a live fetch will be attempted first, and if that succeeds, then the result of the live fetch will be returned to the user, and the cache will be updated with this newest version of the page. Otherwise, if there was any error fetching the page live (i.e., if there was a connection failure or if the site is down), the current cached version of the page will be returned to the user and no change will be made to the cache.

Prefer-Local

The `'prefer_local` option means that the cache will attempt to return to the user the locally-cached copy of the page that he or she requested. If there is no cached version of the page, or if the cached page is older than a specified age, then a live fetch is attempted and that text is both returned to the user and placed into the cache. This option is useful for pages that have information that is not expected to change much, like a page with biographical information about a person or geographical information about a country. `'prefer_local` is the default refresh policy.

Force-Live

The `'force_live` option is a way for users to have access to the niceties of the `safeurl` abstraction described above, without actually caching anything. With this option, a live fetch is attempted, and if it fails, the user receives an error. Nothing is cached with this option.

Force-Local

The `'force_local` option requires that the cached version of the page be returned to the user. For example, this might be used for a site that has been cached but that no longer exists. If a page requested with this option is not in the cache, an error is thrown.

2.5.3 Get-Page

`get-page` is Omnibase's abstraction around the caching library. There are options to not use a cache at all, to use a particular cache, or to use the main cache for Omnibase.

2.6 Types of Wrapper Failure

The following are various causes of wrapper failure or wrapper test case failure.

2.6.1 Changes to Page Layout

Often, websites undergo a total redesign, and, for instance, where once we expected to find the birth date of a Nobel laureate in boldface as the first item in a table, we now find it in a `div` element, surrounded by some new and different context. Thus, a wrapper script expecting to be able to extract attributes by looking for certain regular expressions will not be able to find them, and will throw an error. Alternately, the script might coincidentally still be able to locate a match for the regular expressions, but since the page has changed so much, the match will either be the wrong piece of information, or complete nonsense.

Writing scripts using less-specific regular expressions to start with would not solve the problem, for two reasons: first, there would be the risk of overgeneralizing and finding matches that are not the property that should be extracted, and second, it would be extremely difficult to predict all of the possible ways in which the format of the site might change, so scripts would still break because of the unpredictability of these changes.

This thesis will focus on errors that occur due to changes in page layout.

2.6.2 Changes to the Format of Attribute Values

In contrast to a complete redesign of a website, a comparatively small change that can occur is the change in the format of an attribute value—for example, perhaps the maintainers of the site decided they would rather display the date "January 12, 1998" as "1/12/1998." If the surrounding context remains unchanged, then the wrapper script for this attribute should still be able to extract this correct value—however, given that the test cases for this script expect to match the value returned by the script at the time it was created, these tests will falsely signal a failure, and someone will have to correct the test cases to make them more general.

2.6.3 Changes to the Values of Attributes

The values of certain attributes will inherently change frequently, and this leads to similar problems as a change in the format of an attribute. Some examples of attributes that can be expected to change often include the latest headlines about a particular company, the weather at a particular location, the price of a stock, and the expected time to wait at a particular airport terminal. These perfectly valid changes in the value of some attribute can cause tests to fail incorrectly as well.

Chapter 3

Detection, Recovery and Repair

Addressing the problem of broken wrapper scripts required breaking up the problem into three parts: detecting when such errors occur, recovering the old versions of webpages so that the wrapper scripts could continue to work temporarily, and a more permanent repair process to generate new, working scripts for the newly-formatted web pages.

3.1 Detection

Before I implemented error detection and notification, Infolab would only learn of START or Omnibase errors when test cases would fail, or when users of START would report errors they got when asking questions. As such, many errors would go undetected if no one reported them and they were not covered by the test cases. In order to determine the types of errors that are the most common, and the best way to fix them, some form of notification when errors happen is needed. I placed email notifications in various parts of Omnibase to report different kinds of errors.

3.1.1 Cache Emails

I added an `'email` option to the web caching library so that the user could specify email addresses to notify in the event of a cache failure. The cache library sends an email when a `'prefer_live` or `'force_live` cache tries to fetch a page live, and fails, thus notifying Infolab when there are problems accessing a site that START relies on. Furthermore, an email is sent when the cache library tries to find a local version of a page but can't find it in the cache, which may indicate an earlier failure to index and cache a site fully, or a broken script that requests an incorrect URL.

3.1.2 Omnibase Emails

The `get-page` function in Omnibase designates a default email address to pass on to the cache it creates. Also, in the `get` function, any time a script produces an error, the error is caught and an email is sent about it, and the error is then re-thrown.

3.1.3 Class Emails

I also added the ability to have any particular Omnibase class send emails about its errors. This is because an Omnibase developer who has just added a new class might want to monitor it to determine if the class is working properly. This developer can tag the class with his or her own email address in order to receive emails about errors produced in that specific class. This is implemented with a new kind of meta-script called `email` that the developer can put in his or her class as follows:

```
# mapquest-directions email

'("ecooper@mit.edu")
```

The addresses listed in this script are added to the default addresses to mail when an Omnibase script fails, when the script that failed came from the particular class that is being monitored.

3.1.4 Effectiveness and Improvements

We found that Omnibase sent many emails about different kinds of errors. Very few emails were sent from the cache library about live fetch errors for pages that were down, and there were a few emails about failed local retrieval from the cache. However, there were many emails about script failures. Some of these brought our attention to scripts that hadn't been implemented and to errors in scripts that we could fix. However, some of the errors were "expected" errors, that is, the failure of Omnibase to find some attribute that isn't always there. For example, an IMDb page about a movie might have information about which awards a movie won, so we might want to have an `AWARDS` attribute for IMDb movies. However, we wouldn't necessarily require every page for a movie to list awards, since not every movie has won awards, and IMDb has chosen to omit the "Awards" section entirely for those movies' pages. Attributes like these can be considered "optional," and we might not want to receive error emails about not being able to find them.

Some errors contain a `'reply` tag; this indicates that an error is about an attribute that is expected to be missing sometimes. I modified the emailer in Omnibase to ignore errors with a `'reply`, since these errors can be considered "expected." While all errors about attributes considered "optional" should contain a `'reply`, it is often too difficult to tell the difference between failure to locate an attribute because it's just not available on the page, and failure to locate an attribute because of broken or inadequate regular expressions. Future improvements may include adding metadata to the wrapper scripts to indicate whether or not they are optional.

Furthermore, many error emails are sent each day, often with multiple emails being sent about the same class. To keep redundant emails from being sent, I created a list for Omnibase to keep track of classes, symbols, and attributes that developers do not want to get further emails about, as well as a utility that allows users to add to this list. The user can specify whether the error happened because the attribute was "optional" or because of a broken script, and also, he or she can have the hold on emails apply to errors of some script for a particular symbol, or for all of the symbols in the class. Another option for future improvement would be to send out one batch email daily or weekly about all the errors, either for all of the classes, or for each class individually.

3.2 Recovery

If a site's format has changed and the wrapper scripts for that site no longer work, in some cases it would be useful to "recover" the old versions of pages from that site and get answers from there instead. Often, an "out of date" answer is better than no answer at all, especially for attributes that don't change often or at all.

Since we already cache webpages in case of connection failure, it follows that we would also be able to use the cached webpages in the case of script failure, i.e., for recovery. I changed the caching library to keep it from necessarily caching pages immediately after they were live-fetched, and made changes to Omnibase, to try scripts again on the old cached version of a page when they fail on the live version.

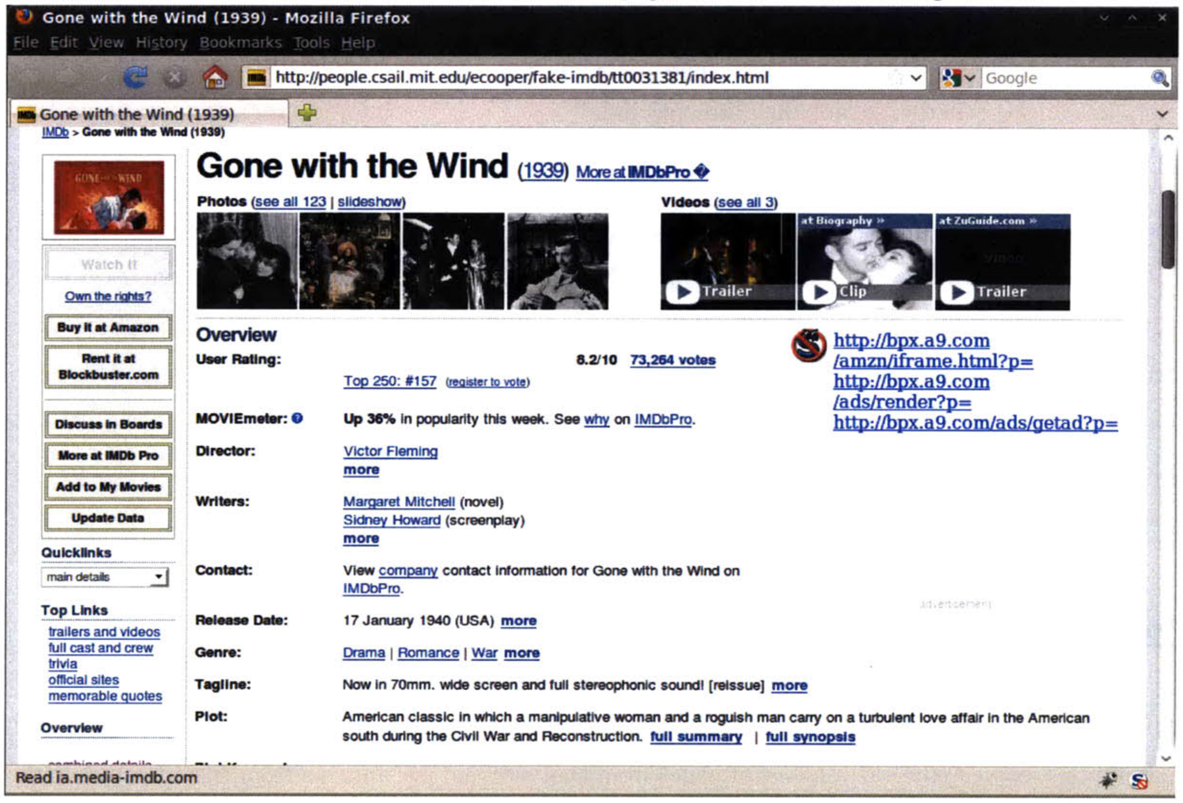
The process for recovery with scripts that use cache option `'prefer_live` works as follows:

- Try the script, using live-fetched pages. Do not store the live-fetched pages text back in the cache yet.

- If the script succeeds, write the live-fetched pages back to the cache.

- If any part of the script fails, discard the live-fetched pages and retry the script, using cache option `'prefer_local`. If there is no local version of the page and the live version is fetched instead, then it gets written to the cache.

Figure 3-1: An example fake IMDb page, before the format changes.



Gone with the Wind (1939) - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://people.csail.mit.edu/ecooper/fake-imdb/tt0031381/index.html

Gone with the Wind (1939)

IMDb > Gone with the Wind (1939)

Gone with the Wind (1939) More at IMDbPro

Photos (see all 123 | slideshow)

Videos (see all 3)

Overview

User Rating: 8.2/10 73,264 votes

Top 250: #157 (register to vote)

MOVIEmeter: Up 36% in popularity this week. See why on IMDbPro.

Director: Victor Fleming more

Writers: Margaret Mitchell (novel) Sidney Howard (screenplay) more

Contact: View company contact information for Gone with the Wind on IMDbPro.

Release Date: 17 January 1940 (USA) more

Genre: Drama | Romance | War more

Tagline: Now in 70mm. wide screen and full stereophonic sound! [reissue] more

Plot: American classic in which a manipulative woman and a roguish man carry on a turbulent love affair in the American south during the Civil War and Reconstruction. full summary | full synopsis

http://bpx.a9.com/amzn/iframe.html?p=
http://bpx.a9.com/ads/render?p=
http://bpx.a9.com/ads/getad?p=

Read ia.media-imdb.com

3.2.1 Testing and Evaluation

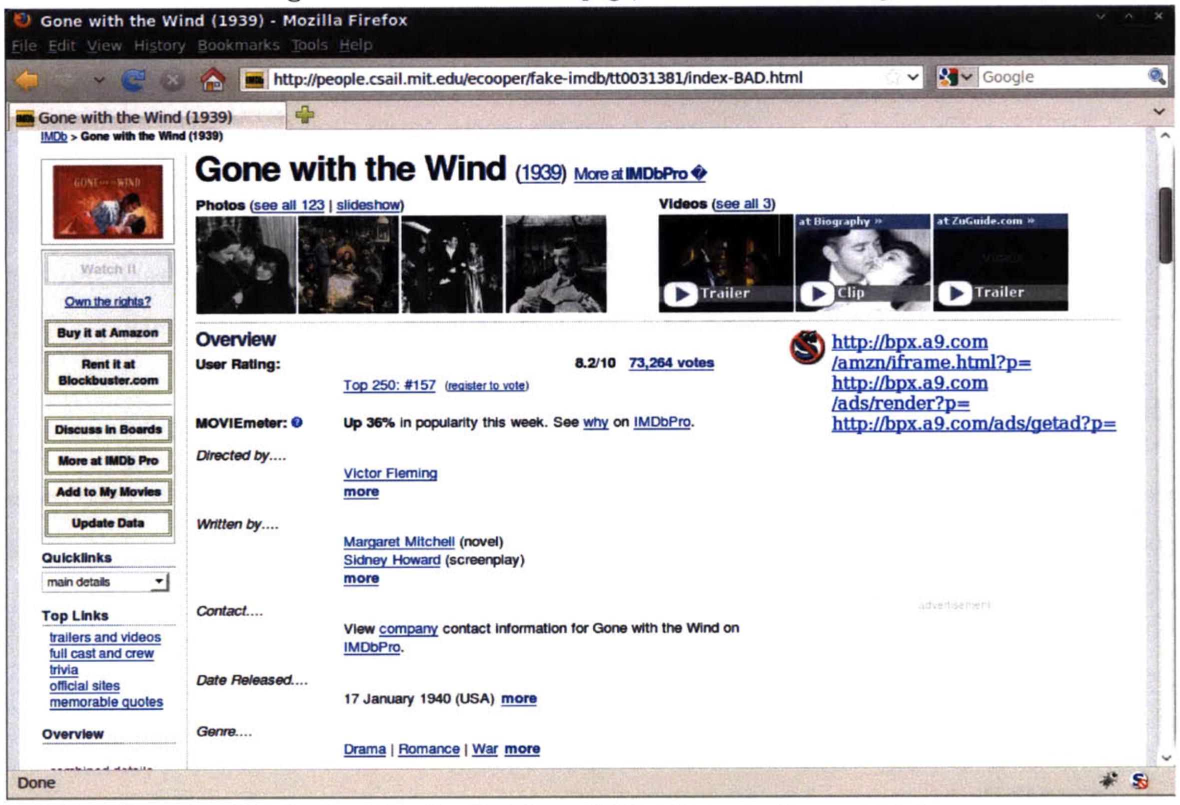
Testing recovery required having a website that could be changed to emulate format changes. I created some local copies of IMDb pages for movies, and a corresponding `fake-imdb` class that had scripts that would point to these pages instead of the real IMDb pages. I also made versions of these pages with format changes that would break the scripts, in particular, the boldface section titles such as **Director:** were changed to be in italic, with slightly different wording, such as *Directed by.....* Since the IMDb scripts utilize Hap-Shu abstractions, the format change alone does not break the scripts, but the change in wording will break them since the title keywords are no longer the same.

3.2.2 Implementation

Changes to the Cache Library

Since writing a live-fetched page back to the cache was behavior inherently built into fetching a page live using the cache library, this had to be changed. I added a `'no_overwrite` option to caches which by default is false, but if it is specified as true, then live fetches using that cache do not automatically overwrite the cached version of the page.

Figure 3-2: The fake IMDb page, after the format changes.



Gone with the Wind (1939) - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://people.csail.mit.edu/ecooper/fake-imdb/tt0031381/index-BAD.html

IMDb > Gone with the Wind (1939)

Gone with the Wind (1939) More at IMDbPro

Photos (see all 123 | slideshow)

Videos (see all 3)

Watch It

Own the rights?

Buy it at Amazon

Rent it at Blockbuster.com

Discuss in Boards

More at IMDb Pro

Add to My Movies

Update Data

Quicklinks

main details

Top Links

trailers and videos
full cast and crew
trivia
official sites
memorable quotes

Overview

Overview

User Rating: 8.2/10 73,264 votes

Top 250: #157 (register to vote)

MOVIEmeter: Up 36% in popularity this week. See why on IMDbPro.

Directed by....

Victor Fleming
more

Written by....

Margaret Mitchell (novel)
Sidney Howard (screenplay)
more

Contact....

View company contact information for Gone with the Wind on IMDbPro.

Date Released....

17 January 1940 (USA) more

Genre....

Drama | Romance | War more

http://bpx.a9.com
/amzn/iframe.html?p=
http://bpx.a9.com
/ads/render?p=
http://bpx.a9.com/ads/getad?p=

Done

Since cache writes no longer necessarily happen automatically, we needed a way for the user to write back to the cache. Since writing back to the cache might mean different things depending upon what the cache is being used for, I made a writer that was specific to the `safeurl` use of the cache and called it `safeurl-write`. This function just abstracts out the already-existing write-back functionality that used to happen automatically into its own function, and makes it a part of the cache library that a user can access.

Changes to Omnibase

Some global state needed to be introduced in order to implement recovery. The first global variable is `RECOVERY_MODE`, which is a boolean indicator of whether or not Omnibase is currently trying to re-run scripts on the cached versions of pages. Another global variable is the integer `SCRIPT_DEPTH`. The reason for the `SCRIPT_DEPTH` global variable is that very often, wrapper scripts are nested (many scripts call other scripts), and we only want to declare success and write back pages to the cache when the top-level script has succeeded. The last new global variable for recovery is `GLOBAL_CACHE_LIST`. This contains information about each page that was fetched live, so we can write those pages back to the cache once the top-level script has succeeded. This list contains tuples of three items each: the HTML of the page that was fetched live, the URL it came from, and a pointer to the cache object that should be used to write the page to the filesystem. The related function `write-pages` iterates over each of those tuples, writing each page back to its respective cache.

In Omnibase, `get-page`'s cache was made to use the `no_overwrite` option. `get-page` also checks to see if `RECOVERY_MODE` is true, in which case it also uses cache option `'prefer_local`. I modified the `get` function to try the designated script once normally, and catch any errors. The error handler sets `RECOVERY_MODE` to true, throws away any live-fetched pages in the `GLOBAL_CACHE_LIST`, and tries the designated script again in recovery mode. If the second try fails, the error is thrown as usual. An email is sent when a script has to enter recovery mode, and when recovery fails for a script as well.

3.2.3 Limitations of Recovery

Recovery doesn't fix broken scripts; it just buys some time to fix them. Furthermore, recovery makes Omnibase return outdated values, so it's only really useful for values that don't change often anyway. Another limitation is that if a format change breaks all but one script, if the script that still works is run, then the new version of the page will get cached and then recovery won't work at all for the rest of the scripts that are now broken.

3.3 Repair

While recovery provides a temporary solution to broken scripts, they will eventually need to be repaired. Currently they are repaired by rewriting them by hand, but a way to automatically regenerate the wrapper scripts would be better.

There are currently two possible approaches to repair. The first is to integrate Wrapster with Omnibase to completely and automatically rewrite the scripts using a set of representative symbols. The second is to use Hap-Shu's ability to find keywords and their surrounding HTML, except instead of finding keywords that are titles of sections, use it to find known attribute values as the keywords instead, and use the surrounding HTML in the new wrapper script.

3.3.1 Regenerating Wrapper Scripts with Wrapster

Wrapster takes a few representative examples of symbol pages, determines the regions on these pages, and generates an XML wrapper for the website based on the examples. In order to integrate this with Omnibase, we needed a way to tell Wrapster which symbols were good representative examples, and also how to turn the XML wrappers into Omnibase-style wrappers, which are written in the Guile dialect of Scheme.

The writer of scripts for a new class must create certain data files, including a list of all the names of symbols (called *classname*-`symbols`), a file called *classname*-`local-data`, which contains information including a mapping from the symbols to the unique codes for those symbols used by the website (if any), and a file with the scripts themselves. I introduced a new type of file, *classname*-`representative`, in which the developer of a new class should list about five symbols that he or she determines to be a good representative sample. More than one representative symbol is needed because some symbols are missing some attributes, and sometimes the formatting may be slightly different across symbols. For example, for a site that has biographies, a good representative sample would be one living person, one deceased person, and one person who lived in years B.C. During repair, we would want to capture these variations in order to have the right amount of generality in the new scripts. Given a set of representative symbols, Wrapster can generate a new wrapper for that class.

Once a new wrapper is generated, it needs to be converted from XML into the Guile Scheme format used by Omnibase. Since both the XML and Scheme wrappers represent the same information, I implemented this conversion with a script that turns the regions specified in the XML wrapper into the corresponding Scheme code with regular expressions for an Omnibase wrapper script. Since the XML wrapper also contains information about what values it got for each of the attributes, the conversion script can also automatically generate test cases for the Scheme wrapper scripts.

3.3.2 Using Hap-Shu for Repair

While Wrapster can produce completely new scripts for a site, this method does not take into account that we already had an existing wrapper for the site that worked at some point in the past, and that we had access to the correct attribute values when the scripts were working. Since Hap-Shu can find titles and their HTML context given keywords for the titles, it could also be used "backwards" by being given expected values for attributes and using the same mechanisms for finding their HTML context.

Answers Cache

To perform repair with known attribute values, we need some knowledge about what the expected values are for some symbols' attributes. With this in mind, I chose to create an answers cache. Originally, the plan was to extend the existing cache library to store the results of wrapper scripts on the filesystem as people asked questions on START, but it was determined that the coverage of questions that people asked was too sparse for all of the scripts to get called on each of the representative symbols we chose for a class. Omnibase stores its necessary data in a backend implemented as an SQL database containing various tables. Instead of implementing the answers cache using the cache library, I implemented it with a table in this database called `answers_cache`, with columns for `class, symbol, attribute,` and `value`. When the author of a class creates the *classname*-`representative` file, he or she can run the new `put-representative` utility, which gets the attribute values for all of those symbols, and writes them into the `answers_cache` table. Once this happens, there exist representative answers for that class, and even if the format of the website changes, if the attributes can be located on the new pages for these representative symbols, then their context can be inferred and new wrapper scripts can be generated.

Finding Titles from Known Answers

Hap-Shu has a command called `all-till-next-title`, which, given some text that should appear in the title of a "section" of the page, locates that text on the page, determines how it is formatted, and gets the text that follows it, up until the next "section" title, which should be formatted in the same way. "Sections" are any logical parts of a page, such as an HTML heading element (`<h1>` through `<h7>`), along with any following text up to the next heading of an equal or higher level, or a bold face span at the beginning of a larger element along with the rest of that element, etc. The `all-till-next-title` command makes use of a helper function called `next-HTML-node` which, given some piece of a webpage, returns the first node, which is a piece of HTML from a tag to its associated end tag. After this function sees the first tag that opens the node, it must recursively match any additional opening tags it sees, in order to avoid mismatching the opening tag for that node.

In order to look "backwards" to find the title associated with some information, a similar `previous-HTML-node` function was needed. Since regular expressions in Guile can't search backwards, I implemented this in the following manner:

- Create a list of the locations of all of the HTML tags on the given portion of the page, using the rules for finding HTML tags in Hap-Shu. Both opening and closing tags are listed. As the list is built, new tags are added to the front of the list, so the list represents the tags in the document in reverse.
- Find the last closing tag in the portion of the page by finding the first closing tag in the list.
- Find the opening tag that matches the closing tag found. Closing tags found before the correct opening tag must be recursively matched as in `next-HTML-node`.
- The HTML node returned is everything from the start of the opening tag that was matched in the previous step, to the end of the closing tag that was found in the first step.

I also added a `find-associated-title` command with Hap-Shu. Given a known attribute value, it locates the best match for that value on the page. The best match is some region of text between HTML tags that contains the given string, and as few other characters as possible. This function finds the match by reusing the code that finds a match for a title in `all-till-next-title`, except instead of searching for a title string, it is searching for an attribute value. Once a match for the value is found, `previous-HTML-node` is called to find the title for the section that contains the found attribute value, and the title text is obtained by removing HTML tags, punctuation, and extra spaces from that node. The new, repaired script can be made by using the newly-found title with Hap-Shu's `all-till-next-title` command.

Limitations of This Method of Repair

Currently, Hap-Shu will only find the node containing the cached answer string if that exact string can be found somewhere on the page. This is problematic when the answers themselves change format, even if the values of the answers haven't changed. For example, the formatting of a date might change, or in the case of IMDb's `GENRE` attribute, which lists the genres of a film, the order of the list might change. In both of these cases, Hap-Shu will fail to find the attribute value.

Similarly, the values of some attributes will inherently change frequently, and this causes problems with this method of repair. Some examples of attributes that can be expected to change often include the latest headlines about a particular company, the weather at a particular location, the price of a stock, and the expected time to wait at a particular airport terminal. These perfectly valid changes in the value of some attribute will mean that the cached answer won't be found on the page and repair using Hap-Shu will fail. Using semantic information in repair would also be useful in this case—for example, it could be specified that an attribute value should be a monetary value or a number of minutes within a certain range.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 4

Conclusions

4.1 Contributions

This thesis achieved its stated goals of improving error detection for Omnibase, utilizing cached webpages to recover from wrapper script failures due to website format changes, and implementing methods of wrapper script repair. Specific contributions were:

- Creating email notification for cache and wrapper script failures, as well as control for these notifications so that users can specify whether they want to keep receiving emails about particular errors.
- Implementing a recovery system that lets Omnibase back off to older, cached versions of pages when scripts fail due to format changes.
- Implementing the conversion from Wrapster-style XML scripts to Omnibase-style Guile Scheme scripts, so that Wrapster could be used to regenerate scripts for Omnibase.
- Modifying Hap-Shu's HTML parser to work in reverse, so that the section title associated with a known attribute value can be found and broken scripts can be repaired.

These contributions allow Infolab to learn about errors within Omnibase more quickly, and provide some automatic mechanisms for Omnibase to be more robust to these errors.

4.2 Future Work

While error detection, recovery, and repair were implemented, there is room for improvement:

- Now that the repair infrastructure is in place, more different types of repair could be experimented with and improved; for example, repair by training a system to find the optimal regular expressions to use in wrapper scripts that are neither too specific nor too general, or repair using descriminative training to distinguish between useful information on a page and boilerplate or advertising.
- Currently, repair with Hap-Shu looks for the exact attribute value to find a context for generating the new wrapper. This method could be extended to handle repair in the case where the answer does not stay exactly the same (for example, stock prices or weather reports) or when the answer stays the same but its format has changed (for example, if a date is formatted differently), by looking for semantic types of the values, rather than the exact values themselves.
- Further improvements to the email notification system could be made, such as having daily or weekly batch emails for errors.

Bibliography

[1] Boris Katz, Gary Borchardt and Sue Felshin. Natural Language Annotations for Question Answering. Proceedings of the 19th International FLAIRS Conference (FLAIRS 2006). May 2006.

[2] Boris Katz. Annotating the World Wide Web Using Natural Language. Proceedings of the 5th RIAO Conference on Computer Assisted Information Searching on the Internet (RIAO '97). 1997.

[3] Boris Katz, Sue Felshin, Deniz Yuret, Ali Ibrahim, Jimmy Lin, Gregory Marton, Alton Jerome McFarland and Baris Temelkuran. Omnibase: Uniform Access to Heterogeneous Data for Question Answering. Proceedings of the 7th International Workshop on Applications of Natural Language to Information Systems (NLDB 2002). June 2002.

[4] Gabriel Zaccak. Wrapster: Semi-automatic wrapper generation for semi-structured websites. Master's Thesis, Massachusetts Institute of Technology, 2007.

[5] Baris Temelkuran. Hap-Shu: A Language for Locating Information in HTML Documents. M.Eng. Thesis, Massachusetts Institute of Technology, 2003.