

The Journal of Computing Sciences in Colleges

Papers of the 18th Annual CCSC
Mid-South Conference

April 17-18, 2020
Lyon College
Batesville, AR

Baochuan Lu, Editor
Southwest Baptist University

David Naugler, Regional Editor
Southeast Missouri State University

Volume 35, Number 9

April 2020

The Journal of Computing Sciences in Colleges (ISSN 1937-4771 print, 1937-4763 digital) is published at least six times per year and constitutes the refereed papers of regional conferences sponsored by the Consortium for Computing Sciences in Colleges.

Copyright ©2020 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

Table of Contents

The Consortium for Computing Sciences in Colleges Board of Directors	5
CCSC National Partners	7
Welcome to the 2020 CCSC Mid-South Conference	8
Regional Committees — 2020 CCSC Mid-South Region	9
Reviewers — 2020 CCSC Mid-South Conference	10
Design of a Cyber Security Awareness Campaign to be Implemented in a Quarantine Laboratory	11
<i>Tanim Sardar, Luay A. Wahsheh, Arkansas Tech University</i>	
A Scalable, Hybrid Entity Resolution Process for Unstandardized Entity References	19
<i>Awaad Al Sarkhi, John R. Talburt, University of Arkansas at Little Rock</i>	
Exploration of Factors Contributing to Academic Success in a Data Analytics Program	30
<i>Matt Brown, Arkansas Tech University</i>	
A Programming Workshop Course to Complement CS 1/2	37
<i>Christopher A. Healy, Furman University</i>	
Why Teach Operating Systems?	44
<i>James W. McGuffee, Christian Brothers University</i>	
Adding a Syntax Macro Facility to iGen	52
<i>Larry Morell, Arkansas Tech University, Xin Wan, Microsoft Corporation</i>	
A New Face for Old Moses: An Exercise in Swift and C Interoperability	60
<i>Robert England, Transylvania University</i>	
Resources for Building Knowledge Base Files for Use With Named Entity Resolution/Disambiguation Tools Such As TIMBER	69
<i>Anthony D. Davis, Lyon College</i>	

A Unified Representation for Teaching Bottom-up and Top-down Parsing 78
Larry Morell, David Middleton, Arkansas Tech University

Toward the Creation of a Personal Device Security Testbed to Aid Student Learning Objectives 86
Charles Walter, Charles Fleming, University of Mississippi

Engaging Students with Computing for the Common Good — Conference Panel 95
James W. McGuffee, Christian Brothers University, Anthony Davis, Lyon College, Mark Goadrich, Hendrix College

Abstract Syntax BNF Is Not Ambiguous/Inadequate — Nifty Assignment 97
Cong-Cong Xing, Nicholls State University, Jun Huang, Chongqing Univ. of Posts and Telecommunications

Increasing Cyber Security Awareness by Creating a Case Study and Video Project — Nifty Assignment 99
Luay A. Wahsheh, Arkansas Tech University

The State of Machine Learning — Conference Tutorial 102
Dan Brandon, Christian Brothers University

Cyber Security Hands-On Learning Using Steganography — Conference Tutorial 104
Luay A. Wahsheh, Arkansas Tech University

Introduction to Jetstream - A Research and Education Cloud — Conference Tutorial 106
Sanjana Sudarshan, Jeremy Fischer, Indiana University

The Consortium for Computing Sciences in Colleges Board of Directors

Following is a listing of the contact information for the members of the Board of Directors and the Officers of the Consortium for Computing Sciences in Colleges (along with the years of expiration of their terms), as well as members serving CCSC:

Jeff Lehman, President (2020), (260)359-4209, jlehman@huntington.edu, Mathematics and Computer Science Department, Huntington University, 2303 College Avenue, Huntington, IN 46750.

Karina Assiter, Vice President (2020), (802)387-7112, karinaassiter@landmark.edu.

Baochuan Lu, Publications Chair (2021), (417)328-1676, blu@sbuniv.edu, Southwest Baptist University - Department of Computer and Information Sciences, 1600 University Ave., Bolivar, MO 65613.

Brian Hare, Treasurer (2020), (816)235-2362, hareb@umkc.edu, University of Missouri-Kansas City, School of Computing & Engineering, 450E Flarsheim Hall, 5110 Rockhill Rd., Kansas City MO 64110.

Judy Mullins, Central Plains Representative (2020), Associate Treasurer, (816)390-4386, mullinsj@umkc.edu, School of Computing and Engineering, 5110 Rockhill Road, 546 Flarsheim Hall, University of Missouri - Kansas City, Kansas City, MO 64110.

John Wright, Eastern Representative (2020), (814)641-3592, wrightj@juniata.edu, Juniata College, 1700 Moore Street, Brumbaugh Academic Center, Huntingdon, PA 16652.

David R. Naugler, Midsouth Representative (2022), (317) 456-2125, dnaugler@semo.edu, 5293 Green Hills Drive, Brownsburg IN 46112.

Lawrence D'Antonio, Northeastern Representative (2022), (201)684-7714, ldant@ramapo.edu, Computer Science Department, Ramapo College of New Jersey, Mahwah, NJ 07430.

Cathy Bareiss, Midwest Representative (2020), cbareiss@olivet.edu, Olivet Nazarene University, Bourbonnais, IL 60914.

Brent Wilson, Northwestern Representative (2021), (503)554-2722, bwilson@georgefox.edu, George Fox University, 414 N. Meridian St, Newberg, OR 97132.

Mohamed Lotfy, Rocky Mountain Representative (2022), Information Technology Department, College of Computer & Information Sciences, Regis University, Denver, CO 80221.

Tina Johnson, South Central Representative (2021), (940)397-6201, tina.johnson@mwsu.edu, Dept. of Computer Science, Midwestern State University, 3410 Taft Boulevard, Wichita Falls, TX 76308-2099.

Kevin Treu, Southeastern Representative (2021), (864)294-3220, kevin.treu@furman.edu, Furman University, Dept of Computer Science, Greenville, SC 29613.

Bryan Dixon, Southwestern Representative (2020), (530)898-4864, bcdixon@csuchico.edu, Computer Science Department, California State University, Chico, Chico, CA 95929-0410.

Serving the CCSC: These members are serving in positions as indicated:

Brian Snider, Membership Secretary, (503)554-2778, bsnider@georgefox.edu, George Fox University, 414 N. Meridian St, Newberg, OR 97132.

Will Mitchell, Associate Treasurer, (317)392-3038, willmitchell@acm.org, 1455 S. Greenview Ct, Shelbyville, IN 46176-9248.

John Meinke, Associate Editor,

meinkej@acm.org, UMUC Europe Ret, German Post: Werderstr 8, D-68723 Oftersheim, Germany, ph 011-49-6202-5777916.

Shereen Khoja, Comptroller, (503)352-2008, shereen@pacificu.edu, MSC 2615, Pacific University, Forest Grove, OR 97116.

Elizabeth Adams, National Partners Chair, adamses@jmu.edu, James Madison University, 11520 Lockhart Place, Silver Spring, MD 20902.

Megan Thomas, Membership System Administrator, (209)667-3584, mthomas@cs.csustan.edu, Dept. of Computer Science, CSU Stanislaus, One University Circle, Turlock, CA 95382.

Deborah Hwang, Webmaster, (812)488-2193, hwang@evansville.edu, Electrical Engr. & Computer Science, University of Evansville, 1800 Lincoln Ave., Evansville, IN 47722.

CCSC National Partners

The Consortium is very happy to have the following as National Partners. If you have the opportunity please thank them for their support of computing in teaching institutions. As National Partners they are invited to participate in our regional conferences. Visit with their representatives there.

Platinum Partner

Turingscraft

Google for Education

GitHub

NSF – National Science Foundation

Silver Partners

zyBooks

Bronze Partners

National Center for Women and Information Technology

Teradata

Mercury Learning and Information

Mercy College

Welcome to the 2020 CCSC Mid-South Conference

On behalf of everyone at Lyon College, we would like to welcome you to the 18th Annual CCSC Mid-South Conference. The Ozark Mountains are especially welcoming this time of year and we hope you enjoy your time here. It is an honor to host such a wonderful group of students, administrators, researchers, faculty, and professionals from all over the region. Many people have worked hard to put this conference together and to ensure that we continue to have opportunities like this in which to share our research.

The steering committee has come together to offer an impressive agenda. The conference program is made up of ten professional papers, four tutorials, a nifty assignment, a panel, and a workshop. This year, the conference will kick off with vendor presentations and the programming contest offering two divisions: a beginning division for first year programmers and an open division. The winners of both divisions will be announced during the Friday banquet, which will be preceded by faculty and student poster sessions. Students also have the opportunity to present papers and network with some of the top researchers and vendors in the region.

We want to thank everyone who has devoted time to make this conference a reality. Without dedicated individuals reviewing papers, updating the website, making sure the bills are paid, etc. we would not have this opportunity to learn and spend time in fellowship with our peers and students. It is our hope that coming together to share our ideas, to share our research, and to spend some time as a community will not only strengthen ourselves, but our industry and region, as well.

Finally, we want to thank you for attending this conference and visiting Lyon College. Please enjoy your time here and hopefully you can take some time away from our digital world to experience the beautiful mountains, the running streams, and fresh air that the Ozarks has to offer. Please consider being part of the 2021 CCSC-MS Conference as an attendee or as a committee member. It takes passionate people coming together with a common purpose to provide these opportunities. Feel free to contact either of us or anyone on the steering committee to see how you can contribute to future conferences.

Anthony Davis
Lyon College
CCSC-MS Site Chair

Dave Sonnier
Lyon College
CCSC-MS Conference Chair

2020 CCSC Mid-South Conference Steering Committee

David Sonnier, Conference ChairLyon College, AR
Tony Davis, Site ChairLyon College, AR
Larry Morell, Papers Chair Arkansas Tech University, AR
David Middleton, Panels/Workshops/Tutorials Chair Arkansas Tech University, AR
Cong-Cong Xing, Nifty Assignments Chair Nicolls State University, LA
Kriangsiri ‘Top’ Malasari, Student Programming Contest Co-Chair University of Memphis, TN
Brent Yorgey, Student Programming Contest Co-Chair Hendricks College, AR
Matt Brown, Student Papers Chair Arkansas Tech University, AR
Mark Goadrich, RegistrarHendrix College, AR
Gabriel Ferrer, Past Conference ChairHendrix College, AR

Regional Board — 2020 CCSC Mid-South Region

David Naugler, Regional Editor Southeast Missouri State University, MO
Mark Goadrich, Regional Registrar Hendrix College, AR
Nan Harrell, Regional Treasurer Arkansas Tech University, AR
Brian McLaughlan, Regional Treasurer .. University of Arkansas–Fort Smith, AR
David Hoelzeman, Regional Webmaster Arkansas Tech University, AR
David Naugler, CCSC National Board Representative Southeast Missouri State University, MO

Reviewers — 2020 CCSC Mid-South Conference

Brandon, DanChristian Brothers University, Memphis, TN
Brown, MattArkansas Tech University, Russellville, AR
Davis, AnthonyLyon College, Lyon College, Batesville, AR
Ferrer, GabrielHendrix College, Conway, AR
Goadrich, MarkHendrix College, Conway, AR
Malasri, KriangsiriUniversity of Memphis, Memphis, TN
Massengale, RickNorth Arkansas College, Harrison, AR
Middleton, DavidArkansas Tech University, Russellville, AR
McGuffee, JamesChristian Brothers University, Memphis, TN
Morell, LarryArkansas Tech University, Russellville, AR
Renwick, JanetUniversity of Arkansas - Fort Smith, Fort Smith, AR
Sonnier, DavidLyon College, Batesville, AR
Xing, Cong-CongNicholls State University, LA

Design of a Cyber Security Awareness Campaign to be Implemented in a Quarantine Laboratory*

Tanim Sardar and Luay A. Wahsheh
Department of Computer and Information Science
Arkansas Tech University
Russellville, Arkansas 72801
{tsardar, lwahsheh}@atu.edu

Abstract

Humans are still the weakest link in the cyber security chain. Cyber criminals are working faster than users can defend themselves. In this research work, we investigate effective counter-measures to help users stay secure and not be vulnerable to cyber threats. We have designed a training program that introduces university students to several types of cyber attacks. The program is not designed to target one specific major and classification, but rather a variety of majors and classifications. The program includes presentations and hands-on exercises that attract the participant's attention. In order to assess whether the participants retained the presented material, we use a game dubbed "Name that Attack" where the participants are given a scenario and they have to name what type of common cyber attack it is. In addition, on the next day following the training, the participants would be sent a phishing attack via e-mail to see if they would fall victims to this type of attack. By going through the training, we anticipate that the participants will increase their awareness about cyber attacks and be less susceptible to cyber crime.

*Copyright ©2020 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

1 Introduction

1.1 Background

As the number of Internet users and Internet connected devices rise, so does the number of cyber crimes committed. Cyber security is currently one of the fastest growing fields in computing. The cyber security training market was valued at \$240 million in 2016 [10]. This is after an increase of 55% from 2014-2015. Companies are starting to realize the value of cyber security training for their employees and are starting to invest not only into software protections, but also training which would function as a “human firewall” and “the last line of defense” [10]. But even with the rise of the interest in cyber security, users with limited computing background have no awareness of how unsafe their online habits are. Our training program intends to raise user awareness in a way that the user is engaged throughout the training.

1.2 Related Research

There are many programs that companies use to train their employees about cyber security practices. There are also many programs that high schools and colleges use at an early age in an effort to expose students to the topic sooner than the workplace in hopes that they will adopt best practices earlier and carry them on. Jin et al. [12] developed games that are targeted towards high school students. The games focused on detecting threats and implementing ways to stop them. One of the games they created was a simulation to detect phishing attacks. The other game they created was a tower defense style game where the player is adding defenses including firewalls, updates, and encryptions to defend against threats including Distributed Denial-of-Service (DDoS), Trojans, and sniffers.

Ford et al. [8] took a different approach to training. They used an offline approach. The participants work on exercises that are related to real cyber security concepts and real tasks a cyber security professional would do including cryptography, stenography, reverse engineering, and forensics. The exercises disguise the learning as fun little games that the participants are playing. Pham et al. [15] developed a cyber range. A cyber range is a sandbox environment with which students can interact and test out malicious software (malware) and cyber attacks. They developed a system to automate the creation of virtual training requirements based on an instructor’s specifications. This system allows an instructor to define the concepts he or she want to teach and can be integrated into an awareness training program to demonstrate the concepts that the participants are learning.

Research work is being done on how social media can be used as a tool for

increasing awareness. Ikhaliya et al. [11] researched a phenomenon called Mass Interpersonal Persuasion (MIP). They go off the concept that social media users tend to trust the content that their connections on the media share more than something upon which they randomly stumble. With this being the case, the researchers detail how the phenomenon is used to spread malware, but also detail how they hope to use it to promote cyber security awareness.

1.3 Training Method Selection

Our training program follows a presentation format rather than a course format like Das et al. [5] developed. Their course uses debate as the primary method to encourage learning and follows a course format. Our training program is not mandatory.

Estes et al. [7] used a similar approach to encourage learning. They have exercises in which the user takes part. They also cover common attacks that a website may face. Our approach differs in the fact that we use knowledge that the participant has and expand upon it. We use scenarios to which the participants could relate. Also, we cover different vulnerabilities that we think are more present and likely to impact our participants in the real world. This is done in an effort to increase retention of knowledge gained from the training.

Cai and Arney [4] also have a course, but their course focuses on real world attacks that were carried out on company technology that does not require direct human interaction. Our program focuses on real world examples of attacks, but relates them back to how these attacks would be used when attacking a participant in the program. Our program has been designed to be portable. It is not very resource or time intensive. The program has been designed to take about fifty minutes, similar to a regular class session. Being portable makes the program more versatile and adaptable. Instructors can just pull parts from the material if they choose.

Brilingaitė et al. [3] created an expansive computer-based game that puts participants into teams and tests their abilities to handle cyber incidents. Their game is rather expansive with many different scenarios and factors that can be logged and tested, which would take time to explain and deploy. Our game that we employ is very low tech and does not require teaching the participants how to play the game. This is designed so that the program can be administered almost anywhere since it is not very resource intensive.

Muhirwe [14] looked at users in three categories: college users, home users, and corporate users. With those categories, our program can be used in any situational setting. Most of the content is designed to span across all categories, but some are tailored for only one category: university students.

2 Program Elements

We have not implemented this training program yet because of challenges on our campus. We have obtained Institutional Review Board (IRB) approval from our university, but the IT Department did not give us permission to conduct the training on campus due to reservations about subjecting the participants to a cyber attack. Subjecting the participants to a cyber attack is a main differentiating aspect of our training program. The university is in the process of building a controlled environment for students to create malware or carry out cyber attacks. This has been named the Quarantine Lab. The lab is to be used for educational purposes only. The lab location exists, but the equipment has not been setup for use by students yet. This lab is completely separate from the university's network, which is what allows us to safely interact with malware and not worry about infecting the university's network. We intend to use this quarantine lab as the location for our training program. This would be the only location on campus that we are allowed to subject participants to an actual cyber attack.

2.1 Passwords

Passwords are the most valuable piece of information one may have. This is what the criminals are after. In our program, information about passwords is discussed with the participants. Facts include the most common passwords, weak and strong passwords, how often passwords should be changed, and advantages and disadvantages of using passwords. Then, the presenter asks a random participant questions about his or her password. For example, if the presenter asks "What do you usually use for a password?" the participant may say "Oh my dog's name and my birth year". At that point, the presenter starts to do some social engineering with the participant. The presenter could ask "Oh how many dogs do you have?". The participant would answer and then the presenter would ask another question trying to get more information. After the presenter has gathered enough information about the password, he or she will explain his or her logic for asking all the questions and move into the explanation of how that was a social engineering attack. By demonstrating the type of attack, we anticipate the participants retaining the knowledge since a real world example is being demonstrated.

Conte de Leon et al. [6] introduced a more technical explanation of an attack on a password. In their program, they created a tutorial that participants go through and learn the hashing algorithms of a password and how to make more secure passwords. We emphasize the same point without the technical details. We also give participants advice on best practices for their passwords, including not using the same password for every site, not using something easy

to guess, and changing them regularly.

2.2 Phishing

Before the training takes place, we set up a simple website that would collect user data, but was disguised as another website. The presenter asks everyone to use their smart phones and go to a specific link. The presenter tells the participants that they have to create an account to access the information from the website. When they are making the account, this is when their information is being collected. In an effort not to steal much personal data, we require users to input their first name, last name, and email address. After the participants are done, then the presenter pulls up all of the participants information that was just collected. The presenter then goes into how to spot a phishing scam, what to do if you are being phished, and what to do after your information was stolen. The reason we chose to actually collect data rather than show mock user data is that it forms a more personal and real connection with the participant, therefore increasing retention on the subject.

Ayyagari and Figueroa [2] studied the impacts of users allowing applets to steal their information. They educate the users on what the permissions do and what the applets do, then they collect user data in surveys. In our approach, we collect the data without having the user install anything and show them that an attack can come from any source. We also use surveys as a self-assessment method and a supplement to the research. Meyers et al. [13] discussed a method that details how cyber criminals use personal data to create these highly personalized and targeted phishing emails. In our program, we warn about the targeted and highly personalized phishing attacks, teach how to spot them, and how to deal with and prevent such attacks.

2.3 Top Cyber Attacks

In this section of the training program, we inform participants of what a cyber attack is, the most commonly used cyber attacks, how to recognize what kind of an attack it is, and how to rectify the aftermath if an attack is successfully carried out on a participant. We start with a definition of cyber attack. We then list the most common cyber attacks and give the participants some information about such attacks, but in non-technical terms. We only cover the most common types of attacks that would be relevant to our participants. This keeps them engaged in the material, not overload them with information, and increases retention of knowledge gained from the program. We describe scenarios in which that attack has been used.

By providing the participants with scenarios instead of technical definitions, they may retain more of the information and have a working knowledge of the

types of attacks. After the participants have a working knowledge of the types of attacks, the presenter then asks if they want to play a game. The game is titled “Name That Attack”. It is an interactive part of the training where the objective is to correctly guess what type of previously discussed attack happened. The participants are given a made-up or real scenario and they have a set amount of time to read it and solve it.

Alrimawi et al. [1] have a similar approach. They give their participants detailed scenarios of cyber incidents in an effort to find a common pattern. Our approach differs in the target audience. They are targeting more technical users than us. We give our participants the scenarios and focus more on why it is important that they know what is happening rather than the technical details of how it is happening.

Ghiglieri and Stopczynski [9] developed SecLab, an e-learning platform that is designed to be customizable by an instructor in a classroom setting to be gradable and to offer more in-depth technical knowledge. In their platform, participants go through exercises that are then graded. In our training, our game functions as the exercise. Their platform is designed to be more in-depth and can showcase the execution of attacks; their platform is designed for technical users and requires more of a classroom setting. The game portion of our training provides a way of self assessment and can be played anywhere without relying on a platform to be installed.

Weanquoi et al. [16] developed a similar game that addresses only phishing. In their game, the user had to learn about phishing and put his or her skills to the test. In our game, the user is presented with more than just a phishing attack and has to figure out what attack is being used and what to do about that attack.

3 Surveys

Before the program starts, we ask the participants to answer a short survey. The survey asks personal questions about how secure they feel online. Then, at the end of the program we administer the same survey with the same questions and a few questions added about how they felt about the training program and if they have any feedback for improving the program. We use surveys because they are not only a good measurement tool, but also they enhance the program and make the participants think about the material and relate it back to their lives, thus increasing retention of the presented material.

By giving our participants surveys before and after the training program, we can gauge the impact our program has made and identify any areas for improving the program. The game also functions as a quiz since it is essentially testing and reinforcing what the participants have just learned.

4 Conclusions and Future Work

In this research work, we have designed a training program that aims to raise awareness about cyber security issues for university students, and to harden the weakest link in the cyber security chain: the human. In the future, we plan to test the program and measure the impact it has on our campus. We expect that our research would show that the program is offering positive results and helps reduce victims of cyber crime. We plan to implement this training program as a part of mandatory orientation that freshmen students undergo. In addition, we plan to make the training program more modular, which would help instructors to adapt modules from the program and incorporate these modules into lesson plans during class.

References

- [1] Faeq Alrimawi, Liliana Pasquale, Deepak Mehta, and Bashar Nuseibeh. I've seen this before: Sharing cyber-physical incident knowledge. In *Proceedings of the 1st International Workshop on Security Awareness from Design to Deployment*, pages 33–40, 2018.
- [2] Ramakrishna Ayyagari and Norilyz Figueroa. Is seeing believing? Training users on information security: Evidence from java applets. *Journal of Information Systems Education*, 28(2):115–121, 2017.
- [3] Agn e Brilingait e, Linas Bukauskas, Virgilijus Krinickij, and Eduardas Kutka. Environment for cybersecurity tabletop exercises. In *ECGBL 2017 11th European Conference on Game-Based Learning*, pages 47–55, 2017.
- [4] Yu Cai and Todd Arney. Cybersecurity should be taught top-down and case-driven. In *Proceedings of the 18th Annual Conference on Information Technology Education*, pages 103–108, 2017.
- [5] Aparna Das, David Voorhees, Cynthia Choi, and Carl E. Landwehr. Cybersecurity for future presidents: An interdisciplinary non-majors course. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, pages 141–146, 2017.
- [6] Daniel Conte de Leon, Ananth A. Jillepalli, Victor J. House, Jim Alves-Foss, and Frederick T. Sheldon. Tutorials and laboratory for hands-on OS cybersecurity instruction. *Journal of Computing Sciences in Colleges*, 34(1):242–254, 2018.
- [7] Tanya Estes, James Finocchiaro, Jean Blair, Johnathan Robison, Justin Dalme, Michael Emanu, Luke Jenkins, and Edward Sobiesk. A capstone design project for teaching cybersecurity to non-technical users. In *Proceedings of the 17th Annual Conference on Information Technology Education*, pages 142–147, 2016.
- [8] Vitaly Ford, Ambareen Siraj, Ada Haynes, and Eric Brown. Capture the flag unplugged: An offline cyber competition. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, pages 225–230, 2017.

- [9] Marco Ghiglieri and Martin Stopczynski. SecLab: An innovative approach to learn and understand current security and privacy issues. In *Proceedings of the 17th Annual Conference on Information Technology Education*, pages 67–72, 2016.
- [10] Nicole Henderson. Can frequent security training help thwart “as-a-service” attacks? <http://www.itprotoday.com/strategy/can-frequent-security-training-help-thwart-service-attacks>, 2017.
- [11] Ehinome Ikhaliya, Alan Serrano, and Johnnes Arreympi. Deploying social network security awareness through Mass Interpersonal Persuasion (MIP). In *International Conference on Cyber Warfare and Security*, pages 668–674, 2018.
- [12] Ge Jin, Manghui Tu, Tae-Hoon Kim, Justin Heffron, and Jonathan White. Game based cybersecurity training for high school students. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 68–73, 2018.
- [13] Jared J. Meyers, Derek L. Hansen, Justin S. Giboney, and Dale C. Rowe. Training future cybersecurity professionals in spear phishing using SiEVE. In *Proceedings of the 19th Annual SIG Conference on Information Technology Education*, pages 135–140, 2018.
- [14] Jackson Muhirwe. Towards a 3-D approach to cybersecurity awareness for college students. In *Proceedings of the 17th Annual Conference on Information Technology Education*, pages 105–105, 2016.
- [15] Cuong Pham, Dat Tang, Ken-Ichi Chinen, and Razvan Beuran. CyRIS: A cyber range instantiation system for facilitating security training. In *Proceedings of the Seventh Symposium on Information and Communication Technology*, pages 251–258, 2016.
- [16] Patrickson Weanquoi, Jaris Johnson, and Jinghua Zhang. Using a game to teach about phishing. In *Proceedings of the 18th Annual Conference on Information Technology Education*, pages 75–75, 2017.

A Scalable, Hybrid Entity Resolution Process for Unstandardized Entity References*

Awaad Al Sarkhi and John R. Talburt
Information Science Department
University of Arkansas at Little Rock
Little Rock, AR 72204
{aalsarkhi, jrtalbert}@ualr.edu

Abstract

Conventional entity resolution and record linking processes all require a pre-process step to transform the entity references into a standard format with a highly granular, uniform metadata annotation. The ability to avoid the standardization of entity references as a pre-process for entity resolution and still obtain accurate data integration results could substantially accelerate many important data analytics processes including machine learning scenarios [10, 13, 8]. Prior research has shown that a hybrid entity resolution model comprising frequency-based blocking, frequency-based stop word removal, and the scoring matrix can be an effective method for unstandardized and heterogeneously standardized entity references [2, 3, 1]. However, in order for this new process to have practical application, it must be scalable. In the original research, the OYSTER open source ER platform [7, 19] was modified to implement frequency-based blocking and the scoring matrix with frequency-based stop words in order to build a proof-of-concept (POC) system [3]. While the modifications were sufficient to demonstrate the effectiveness of the hybrid model for small reference sets, the POC system is not scalable to large datasets. This paper updates the previous research by describing a second implementation of the hybrid model that is scalable, the results of validating the new process against the original process, and outlining an approach for implementing the hybrid model on a distributed computing platform.

*Copyright ©2020 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

1 Introduction

Entity resolution (ER) is the process of determining whether two references to real-world objects in an information system are referring to the same object, or to different objects. References to the same entity are called equivalent references [14]. The goal of ER is to link two references if and only if the references are equivalent. For this reason, ER is sometimes referred to as record linking [6].

The quality of the linking results of an ER process is measured by the precision, recall, and F-measure of the links it makes between references. The linking precision is the ratio of true positive links (links between equivalent references) to the total number of links made. The linking recall is the ratio of the true positive links to the total number of equivalent pairs (possible true positive links). The F-measure is the harmonic mean of precision and recall.

ER logic is based on the Similarity Assumption which states “the more similar two references are, the more likely they are equivalent, and the less similar they are, the less likely they are equivalent [16]. Both deterministic and probabilistic ER systems start by first assessing the similarity of corresponding attributes in each reference such as the similarity of first names, last names, street numbers, and dates-of-birth [19]. However, this approach assumes the attribute values in both references have metadata tags to indicate their usage (semantics), and all references are using the same metadata tagging scheme [6].

The process to create a uniform set of metadata tags across multiple sources is called data standardization. Traditional ER methods rely on having standardized input references to ensure only values of the same attribute are compared [4]. However, when there are many disparate sources of data, the standardization process often requires a great deal of time and effort to harmonize [5]. Even if each source has already been standardized by an external data provider, different sources may have different standardization schemes. For example, one source may have standardized the references to have separate fields (metadata tags) for the street number and the street name whereas in another source, the standardization scheme has the street number and street name together in a single street address field.

1.1 Background

The Hybrid Model for the ER of unstandardized references developed in prior research comprises three major components [3]. These are frequency-based blocking, frequency-based stop words, and the scoring matrix. Both frequency-based blocking and frequency-based stop words processes begin by preprocessing the reference source. Each reference is treated as a character string. If the

reference has values delimiters, for example a comma-separated value format or CSV files, then each delimiter character is replaced by a blank character. Next, each reference is split into tokens separated by one or more blanks. Then for each token, the non-word characters (non-alphanumeric) in the token are replaced by the empty string, and all letters are changed to upper case. For example, the reference “John,555-1234” would produce two tokens, “JOHN” and “5551234” assuming the comma is a value delimiter. Finally, the extracted tokens are sorted and counted to determine the frequency of each unique token in the reference source.

1.2 Frequency-Based Blocking

Frequency-based blocking is a form of inverted index blocking where each reference is only indexed by its low frequency tokens [6]. Based on the frequency distribution obtained from the preprocessing step, a blocking frequency threshold is set. As a reference is processed in the ER step, it is only indexed by tokens in the reference with a frequency at or below the threshold. Two references in the source will only be compared if they share at least one low-frequency token.

Using a frequency threshold for blocking helps to ensure against large blocks (components) which reduce run-time performance. Depending upon the threshold, it is possible some references might not be indexed if all of their tokens are above the threshold. In that case, each will form a singleton cluster, and not be linked to any other reference.

1.3 Frequency-Based Stop Words

Frequency-based stop words are tokens selected from the reference source based on frequency, but for a different purpose than blocking tokens. Blocking tokens are used to decide which references should be compared, whereas stop words are removed and excluded from the comparison of the two references. Frequency-based stop words are similar to natural language or text processing stop words, except they do not come from a fixed table or list based on language usage [12]. Even though entity references can be treated as documents, they are not “narratives” in the literary sense so they rarely contain typical English language stop words such as “a”, “an”, “and”, “but”, and “of”. For the Hybrid Model, stop words are selected purely on the basis of their frequency. Given a fixed stop word frequency threshold, any token with a frequency at or above the threshold is treated as a stop word, and consequently, does not participate in the reference similarity determination. Prior work has shown that the best results for the Hybrid Model are obtained when the stop word frequency threshold is higher than the blocking frequency threshold [2, 1].

Stop words can be considered a simple form of token weighting in which the most frequently occurring words are given a weight of zero, i.e., are excluded from a comparison in the matrix. All other tokens are given a weight of one. Stop word represent a simplification of the term frequency, inverse document frequency (if-IDF) technique often applied in document retrieval [17].

1.4 Scoring Matrix

After frequency-based blocking and frequency-based stop word removal, the final similarity between two references, and ultimately the linking decision, is done using the scoring matrix. When the scoring matrix processes a pair of references, each reference is first transformed into a list of tokens (words), then the stop word tokens are removed from the list. The remaining tokens from the first reference are used to label the rows of the matrix, and the remaining tokens from the second string label the columns of the matrix. The cell value of the matrix is a normalized similarity measure, i.e., a value in the interval [0,1], between the two tokens.

Several similarity functions such as Jaccard, Normalized Levenshtein Edit Distance (nLED), and Jaro-Winkler can be used individually or in combination to compute cell values [6]. Also, Boolean (True or False) comparators such as nickname match or Soundex match can also be used if their agreement is assigned a numeric value. For example, if two tokens are in agreement by nickname, the agreement could be assigned a fixed similarity value such as 0.95 thus allowing nickname to operate in combination with other similarity functions. In all of the research described here, only normalized Levenshtein Edit Distance (nLED) function was used [11].

To illustrate the operation of the scoring matrix, consider the following two references:

A045, Smith, John, Apt 21, 345 Oak St, Anytown, NY
B167, Jon Smith, 345 Oak Street #21, Anytonw, NY

Furthermore, suppose the threshold for the comparator has been set to 0.80, and the list of stop words contains the token “NY.” The resulting token matrix would then appear as shown in Figure 1. The process begins by finding the largest similarity value in the matrix. This value is the initial value of a total running value. After the largest similarity value is used to initialize the total value, all of the values in the same row and column are removed (set to zero). In the next iteration, the largest similarity value from the remaining values in the matrix is identified and added to the overall total. Again, all of the nLED values in the same row and column as the largest value are removed. The process continues in subsequent iterations until all of the similarity values have been removed from the matrix. In Figure 1, the cells with underlined and bold font are the surviving tokens from this process.

The number of iterations will be equal to the number of tokens from the reference generating the fewest tokens. After the last iteration, the running total is divided by the number of iterations. If the calculated average value is greater than or equal to a threshold value provided by the user, then the comparator returns a “true” result and links the references. Otherwise, the comparator returns a “false” result, and the references are not linked. At the end of the algorithm, the final matrix score for a pair of references in Figure 1 is 0.83. Because 0.83 is above the 0.80 thresholds, the two references would be linked.

	JON	SMITH	345	OAK	STREET	21	ANYTONW
SMITH		1.00			0.17		0.14
JOHN	0.75			0.25			0.14
APT		0.20			0.17		0.29
21						1.00	
345			1.00				
OAK				1.00			0.14
ST		0.40			0.33		0.14
ANYTOWN	0.29	0.14		0.14			0.71

Figure 1: Example References in a Scoring Matrix (zero similarity values omitted)

2 Scalable Solution

2.1 The POC Implementation

The proof-of-concept (POC) implementation of the Hybrid Model was constructed by enhancing the OYSTER open source entity resolution system [15] with two new modules, MatrixTokenizer and MatrixComparator. The index hash function “MatrixTokenizer” implements frequency-based blocking and “MatrixComparator” implements the scoring matrix with stop words. Both the MatrixTokenizer and the MatrixComparator functions take as parameters a list of exclusion tokens. In the case of the MatrixTokenizer, the list comprises the tokens not to be indexed. For the MatrixComparator, it is the list of stop words, i.e. the tokens to be excluded from the comparison of two references. For simplicity in the POC implementation of the Hybrid Model, a preprocessing program OR a preprocessor generated these parameter lists and inserted them into the script used to setup an OYSTER run invoking these functions.

The following example (Example 1) is an excerpt from the setup (run) script for OYSTER showing a typical configuration of the MatrixTokenizer and the MatrixComparator for the POC implementation of the Hybrid Model.

```

<Indices>
  <Index Ident="X1">
    <Segment Item="FullRef"
      Hash="MatrixTokenizer(1,'NC|SALEM|...|27103')"/>
  </Index>
</Indices>
<IdentityRules>
  <Rule Ident="R1">
    <Term Item="FullRef"
      Similarity="MatrixComparator(0.78,'DR|RD|ST|...|27103')"/>
  </Rule>
</IdentityRules>

```

In ordinary usage for standardized references, the `<Indices>` element of the script would define several child `<Index>` elements each defining a multi-segment match key. For example, one match key might be the concatenation of the Soundex of the Student First Name with the string value of the Student Last Name, and second match key might be the concatenation of the string of value of the Student First Name with the 8-digit Student Date-of-Birth. These match keys would support blocking for a set of rules comparing these attributes. While the configuration will vary with type of reference data and matching approach, this approach to blocking and entity resolution depends upon having a uniform, standard layout for each reference.

Because the Hybrid Model does not assume any standardization, the entire input reference (aside from the unique reference identifier) is defined as a single, string value attribute (`FullRef`). The `MatrixTokenizer` parses the reference string into individual tokens for indexing provided the token is not in the exclusion list given as a parameter (shown as `'NC|SALEM|...` in this example). The parameter string for the exclusion list is a pipe character (`|`) delimited list of tokens.

What is not shown in this example is the entire extent of the list. For brevity in this paper, the ellipsis shown in the string parameter indicates the list has more tokens than shown here. In reality, this list may contain several hundred or several thousand tokens depending upon the size of the reference set being processed and type of reference data. Prior research has shown the blocking frequency thresholds producing the best linking results in the Hybrid Model is a relatively low value in the overall token frequency distribution. For example, if the blocking frequency threshold is 10 then every token in the input dataset with a frequency of 11 or higher will be excluded from the inverted index and therefore, must be included in the `MatrixTokenizer` parameter list.

The same is true for the `MatrixComparator` shown in this same script excerpt. Even though the stop word frequency threshold giving the best linking results is larger than the blocking frequency threshold, the stop word frequency threshold is still low relative in the overall token frequency distribution. For example, if the blocking frequency threshold is 10 the stop word frequency threshold might be 15. This would mean that every token with a frequency of

16 or higher must be included in the MatrixComparator stop word list show in this example as starting with the token “DR”. It also means the stop word list is redundant to (is actually the tail of) the blocking exclusion list. In this example, the script indicates pairs of references with a scoring matrix similarity of 0.78 or higher should be linked.

While the POC implementation of the Hybrid Model as originally implemented was effective in demonstrating the effectiveness of the model, it is not scalable. As the reference datasets become larger, so do the parameter lists that must the MatrixTokenizer and MatrixComparator must access. As some point, most or all of the working memory will be consumed with the token tables. It is also not time efficient to perform table lookup operations for every input token for every reference.

2.2 The Scalable Implementation

The primary change in the scalable implementation of the Hybrid Model is to add a second preprocessing step to remove the excluded blocking tokens and stop words. While the first preprocess calculates the frequency of each input token, the second preprocess creates reduced-token (“skinny”) references. While there are several ways to do this; only one is described here.

The original POC and the scalable process both start with a process to read and tokenize each input reference and create a token frequency table. This table contains every token found along with its frequency. In the POC the second process was to build the two parameter lists of tokens. However, the scalable process takes the complementary approach. In the second process, each input reference is read and tokenized, and is output as a re-built reference comprising three parts, the reference identifier, the blocking tokens, and the comparison tokens. Take as Example 2 the following input reference.

B123, John E. Doe, 123 Main St, MyCity CA, 91560, 510-555-6127

Assume that the tokens “JOHN”, “E”, “ST”, “MYCITY”, “CA”, “91560”, “510”, and “555” have frequencies above the blocking frequency threshold, and the tokens “ST”, “CA”, “91560”, and “510” have frequencies above the stop word threshold and are removed. The resulting output reference from the scalable process would be

B123:DOE 123 MAIN 6127:JOHN E DOE 123 MAIN MYCITY 555 6127

Furthermore, this reference would be compared to other references containing the tokens “DOE”, “123”, “MAIN”, and “6127”, because these are blocking tokens. Note the three segments of the reduced-token output reference are separated by the colon (:) character. The first segment is simply the unique reference identifier (RefID) carried forward from the original reference. The

second segment comprises the tokens from the original reference with frequencies at or below the blocking frequency threshold. The third segment comprises only the tokens with frequencies at or below the stop word threshold.

```
<Indices>
  <Index Ident="X1">
    <Segment Item="Seg2" Hash="MatrixTokenizer(1, ' ')" />
  </Index>
</Indices>
<IdentityRules>
  <Rule Ident="R1">
    <Term Item="Seg3" Similarity="MatrixComparator(0.78, ' ')" />
  </Rule>
</IdentityRules>
```

These reduced-token records can now be input directly in the Hybrid Model without the need for token tables as was required in the POC process because all of the excluded tokens have already been filtered. Suppose the second segment of the reduced-token record is identified as “Seg2” and the third segment as “Seg3”, then the OYSTER configuration script for the scalable process would be as shown here as Example 3.

In this script, the MatrixTokenizer function is directed to index references by all tokens in the Seg2 attribute of the reduced-token record with no exclusions (the parameter list is empty). Similarly, the MatrixComparator is directed to compare all tokens between Seg3 of a pair of references with no stop words (again the parameter list is empty). Pairs of references with a scoring matrix similarity of 0.78 or higher should be linked.

Table 1 shows both the POC and Scalable implementations of the Hybrid Model produce the same results. From a performance perspective, the run-time for the Scalable implementation is somewhat less than the POC. However, the most important result is observed for the 670K and 700K samples. The POC process failed with an “out of memory” error while trying to load the token exclusion list for the frequency-based blocking, whereas the Scalable process ran without problems.

Table 1: The POC and Scalable Implementations

Number of Reference	Match Threshold	Blocking Frequency	Stop Word Frequency	POC Implementation		Scalable Implementations	
				F-Meas.	Run Time	F-Meas.	Run Time
2,000	0.75	10	22	0.934	1 sec	0.934	1 sec
5,004	0.76	10	51	0.905	3 sec	0.905	4 sec
100,002	0.71	61	1,492	0.918	2 min	0.918	1 min
506,228	0.80	100	4,000	0.656	1 hour 43 min	0.656	50 min
650,071	0.81	92	6,008	0.654	1 hour 45 min	0.654	59 min
670,063	0.79	90	6,000	Out of Memory Error		0.642	1 hour 21 min
700,212	0.80	80	7,004	Out of Memory Error		0.618	1 hour 40 min

3 Conclusion and Future Work

Prior research has shown the Hybrid Model to be an effective method for entity resolution of unstandardized references, an important step toward fully automating the data curation process. The research presented here shows that this model has a scalable implementation. One direction for future work is translating the Hybrid Model into originally implemented was suitable in demonstrating the effectiveness for the distributed (HDFS) computing environment. The basic premise of frequency-based blocking and stop words is ideal for Map/Reduce (M/R) programming because both are fundamentally “word count” problems.

The high-level schematic of a M/R implementation of the Hybrid Mode is shown in Figure 2. In the top flow of Figure 2, the process starts as a simple word count. The references are mapped to nodes that parse them into tokens and emit Token-RefID key-value pairs. These are reduced into token (key) groups. If the size of the group (N) is greater than both the blocking frequency threshold (β) or the stop word frequency (σ), the group is ignored. If the N is greater than β the pairs of the group are reversed and written to File1. Similarly, if N is greater than σ the pairs are reversed and written File2.

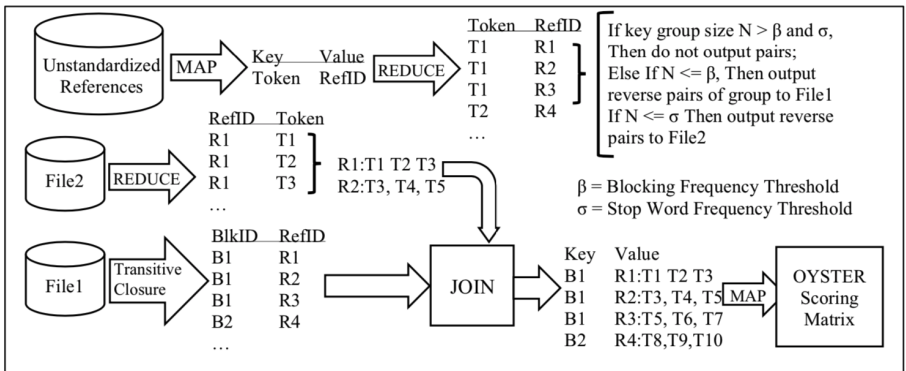


Figure 2: Schematic of Map/Reduce Implementation of Hybrid Model

In the middle flow of Figure 2, File 2 is sorted by RefID and in a reduce process, the reduced-token version of the input is rebuilt using only non-stop words. In the bottom flow of Figure 2, an iterative transitive closure process [18, 9] creates the blocks of references sharing one or more of the blocking tokens. The final step is a join between the two flows to create key-value pairs where the key is the block identifier (BlkID) and the value is reduced-token reference including the RefID. The final processing step is to map the block

groups to nodes running OYSTER. The OYSTER run script for processing each block can also be simplified as shown here as Example 4 without indexing because the references are already blocked. The link outputs of OYSTER can simply be merged in a final reduce step to produce the final clusters for the original input.

```
<IdentityRules>  
  <Rule Ident="R1">  
    <Term Item="TokenString" Similarity="MatrixComparator(0.78, ' ')" />  
  </Rule>  
</IdentityRules>
```

References

- [1] Awaad Al-Sarkhi and John R Talburt. Estimating the parameters for linking unstandardized references with the matrix comparator. *Journal of Information Technology Management*, 10(4):12–26, 2018.
- [2] Awaad Alsarkhi and John R Talburt. An analysis of the effect of stop words on the performance of the matrix comparator for entity resolution. *Journal of Computing Sciences in Colleges*, 34(7):64–71, 2019.
- [3] Awaad Alsarkhi and John R Talburt. Optimizing inverted index blocking for the matrix comparator in linking unstandardized references. In *Proceedings of the International Conference on Scientific Computing (CSC)*, pages 10–16, 2019.
- [4] Éloi Bossé and Galina L Rogova. *Information Quality in Information Fusion and Decision Making*. Springer, 2019.
- [5] Ursin Brunner and Kurt Stockinger. Entity matching on unstructured data: an active learning approach. In *2019 6th Swiss Conference on Data Science (SDS)*, pages 97–102. IEEE, 2019.
- [6] Peter Christen. *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection*. Springer Science & Business Media, 2012.
- [7] Center for Advanced Research in Entity Resolution and Information Quality. Oyster open source project. <https://bitbucket.org/oysterer/oyster/> Accessed 2019.
- [8] Anna Jurek-Loughrey and P Deepak. Semi-supervised and unsupervised approaches to record pairs classification in multi-source data linkage. *Linking and Mining Heterogeneous and Multi-view Data*, page 55, 2018.
- [9] Lars Kolb, Ziad Sehili, and Erhard Rahm. Iterative computation of connected graph components with MapReduce. *Datenbank-Spektrum*, 14(2):107–117, 2014.
- [10] Xinming Li, John R Talburt, Ting Li, and Xiangwen Liu. Scoring matrix combined with machine learning for heterogeneously structured entity resolution. *Journal of Computing Sciences in Colleges*, 34(7):38–45, 2019.

- [11] Doaa Medhat, Ahmed Hassan, and Cherif Salama. A hybrid cross-language name matching technique using novel modified levenshtein distance. In *2015 Tenth International Conference on Computer Engineering & Systems (ICCES)*, pages 204–209. IEEE, 2015.
- [12] George V Moustakides and Vassilios S Verykios. Optimal stopping: A record-linkage approach. *Journal of Data and Information Quality (JDIQ)*, 1(2):1–34, 2009.
- [13] Kevin O’Hare, Anna Jurek-Loughrey, and Cassio de Campos. An unsupervised blocking technique for more efficient record linkage. *Data & Knowledge Engineering*, 122:181–195, 2019.
- [14] John R Talburt. *Entity resolution and information quality*. Elsevier, 2011.
- [15] John R Talburt and Yinle Zhou. A practical guide to entity resolution with OYSTER. In *Handbook of Data Quality*, pages 235–270. Springer, 2013.
- [16] John R Talburt and Yinle Zhou. *Entity information life cycle for big data: Master data management and information integration*. Morgan Kaufmann, 2015.
- [17] Na Wang, Pengyuan Wang, and Baowei Zhang. An improved TF-IDF weights function based on information theory. In *2010 International Conference on Computer and Communication Technologies in Agriculture Engineering*, volume 3, pages 439–441. IEEE, 2010.
- [18] Bingyi Zhong and John Talburt. Using iterative computation of connected graph components for post-entity resolution transitive closure. In *2018 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 164–168. IEEE, 2018.
- [19] Y Zhou and JR Talburt. OYSTER: An open source entity resolution system supporting identity information management. In *ID360-The Global Forum on Identity, Austin*, volume 90, 2012.

Exploration of Factors Contributing to Academic Success in a Data Analytics Program*

Matt Brown
College of Business
Arkansas Tech University
Russellville, AR 72801
hbrown11@atu.edu

Abstract

This paper examines potential indicators of student risk for failure to complete a degree in data analytics. Records from 207 students from 2012 to 2019 were used in the study. Factors considered in the study include, ACT exam composite score, high school grade point average, transfer students, grade earned in a first mathematics course, grade earned in a first data analytics course, first generation students, and income level as classified by Pell Grant eligibility. Of these factors, only four factors were found to be significant, ACT composite score, grade earned in a first mathematics course, grade earned in a first data analytics course, and income level. Further, data indicate that performance in a first college mathematics course is the strongest indicator of a student's likelihood to succeed in a data analytics degree. Students in this study earning a C or lower in a first course in mathematics had less than a 40% change of completing a data analytics degree.

1 Introduction

Data analytics, data science, and related degrees are relatively new fields of growing importance. Degree offerings in these fields are on the rise in universities [7]. At the same time universities and academic programs are facing

*Copyright ©2020 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

increasing pressures to improve student retention and academic success [12]. However, because the newness of data analytics related degrees, little research has specifically focused on retention in these fields. Toward that goal, this research seeks to gain insight on risk factors in the completion of a data analytics degree. In particular, records from 207 students majoring in a data analytics degree for at least one semester from 2012 to 2019 were analyzed for factors that may lead to students failing to complete a business data analytics major.

Predicting student risk for failure to succeed early in academic careers provides a chance for intervention, such as tutoring or other retention programs. Therefore, student risk of failure needs to be predicted from data typically available in an academic environment. This research attempts to determine factors contributing to academic success in a business data analytics program using data readily available to university personnel.

2 Background

Data analytics, data science, business intelligence, big data, and other related fields have become essential in business and academic communities [9]. As a result, the demand for data analytics as a field of study and the number of universities offering these degrees are steadily increasing [7]. Data analytics and related programs are multidisciplinary and can be housed in computer science, business, mathematics, or statistics departments [10]. While research regarding student success in each of these larger general fields of study has been done, factors related to student success specifically in the newer data analytics related majors have not been as well studied. This research specifically considers factors impacting student success in a business data analytics major.

Conclusions from related studies in the larger fields that relate to data analytics, computing, business, and mathematics are briefly summarized. Factors in computer science and related majors that were shown to impact success include negative student perceptions concerning the major, students feeling inadequately prepared curriculum, poor advising, poor math and problem solving skills, poorly designed courses, teaching practices, and poor performance in a first course in computing [2, 3, 4]. In the fields of mathematics and statistics, similar factors with the addition of high school performance, were shown to be factors impacting performance [14]. Further, in mathematics, factors that were traditionally thought to be good indicators of student success, such as general numeracy tests, were not always found to be good predictors of student success [8]. In the field of business, grades earned in a principles of economics courses, student sense of purpose, teaching quality and support, academic self- efficacy and social integration were shown to be significantly impact student success [11, 13].

The importance of student retention is the motivation for discovering factors that impact student success. Early identification of students at risk can lead to interventions to try and improve retention [6]. Therefore, the goal of this study is to identify factors that can be used from readily available data for early identification of students at risk for not completing a business data analytics degree.

3 Details of the Study

The university considered in this study is a SREB 3, four-year state institution. The university has an annual enrollment of approximately 12,000 students. The data analytics degree is a four-year Bachelor of Science in Business Administration with a major in Business Data Analytics (BDA) from an AACSB-accredited undergraduate business program. The data analytics program includes courses in statistics, databases, predictive modeling, business intelligence, and application development. The data analytics major is an applied data science program, emphasizing tools such as spreadsheets, SQL, Python, R, SAS, Hadoop, and other analytics, big data, data mining, and data science technologies. Current enrollment in the BDA program is approximately 80 students. The study considers 206 BDA students from 2012 to 2019.

Three different dependent variables were used to quantify student success, Grade Point Average (GPA) for students enrolled in the BDA program, hours completed toward a BDA degree, and successful completion of a BDA degree. The potential independent variables considered include, high school grade point average, a variable indicating if the student transferred from a different institution, grade earned in their first mathematics class taken at this institution, grade earned in their first data analytics course taken at this institution, a variable indicating if the student is a first generation student, student income level as classified by Pell grant eligibility (not low income, low income, lowest income, see [1] for more details), and composite score on the ACT college entrance examination. The first college mathematics course varied based on student preparedness, for 39% of students it was college algebra, for 32% of students it was quantitative business analysis, for 12% of students it was a remedial mathematics course, for 8% of students it was calculus, the rest were miscellaneous mathematics courses. The first course in data analytics for 95% of the students was a course entitled, "Business Problem Solving". The number of students used for each model varied because of missing observations. Also, only students who have either completed the BDA degree or were no longer enrolled in the BDA program (without successful completion) were considered when looking at successful completion of a BDA degree. For earned hours, only students that did not successfully complete the BDA major were used to de-

termine what factors might impact how far into the program they progressed. Students currently still working toward the BDA degree were left out of the study, since their success is yet to be determined.

4 Results

4.1 Factors that Predict GPA

Next, earned hours for students enrolled in the BDA program that did not complete the degree, was used as a dependent variable. Specifically, 120 hours are required for the BDA degree, students who did not earn the 120 hours required and were no longer enrolled in the BDA program were considered. The potential independent variables considered again include, high school grade point average, indicator of transfer student status, grade earned in first mathematics course, grade earned in first business data analytics course, indicator of first generation student status, income level, and ACT composite score. Multiple regression with stepwise variable selection was again used to determine which factors were most related to earned hours of unsuccessful BDA students. Only two variables were selected as significant: first mathematics course grade and ACT composite score. Together these factors had an R-squared of 0.40. The single best predictor was again first earned grade in a mathematics course, with a partial R-squared value of 0.29. Among students that did not complete the BDA degree, as grades in the first mathematics course increased and scores on the ACT increased the earned hours toward a BDA degree increased.

4.2 Factors that Predict Student Success in Competition a BDA Degree

Finally, student success in completing a BDA degree was used as a dependent variable. Only students who have either completed the BDA degree or were no longer enrolled in the BDA program (without successful completion) were considered when looking at successful completion of a BDA degree. Again, the potential independent variables considered include, high school grade point average, indicator of transfer student status, grade earned in first mathematics course, grade earned in first business data analytics course, indicator of first generation student status, income level, and ACT composite score. Only first mathematics course grade was found to be a significant predictor of student success in completion of a BDA degree. From the estimated odds ratio, for every one letter grade increase in a first mathematics course students were 3.5 times more likely to complete a BDA degree. Further, as can be seen in Figure 1, a grade of a C or lower in a first mathematics course increases the probability of failure to complete a BDA degree to more than 0.6.

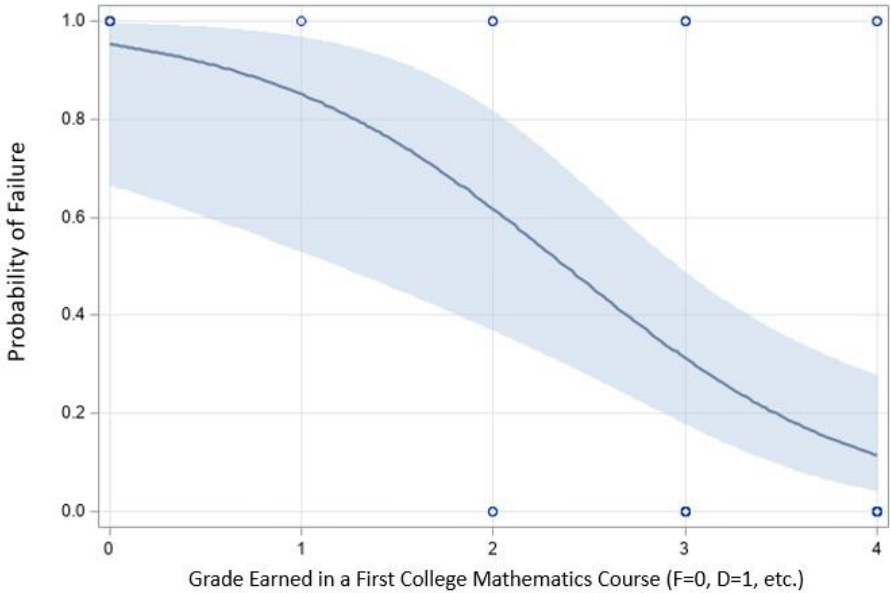


Figure 1: Logistic regression model: probability of failure to complete a BDA degree versus grade earned in a first mathematics course (with 95% confidence limits)

5 Discussion

Of the variables considered in this study, factors that impact success of business data analytics students were identified as grade earned in a first course in mathematics, grade earned in a first data analytics course, income level as identified by Pell grant status, and composite ACT score. Of these factors, grade earned in a first mathematics course was the only factor found to be significant in predicting all three measures of success. Performance in a first mathematics course was also found to be the most significant of these factors in all three models. In particular, students earning a C or lower in a first course in mathematics have less than a 40% change of completing a data analytics degree.

Due to the quantitative nature of data analytics, the importance of performance in a first course in mathematics in predicting success in a BDA program should come as no surprise. Further, university mathematics courses have long been identified as a “gatekeeper” courses for overall student success [5]. However, the degree to which performance in a first mathematics course predicts

success, in particular in comparison to other factors typically tied to student success, identifies it as a simple, but potentially good indicator of student risk to complete a BDA degree. Since the majority of students take their first mathematics course early in their academic careers, intervention to improve retention based on this factor should be possible.

This study is not exhaustive in terms of which factors were considered as predictors or in terms of a representation of all data analytics and data science programs. The extent to which these results can be generalized to other universities would need to be tested with more research. However, despite these limitations, this research provides evidence of the efficacy of the readily available student grades in a first course in mathematics as a predictor of success in a data analytics program.

References

- [1] Federal student aid. <https://studentaid.ed.gov/sa/fafsa/> Accessed December 11 2019.
- [2] Theresa Beaubouef and John Mason. Why the high attrition rate for computer science students: some thoughts and observations. *ACM SIGCSE Bulletin*, 37(2):103–106, 2005.
- [3] Maureen Biggers, Anne Brauer, and Tuba Yilmaz. Student perceptions of computer science: a retention study comparing graduating seniors with CS leavers. *ACM SIGCSE Bulletin*, 40(1):402–406, 2008.
- [4] Matt Brown. CS0 as an indicator of student risk for failure to complete a degree in computing. *Journal of Computing Sciences in Colleges*, 28(5):9–16, 2013.
- [5] Anthony S Bryk and Uri Treisman. Make math a gateway, not a gate-keeper. *Chronicle of Higher Education*, 56(32):B19–B20, 2010.
- [6] AT Chamillard. Using student performance predictions in a computer science curriculum. *ACM SIGCSE Bulletin*, 38(3):260–264, 2006.
- [7] Penny R Clayton and Jeremy Clopton. Business curriculum redesign: Integrating data analytics. *Journal of Education for Business*, 94(1):57–63, 2019.
- [8] Alistair J Harvey. The merits of a general numeracy test as a predictor of undergraduate statistics performance. *Psychology Learning & Teaching*, 8(2):16–22, 2009.

- [9] Ismail Bile Hassan and Jigang Liu. Data science academic programs in the US. *Journal of Computing Sciences in Colleges*, 34(7):56–63, 2019.
- [10] Jessen Havill. Embracing the liberal arts in an interdisciplinary data analytics program. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 9–14, 2019.
- [11] Rupert G Rhodd, Sandra M Schrouder, and Marcus T Allen. Does the performance on principles of economics courses affect the overall academic success of undergraduate business majors? *International Review of Economics Education*, 8(1):48–63, 2009.
- [12] Amy Rummel, Maeghen L. MacDonald, and Justin Cornelius. Drivers of student retention: The need for service marketing. In *Allied Academies International Conference: Proceedings of the Academy of Marketing Studies (AMS)*, volume 16, 2011.
- [13] Lesley Willcoxson. Why do business students drop out? evidence from first, second, and third year students. In *Proceedings of the 23rd Australian and New Zealand Academy of Management Conference*, pages 1–19. Australian and New Zealand Academy of Management (ANZAM), 2009.
- [14] Darwish Abdulrahman Yousef. Determinants of the academic performance of undergraduate students in statistics bachelor’s degree program. *Quality Assurance in Education*, 2019.

A Programming Workshop Course to Complement CS 1/2*

Christopher A. Healy
Department of Computer Science
Furman University
Greenville, SC 29613
chris.healy@furman.edu

Abstract

This paper describes the implementation of a new laboratory-only course called the Programming Workshop. Its purpose is to reinforce the problem-solving skills learned in early programming courses such as CS 1. Without the burden of lectures, exams, homework, and grades, students take this class simply because they are motivated to learn. Because each student's needs are different, the course is self-directed. On the first day, students individually confer with the instructor about their goals, and the instructor draws up a suggested plan of activities. Students create a portfolio of their work. Of the twenty students who have completed this course in the last two years, the average student finished 19 programs, likely more than they would have accomplished in a full-credit course.

1 Introduction

Like many colleges, ours offers the canonical courses CS 0, CS 1, and CS 2 [6] at the beginning of the computer science curriculum. However, our experience has shown that many students do not take these courses in consecutive terms, and sometimes not even in consecutive years. Foundational programming skills learned in CS 1 may be stale by the time the student enters CS 2. A student may earn a mediocre grade in CS 1 and be apprehensive to continue in CS 2. Computer science majors in the bachelor-of-arts track sometimes wait until

*Copyright ©2020 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

their senior year to take CS 2, which could be their last programming course. In addition, non-majors often do not have room in their schedule to take another computing course after CS 1. Finally, since we teach Python in CS 1 and Java in CS 2, some students desire additional experience in one of these languages. These considerations motivated our department to offer a new course, Programming Workshop, as a means to complement the CS 0-1-2 sequence.

The Programming Workshop is a zero-credit laboratory course where students can hone their programming skill, to become more proficient and confident in their abilities. It is analogous to a student taking a foreign language spending several hours per week in a language laboratory to practice listening and speaking. It is also analogous to a student learning a musical instrument who must spend hours a week in practice. The philosophy of the Programming Workshop is to provide students an environment where they can intensively practice programming, and not be burdened with lectures, exams, and grades. In order to learn, it is often necessary to experience many mistakes along the way. Our goal is for the class to be an uninhibited environment where students can feel free to learn from mistakes and know that their grade is not being harmed by them.

2 Related Work

The Programming Workshop can be compared to other alternatives to traditional instruction. For example, Hodges [4] discusses a weekly flipped classroom exercise, where students come to class ready to practice problem solving having just watched a video lesson. Our workshop does not have its own formal lessons because it is a review course, rather than an introduction to programming. Conner and Lambert [2] use Git as a tool to help students ultimately create a portfolio of programming work. We stress portfolios as well, but we do not insist on a formal structure such as Git. Vanderhyde [8] discusses how scaffolding a large assignment into manageable chunks allows students to accomplish each in a short period of time. Jonas [5] discusses how programming students use class time to work in groups and discover solutions to problems posed in class. The main difference with our workshop is that most of our students work individually.

Online platforms such as Turing's Craft [3] provide students with hundreds of exercises that can be done outside of class with immediate feedback. These problems are usually very short and are generally used during CS 1 because the exercises are synchronized with the textbook. In our workshop, we aim to give problems that take on the order of an hour to complete. Finally, some colleges offer a specialized laboratory course to prepare students for programming contests, for example [7]. We do not advertise our workshop as a contest

prep course, though some students could conceivably use it for that purpose.

3 Course Features

The pre-requisite of the Programming Workshop is CS 1, because the purpose of the course is not to teach programming from scratch. Instead, students should take this course to review CS 1, or to apply their CS 1 experience to a new language. Students may repeat the course an unlimited number of times. The course capacity is 12, to ensure that students receive individual attention as needed from the instructor.

The course is scheduled differently from our other programming classes. A typical course in our department is taught three times per week for 50 minutes, plus a two-hour lab period once a week. Since the workshop is designed to complement the existing curriculum, we scheduled it during a mid-day Tuesday-Thursday time period when no other computer science course is offered, in order to encourage as many people as possible to register. The class period is 75 minutes long, which is closer in duration to the two-hour lab period that students are already accustomed to in other classes.

As befitting a course that does not carry credit, the workshop has no lectures, no homework, no tests, and no grades. The grading is pass/fail. Therefore, students come to this course by their motivation to learn, rather than to receive a grade or improve their grade point average. Similarly, we sought to remove the worry of possibly receiving a low grade when taking an extra programming course. We could have created this as a one-credit course, but we decided not to for a couple of reasons. First, we did not want students to take the workshop in place of another course. Second, the college's committee that approves new courses informed us that a one-credit course would need to be graded A-F. A further advantage to making the workshop a zero-credit course is that students can easily add it to their schedule without incurring any additional tuition fees. And the time commitment is minimal, just attending the class three hours per week.

The Programming Workshop is an elective course, not applicable towards the major. Although the instructor receives no teaching credit for offering the course, it should be noted that preparing the course requires much less effort than a typical course, as there are no lectures and practically nothing to grade.

4 Course Procedure

The course begins with a short in-class writing exercise. On the first day, students describe their personal course goals. The instructor confers personally with each student to clarify these goals and give direction on how to proceed.

The student decides on a language, and specifies which skills to strengthen. If the student is undecided, then the instructor recommends a pre-defined battery of laboratory exercises that begins with the basics and gradually increases in difficulty.

At each class meeting, the instructor is in the room, always available to discuss ideas, to answer questions, and to assist in testing programs. In order to successfully complete the course, students must attend at least 75% of the class meetings, demonstrate appreciable progress toward their goals outlined at the beginning of the term, and compile an organized portfolio of their work. On the last day of the course, students briefly showcase their favorite work, and they submit their portfolio to the instructor for review. The portfolio must include a table of contents indicating what work has been completed in the term, and what work the student did not finish.

5 Course Content

The author wrote a set of 59 laboratory exercises to support this course, a large enough assortment from which the students can choose. Each problem is designed to be completed within one or two 75-minute class periods. Example exercises include:

- Count how many times each letter of the alphabet appears in a text file.
- Compute the standard deviation of numbers listed in a text file.
- Input an integer and convert it into Roman numerals.
- Input an integer and determine its prime factorization, and whether it is abundant, deficient, or perfect.
- Given a selection of nine letters, find the longest word(s) in the dictionary that can be formed using letters from this selection.
- Score a bowling game.
- Write and use comparator classes to sort a list of basketball players.

Students are not expected to complete all of these exercises in one semester, as they cover a wide range of difficulty.

The students are given a recommended order in which to attempt the 59 exercises. The content of these exercises is similar to all of CS 1 plus most of CS 2. This recommended order of exercises differs from how programming is introduced in CS 1. It begins with a thorough treatment of input and output. The first exercise covers input, including interactive (stdin) input, file input, tokenizing strings, and exceptions that could occur while reading input. As one can see, these concepts are generally taught at different points during a CS 1 course. In CS 1, interactive input is taught very early, and exception handling much later. But for a review course, we felt it was important to think

of input holistically. Similarly, the output lab also covers multiple topics such as formatted output and file output. Once students have completed these first two exercises, they should be able to handle the I/O requirements of all of the remaining exercises.

Beyond the provided exercises, advanced students are encouraged to seek additional exercises from outside sources. One type of source is a book designed to prepare students for technical programming interviews [1]. For some students, past programming contest questions are another possible source of problems to work on.

6 Results

The Programming Workshop was offered in 2017 and 2018. A total of 22 students enrolled. This enrollment figure was lower than we anticipated. A year before the course was first offered, we administered a survey to all computer science students. Of 53 students who completed the survey, 40 (75%) said that they were willing to take the course. When we later asked students why they did not take the course, the main objection was that it was not worth investing three hours a week to receive no credit, even though the course would appear on their transcript.

Of the 22 students who have taken the workshop, six were not computer science majors (four mathematics majors and two art majors). Figure 1 shows the number of students by each year of college. Evidently, most students take the workshop rather late in their careers, when their minds are turning to jobs and preparing for technical interviews or graduate school. Anecdotally, several students told us they took their required programming courses early in their college career, and now they want to refresh these skills to prepare for job interviews or advanced courses that require programming. Among the 22 students, 16 (73%) had already taken CS 2, even though the pre-requisite is only CS 1. Figure 2 shows the number of students who selected each language. Python and Java are the only languages formally taught in our regular curriculum. Clearly, most students were using the workshop to review a language taught in CS 1 or 2. It is interesting to note that six students selected a different language to further diversify their repertoire.

Twenty of the 22 students successfully completed the course. The other two students (a sophomore art student and a freshman CS student) withdrew during the first few weeks of the term. We were pleased with the size of the portfolios created by the students who finished the term. The average student completed 19 programming exercises. It should be noted that a typical CS 1 or 2 course may require a student to complete fewer than ten programs. Almost every student worked independently, since they worked on different

Students by Year in School

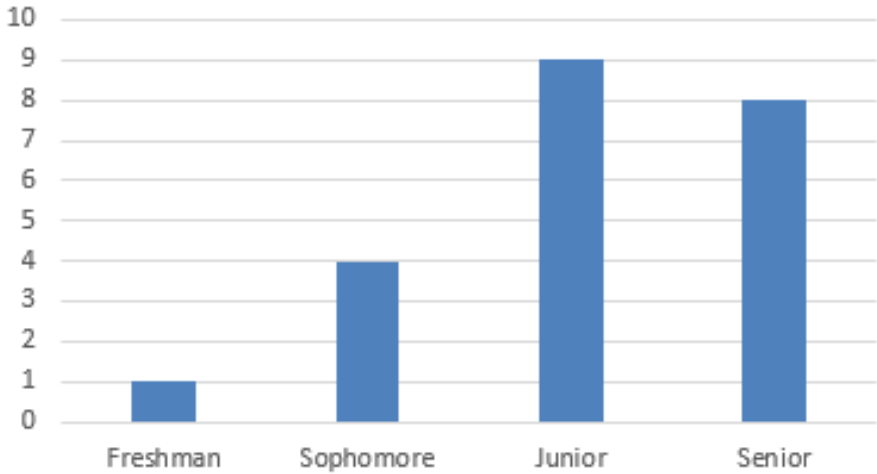


Figure 1: Number of workshop students by year in school.

Students by Chosen Language

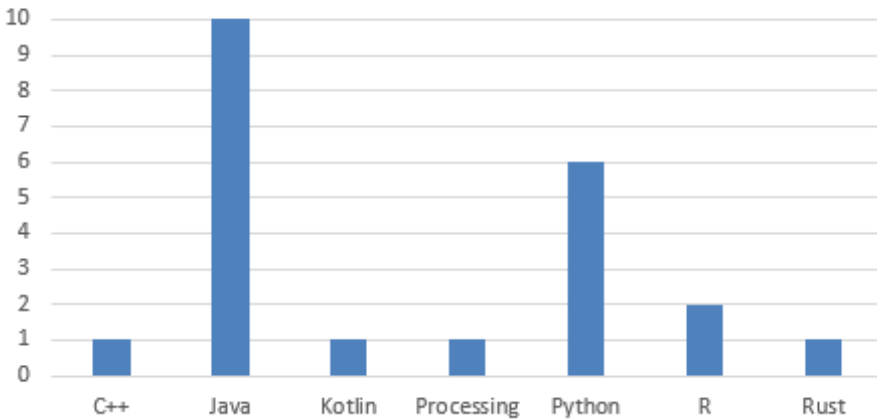


Figure 2: Number of workshop students by language selected by each.

problems or progressed at different speeds. Only once was it observed that a pair of students worked problems in tandem during the course, where they often compared approaches.

7 Conclusion

The Programming Workshop is a new zero-credit course designed for students to review programming skills or learn a new programming language. It is self-directed to meet the various needs of the students. It serves as a complement to the traditional CS 0-1-2 sequence, and can be taken before, during, or after CS 2. Students create their own course objectives, and they compile a portfolio of their successful work. This course has been offered for two years. For the students who were able to fit it in their course schedule, all showed significant growth on par with taking a full-credit course.

References

- [1] Adnan Aziz, Amit Prakash, and Tsung-Hsien Lee. *Elements of Programming Interviews*. CreateSpace Independent Publishing, 2012.
- [2] David Conner and Lynn Lambert. Integrating git into CS1/2. *Journal of Computing Sciences in Colleges*, 35(3):112–121, 2019.
- [3] Turing’s Craft. <http://turingscraft.com>.
- [4] Mark Hodges. Flipping one day each week in a smaller CS1 course: An experience report. *Journal of Computing Sciences in Colleges*, 34(7):20–27, 2019.
- [5] Michael Jonas. Lessons learned from integrating POGIL into a CS1 course. *Journal of Computing Sciences in Colleges*, 34(6), 2019.
- [6] Joint Task Force on Computing Curricula 2013. Curriculum guidelines for undergraduate programs in computer science. https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf.
- [7] Stanford University Computer Science. CS 97SI: Introduction to programming contests. <https://web.stanford.edu/class/cs97si>.
- [8] James Vanderhyde. Scaffolding assignments: How much is enough? *Journal of Computing Sciences in Colleges*, 34(3), 2019.

Why Teach Operating Systems?*

James W. McGuffee
School of Sciences
Christian Brothers University
Memphis, TN 38104
jmcguff1@cbu.edu

Abstract

In the past four decades, there has been much written on how to teach operating systems courses. This paper reviews and reflects on that history. Additionally, this paper attempts to answer why we should be teaching operating systems. The answer to why we should be teaching operating systems should necessarily inform us on how we should be teaching operating systems. This is a professional position and advocacy paper within the context of computer science education at the undergraduate level.

1 Motivation and Scope

On Thursday, February 28, 2019, as part of ACM SIGCSE’s Technical Symposium, I attended a Birds of a Feather session on “Developing a Contemporary and Innovative Operating Systems Course”. This session was led by Saverio Perugini and was related to his work as a primary investigator of an NSF IUSE funded project to support the development and widespread dissemination of a contemporary operating systems course as part of an undergraduate computer science curriculum [12]. Specifically, the Perugini led project is attempting to develop a model operating systems curriculum that includes modules that cover cybersecurity, mobile operating systems, the Internet of Things (IoT), concurrent programming, synchronization, cloud computing, and big data processing [10]. The Birds of a Feather session was a chance for ACM SIGCSE members interested in the teaching of operating systems at the undergraduate level to

*Copyright ©2020 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

brainstorm and discuss what should be considered as essential components of a modern operating systems course. One of the outcomes of this session was the creation of a “Teaching Operating Systems Community of Practice” working group that I enthusiastically joined [11].

As a result of joining this community of practice, I have been reviewing professional articles and other sources related to the teaching of operating systems. It has been fascinating to discover the rich history of what has been proposed as essential to teach in an operating systems course. I have also learned the various ways of how the operating systems course has been taught. One interesting omission from all this work is a discussion of why we teach operating systems.

My assumption is that most will feel that the answer to why we teach operating systems is obvious. That the question itself is trivial and unnecessary. I think both of those positions are short sighted. Before we can answer what we should teach or how we should teach, we really need to answer why should we teach this material. After a summary of the history of operating systems education publications over the past 40+ years, I will attempt to give some insight into and begin to answer the question of why we should be teaching operating systems in higher education at the undergraduate level.

2 The History of How

In the opening section of a booklet intended as commentary on the sixth edition of the UNIX operating system source code, J. Lions describes three main approaches to the teaching of operating systems [7]. The first approach described is the general principles approach where fundamental principles are expounded and illustrated by references to various modern operating systems. Lions argues that undergraduate students lack the maturity to take full advantage of this approach. The second approach described is the building block approach that has students build a simple operating system from scratch. Lions dismisses this approach by deriding the systems as toy operating systems lacking in complexity to be of any practical use,

The third approach to teaching operating systems as described by J. Lions is the case study approach. The case study approach advocates devoting most of the course to the study of one operating system. This approach was first advocated in the 1968 ACM curriculum guidelines. Lions argues that this is the best approach but was unrealistic for most schools in 1968 but, by 1976, things had changed and UNIX was then an ideal operating system for the case study approach to teaching operating systems at the undergraduate level [7].

In 1978, Joel Gyllenskog described two different ways to teach an operating systems class. In the classical principles class, students are introduced to

concepts of operating systems without actually having to work on an actual computer system. The topics covered in a principles class included: multiprogramming, critical sections, deadlocks, storage management policies, memory mapping, file directories, handling interrupts, real time clock, memory protection, privileged instructions, communication with I/O devices, and re-entrant code [3].

The other approach to teaching operating systems as described by Gyllenskog is known as the pragmatic approach. In the pragmatic approach the same general topics are generally covered as in the principles class but the topics are taught within the context of designing and constructing an operating system. While acknowledging that the scope of the topics within a pragmatic framework would be limited, Gyllenskog defined four necessary components that were necessary implementations. The operating system must be multiprogrammed, provide virtual I/O, include a consistent file structure, and protect each program from all other programs [3].

In 1989, Su Yun-Lin described what had been learned from teaching operating systems from the previous decade. Yun-Lin described three ways to teach operating systems. Though J. Lions work is not referenced directly, two of the ways described are nearly identical to the first and third approaches referenced by J. Lions. These two approaches are a general principles approach and a case study of one system approach. It is Yun-Lin's description of an historical development approach that is unique. This approach is to teach operating systems by covering the historical development of the four generations of operating systems. While liking this approach, Yun-Lin cautions that history is not always straightforward and that anomalous examples exist for each generation. Ominously, Yun-Lin concludes the paper with a brief warning that new generations of computers are requiring even more sophisticated operating systems and that an additional challenge to teachers of operating systems classes is the need to consider the emerging influence of computer networks and distributed systems [16].

Allen Downey describes yet another approach to teaching the operating systems class. Instead of implementing operating systems components or modifying existing components of an existing operating system, students in Downey's class conducted a series of experiments that measured the performance of system services and attempted to infer implementation information from the results. Downey also advocated that this approach to teaching operating systems allowed students to develop skills in hypothesis testing, analyzing data, and writing [2].

Some operating system educators advocate for the importance of using one programming language over another when teaching the operating systems class. Sattar et. al. advocated for the use of operating systems assignments

in Java. They stated that they gave students a set of Java programming assignments that required simulation of the behavior of process management, process synchronization, memory management, and storage management [13]. Robert Sheehan argued for the use of the dynamic programming language Ruby to be used in operating systems courses. His primary stated reason for using Ruby was the ease with which Ruby provides access to UNIX commands and system calls. From reading his paper the actual reason seems to be that Sheehan just really likes Ruby and wanted to include the Ruby programming language in as many classes as possible [14].

Other operating system educators advocate for the importance of one operating system platform over another when teaching the operating systems course. In 2005, Nieh and Vaill argued that having students modify the kernel was of utmost importance and thus argued for a Linux based system using virtual platforms [8]. Just seven years later, Andrus and Nieh argued that the computing landscape was shifting towards mobile devices and advocated for kernel level experience using the Android operating system [1].

The *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science* is the most current undergraduate computer science curricular recommendations from the ACM. Under the section for operating systems, the guidelines recommend a minimum of four core Tier1 hours in operating systems overview and principles. The guidelines also recommend eleven core Tier2 hours in the areas of concurrency, scheduling dispatch, memory management, and security protection. The guidelines also describe six elective areas that include virtual machines, device management, file systems, real time embedded systems, fault tolerance, and system performance evaluation. The guidelines also make a strong point that the knowledge area of operating systems is structured to be complementary to four other knowledge areas. These areas are systems fundamentals, information assurance security, networking communication, and parallel distributed computing. This acknowledgement of overlap mirrors the concerns raised by Su Yun-Lin nearly a quarter of a century earlier and is consistent with the curriculum guidelines recommending knowledge areas and not any specific course [9].

Finally, I would like to conclude this abbreviated review of operating systems education history with a look at two papers that sought to examine the teaching of the operating systems course from a distinctly pedagogical viewpoint. In 2003, Hill et. al. wrote an extraordinary paper on the gamification of various operating systems concepts. Their stated purpose was to reach students that had different learning styles. They described two specific games for the operating systems class: a modification of the popular Hasbro Battleship game called BattleThreads and a process state transition game [5]. In 2014,

Webb and Taylor, described their work in creating a pre- and post-course concept inventory that they used to explore students' misconceptions of operating concepts with an attempt to discover how various approaches to teaching the operating systems class affected learning [15].

3 Reasons Why

I have organized my reasons on why we should be teaching operating systems at the undergraduate level into three broad categories. These categories are (1) for students to have a deep and clear understanding of the operations of computers and computer systems, (2) expectations from future employers regarding what a student should learn in a computer science degree program, and (3) the sheer joy of learning something that is complex.

3.1 Computers Are Not Magic

For many students the operating systems class is the first time they are exposed to the concept and reality of how hardware and software work together. Engineering doesn't just happen and the operating systems class is ideally situated to illustrate that point. For example, teaching students what happens from the time the power is physically turned on until a user sees the graphical user interface is often quite revelatory. Having students explore this concept in depth helps them gain a deep and rich understanding of the material. As informatics professor Juraj Hromkovic has pointed out "teaching computer science is a chance to introduce engineering as a highly creative, constructive activity to our educational system" [6].

The implications for how we teach based on this reason is that it may not matter which specific system we teach but it would be important that students have the ability to thoroughly investigate a system. The approach of using an open source operating system that could be modified would seem ideal. However, the approach of teaching by experimental design as advocated by Downey would also seem to satisfy this justification for teaching operating systems [2].

3.2 Workforce Expectations Of A Computer Science Graduate

The reality is that most of our computer science graduates will not be working for a company that builds and designs operating systems. If our graduates pursue careers in software development they will be coding applications that run on an operating system. This is where knowledge gained from an operating systems class is critical. In order to optimize the performance of their

applications, it is critical that they have a robust understanding of the underlying systems upon which the applications will be running. How the operating system will affect the execution of application software is critically important. Software companies are also often concerned with their software being able to run on multiple platforms. Students should be at the very least be able to write and execute software on the following operating systems platforms: Android, iOS, Linux, and Windows.

Being able to understand the limitations of software may be just as equally important. The idea of understanding what algorithms cannot do is fully explored by David Harel in his book *Computers Ltd. What They Really Can't Do* [4]. I believe this idea can be extended more pragmatically to what is realistically possible given the constraints of an operating system. The implication for how we teach based on this criteria is that we need to cover the major operating systems that exist and give students hands on exposure to as many different systems as possible. If this is not possible in the single operating systems class, then an undergraduate degree program can seek to include this learning outcome in other places throughout the curriculum.

3.3 Joy Of Learning

The last position that I wish to strongly advocate as a reason to why we should teach operating systems is that learning about operating systems is a lot of fun. As educators, I think that we sometimes get so busy with all the work we need to do that we forget how incredibly joyous it is to learn about systems and develop a deep understanding of a complex subject. We can all too easily overlook the thrill of programming an operating system for the very first time. It's been more than 30 years, but I still remember making source code corrections to Andrew Tannenbaum's MINIX system and being able to test and observe how the operating system actually changed on an IBM PC clone. It was that spark of joy that kept me going to actually learn and continue to learn about computer operating systems.

My big advice for operating systems teachers is to be enthusiastic. It can be very daunting for students encountering this material for the first time. It can be very challenging for our students to understand the importance of all they are learning until after the class is over. If somehow we can convince our students the importance of the material via our enthusiasm, I think we will have taken a very important step in helping our students become the computer scientists they wish to be.

References

- [1] Jeremy Andrus and Jason Nieh. Teaching operating systems using Android. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, SIGCSE '12, pages 613–618, New York, NY, USA, 2012. ACM.
- [2] Allen B. Downey. Teaching experimental design in an operating systems class. In *Proceedings of the 30th ACM Technical Symposium on Computer Science Education*, SIGCSE '99, pages 316–320, New York, NY, USA, 1999. ACM.
- [3] Joel Gyllenskog. Teaching operating systems design. *SIGCSE Bulletin*, 10(2):44–46, 1978.
- [4] David Harel. *Computers Ltd. What They Really Can't Do*. Oxford University Press, New York, NY, USA, 2000.
- [5] John M. D. Hill, Clark K. Ray, Jean R. S. Blair, and Curtis A. Carver Jr. Puzzles and games: Addressing different learning styles in teaching operating systems concepts. In *Proceedings of the 34th ACM Technical Symposium on Computer Science Education*, SIGCSE '03, pages 182–186, New York, NY, USA, 2003. ACM.
- [6] Juraj Hromkovic. Homo informaticus - why computer science fundamentals are an unavoidable part of human culture and how to teach them. *Olympiads in Informatics*, 10:99–109, 2016.
- [7] J. Lions. *A Commentary on the Sixth Edition UNIX Operating System*. self published, 1977.
- [8] Jason Nieh and Chris Vaill. Experiences teaching operating systems using virtual platforms and Linux. In *Proceedings of the 36th ACM Technical Symposium on Computer Science Education*, SIGCSE '05, pages 520–524, New York, NY, USA, 2005. ACM.
- [9] Joint Task Force on Computing Curricula: Association for Computing Machinery (ACM) and IEEE Computer Society. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. ACM, New York, NY, USA, 2013.
- [10] Saverio Perugini. Developing a contemporary operating systems course. <https://sites.google.com/a/udayton.edu/operatingsystems/>.

- [11] Saverio Perugini. Teaching operating systems community of practice. <https://saveriooperugini.github.io/Teaching-Operating-Systems-Community-of-Practice/>.
- [12] Saverio Perugini and David J. Wright. Developing a contemporary operating systems course. *Journal of Computing Sciences in Colleges*, 34(1):155–156, 2018.
- [13] Abdul Sattar, Lee Mondschein, and Torben Lorenzen. An operating systems course with projects in Java. *ACM Inroads*, 1(2):24–26, 2010.
- [14] Robert J. Sheehan. Teaching operating systems with Ruby. In *Proceedings of the 12th ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '07, pages 38–42, New York, NY, USA, 2007. ACM.
- [15] Kevin C. Webb and Cynthia Taylor. Developing a pre- and post-course concept inventory to gauge operating systems learning. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 103–108, New York, NY, USA, 2014. ACM.
- [16] Su Yun-Lin. On teaching operating systems. *SIGCSE Bulletin*, 21(3):11–14, 1989.

Adding a Syntax Macro Facility to *iGen**

Larry Morell¹ and Xin Wan²

¹*Computer and Information Science*

Arkansas Tech University

Russellville, AR 72801

lmorell@atu.edu

²*Microsoft Corporation*

Redmond, WA 98052

wan.xin@microsoft.com

Abstract

iGen is a system for specifying and building programming language interpreters. *iGen* has been enhanced to include a macro facility that enables the underlying grammar to be extended by the programmer when the new construct can be defined in terms of existing constructs. We call this a *syntax macro* facility because the macros are defined via new grammar rules, which are implemented by the *iGen* parser. Each macro declares a new syntax rule and gives its meaning in terms of code in the programming language. *iGen* implements macro definitions by extending the parser to recognize the new construct. When an *iGen* parser encounters the new syntax it transforms the parse tree of new construct into a tree that corresponds to the semantics given in the macro.

1 Introduction

iGen is a system for implementing programming language interpreters. The principal goal of *iGen* is to redesign the structure of interpreters to ease the effort needed to extend a programming language. In a traditional implementation of an interpreter adding rules to the programming language requires changing code in several modules, including the scanner, parser and evaluator.

*Copyright ©2020 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

In contrast, an *iGen* interpreter stores all information related to each grammar rule in a corresponding class. This makes it easier to extend and contract the language as needed in a manner consistent with recommendations by Parnas [8]. For this reason *iGen* is described as being extensible [7].

An interpreter created with *iGen* begins by loading its grammar rules found in each of the grammar rule classes. Adding new grammar rules therefore requires defining one or more classes for the new rules and adding them to the list of classes to be loaded. Extending the interpreter is therefore limited to those who know the structure of *iGen* and its implementation language, Java.

A macro facility is described here that enables an *iGen*-implemented interpreter to be extended without having to know its internal structure. We call these macros *syntax macros* to distinguish them from the macros found, for instance, in the C preprocessor or in Lisp. After describing the background necessary to understand *iGen* and the limitations of conventional macros, syntax macros are defined and discussed. The paper concludes by noting limitations and possible applications of syntax macros in university courses.

2 Overview of *iGen*

iGen assists with the construction of interpreters. *iGen* reads a specification of the programming language and writes a collection of classes that implements an interpreter for the language. The input file specifies a language's syntax via BNF rules and the semantics of each construct via Java code. Details of this process are found in [7] and is summarized below.

Each grammar rule is translated into a Java class. That class includes four elements: (a) the grammar rule, (b) the parser for parsing the grammar rule, (c) a method for building an abstract syntax tree (AST) from the parsed right-hand side of the grammar rule, and (d) a method for evaluating the AST. *iGen* supplies a recursive-descent parser that the language implementer can use or replace.

For example from the rule `<exp> ::= <term> + <exp>` *iGen* generates a Java class called `ExpAddOp`:

```
public class ExpAddOp extends ELSEInstruction {
    private static String rule = "<Exp> ::= <Term> <AddOp> <Exp>";
    private static SyntaxRule syntaxRule =
        new SyntaxRule(GRAMMAR, rule);
    public static void initialize() {
        GRAMMAR.addRule(syntaxRule,
            new ExpAddOp(), new enericParser(syntaxRule.lhs()));
    }
    public ExpAddOp(){ }
```

```

public Instruction createInstruction(InstructionList rhs) {
    // Code supplied in the input file
}
public Value eval(EnvironmentVal env) {
    // Code supplied in the input file
}
}

```

All the grammar classes inherit from an `Instruction` class that corresponds to the language being interpreted, in this case a small language called `ELSE`. Note that the grammar rule is supplied by a string; `addRule` compiles the supplied rule and inserts it along with the required parser into a trie that holds the entire grammar and directs the recursive-descent parser. The parser creates an instruction for each member of the right-hand side of a rule and passes the completed instruction list to `createInstruction`, which, in this case, stores a reference to it as `il`, the instruction list that stores the references to the children of the AST rooted here. The code found in `eval` interprets the resulting AST, using the environment `env` to look up values of variables. Every generated class has a similar structure, simplifying the specification of these classes in the input file. `iGen` builds the parser by calling `initialize` for each grammar rule of the language. It then reads the input file and parses it using the parser supplied for the start symbol of the grammar. The resulting AST is then interpreted using the evaluation routine associated with the start symbol. Details can be found in [7].

3 Textual Macros

Textual macros [1] have a long history of enabling text-to-text transformations. A textual macro generally has two parts: a pattern to be matched and a transformation to be performed on the matched text. For example, in the C preprocessor the pattern to be matched in the macro `define PROD(x,y) x*y` is `PROD(x,y)` and the transformation is `x*y`. Thus, when the processor encounters `SUM(a+b,c+d)` it expands it to `a+b*c+d`.

Note two things in this example. First, the invocation of a C preprocessor macro has a fixed syntax of `MACRONAME(p1,p2,...)`; without this the preprocessor cannot discover a macro invocation. Second, this macro, despite appearances, does not produce code that multiplies `x` and `y`. The programmer must insert parentheses to reflect that both `x` and `y` should be computed before the multiplication. This is because the preprocessor knows only the syntax of its own macro language and nothing of the syntax of the underlying language (in this case C). Additionally, nothing prevents the user from writing macros that generate nonsense, and that will not be discovered until parse time, at

which point the error message may be given in terms of the expanded text and not the original macro.

In Lisp-like languages, a macro definition includes the name of the macro, a parameter list, an optional documentation string, and a body of Lisp expressions that defines the new form to be represented by the macro. It is invoked in the same manner as every other Lisp function, namely with Cambridge prefix notation ((function param ...)). Macro invocations therefore appear to be native Lisp[4].

The following defines a macro that increments a value by 2:

```
(defmacro plusPlus2 (num) (setq num (+ num 2)))
```

The pattern is given by `plusPlus2 (num)` which is transformed into `(setq num (+ num 2))`. When evaluating the expression `(plusPlus2 x)`, the interpreter identifies a macro expression by its first argument, returning the expansion of the macro call as a new Lisp form `(setq x (+ x 2))`, which is then evaluated in place of the original form.

It should be noted that requiring the macro to be expressed in Cambridge notation means the that macro is not actually extending the syntax of the language which continues to use Cambridge notation.

4 Syntax Macros in *iGen*

It seems useful to explore how macros can be used to extend the syntax of a programming language. This paper calls such macros *syntax macros* following earlier suggestions by both Cheatham [2] and Leavenworth [5], who formulated macros that specify syntax-directed transformation on syntax trees, rather than on phrases of text. This imbues the macros with knowledge of the syntax of the programming language, which is a deficiency of textual macros. Their formulations, however, required specialized syntax for the preprocessor to recognize a macro invocation. This limitation is removed here.

The syntax macro scheme described here extends the syntax of a language and the meaning of that syntax by code written in the language. *iGen* uses the macro to modify the existing parser and evaluator. Thus, recognition of a macro invocation is done by the *iGen*-generated parser as part of its normal parsing action and evaluation of the resulting parse tree is accomplished by the normal operation of the evaluator.

A macro definition in *iGen* has the following format:

```
#MACRO_RULE
// New grammar rule for the language
#MACRO_BODY
// The meaning of the grammar rule in existing language syntax
```

```
#MACRO_RESULT
// The result (if any) to be returned after executing the body
#MACRO_END
```

Macros are available from the point of their introduction to the end of the program. The pattern for discovering a macro is given by the macro rule which is used to enhance the parser. This, of course, requires the language extender to know the syntax of the language, which is not unreasonable. The transformation is given by the macro body and the macro result. The macro processing ensures the code for the macro body is executed and, if present, the code for the return result. This allows an *iGen* syntax macro to function like a either a procedure or a function.

When *iGen* encounters a macro definition, it builds an instance of a class called `Macro` that stores the components processed from the definition. The grammar rule is stored as a string. The macro body and macro result code are stored as syntax trees.

The following macro definition increments a variable and returns the incremented result:

```
#MACRO_RULE
<Factor> ::= ++ <id>
#MACRO_BODY
<Statement>
$id1 := $id1 + 1
#MACRO_RESULT
<Factor>
$id1
#MACRO_END
```

When *iGen* encounters this macro definition it extracts the grammar rule, the macro body and the macro result and passes them to a constructor for `Macro`. The constructor adds the rule to the programming language’s grammar (via `GRAMMAR.addRule(...)`), parses the macro body as a `<Statement>` and stores the resulting syntax tree. It then parses the macro result using `<Factor>` as the goal to produce produce another syntax tree. These are called *skeleton trees* because they contain one or more placeholders in lieu of a fully developed tree. In this example, `$id1` acts as a placeholder in each tree, indicating where a parsed `<id>` is to be placed when a `++ <id>` is parsed. The elements of the right-hand side of the grammar rule are numbered starting at 0, so `$id1` references the second element of the right-hand side, namely `<id>`.

Portions of a macro that can be inferred may be omitted. For example, to add a “half” statement into the language so that `halve n` translates to `n := n/2`, the indication that the body is a single `<Statement>` can be omitted, as well as the macro result, which defaults to the the result of a `<Statement>`:


```

MACRO_RULE
<Statement> ::= halve <id>
#MACRO_BODY
$id1 := $id1 / 2
#MACRO_END

```

Parameters to macros may be macro invocations. Given the macro

```

#MACRO_RULE
<Factor> ::= double <Expression>
#MACRO_BODY
<Expression>
2 * $Term1
#MACRO_END

```

the invocation `x := double double 3*4` yields 48. Similarly, macros may invoke other macros in their macro body or macro result sections. Thus a macro for quadruple may be written as

```

#MACRO_RULE
<Factor> ::= quadruple <Expression>
#MACRO_BODY
<Expression>
double double $Term1
#MACRO_END

```

Finally, macros can be created to support infix notations such as

```

#MACRO_RULE
<Print> ::= print <Expression> and <Expression>
#MACRO_BODY
<StatementList>
print $Expression1
print $Expression3
#MACRO_END

```

This enables two expressions to be printed via `print a+5 and b-10`.

5 Discussion

We have described a means of implementing syntax macros for extending the syntax of a programming language. The meaning of the extension is given by writing code in the programming language being interpreted. This enables a user of the language to extend its structure and not just the language implementer. The implementation is made possible by the fact that *iGen* builds its parser by loading grammar rules dynamically.

Previous approaches to macros used fixed syntax for the macro invocation to make it easier to identify a macro invocation. This is similar to using the fixed syntax of functions and procedures in programming languages as an approximation to language extension. For example, writing `insert (elem, n, lst)` to mean “insert element `elem` at position `n` in list `lst`”, we have simulated extending the language to have a new construct: `insert()`. But when reading the new syntax, does `n` represent a position or a value? Is the `insert` before, after or at position `n`? Of course one can embed comments within the parameter list to remind the reader of the meaning, but that requires extra discipline on the programmer’s part. Using a syntax macro one may extend the syntax by adding a rule such as `<stmt> ::= insert <exp> into <factor> after position <no>` to the grammar. This may not be particularly useful for professional programmers, but it is quite helpful for students learning to write algorithms (and programs). This observation motivated an elementary form of syntax extension in the algorithm language *Genesis*[6], which allows functions and procedures names to be written in infix, with parenthesized parameters interspersed among id’s.

There are limitations to the current system. Syntax macros can only introduce new syntax that can be described in terms of existing syntax. Macros cannot be recursive. Currently an macro invocation can produce a variable that clashes with existing variables. Adding a facility for conditional expansion of macros would be useful. Precedence in grammars makes it difficult to envision how a macro facility could be designed to enable new levels of precedence to be inserted into a language. More work is needed in these areas.

6 Conclusion and applications

The *iGen* interpreter generator has been extended with syntax macros. Such macros allow a programmer to extend the syntax of an *iGen*-implemented language without having to know how to modify the underlying scanner, parser, evaluator or data structures of the interpreter.

Several potential advantages accrue from this work. In an introductory course an instructor can replace arcane function calls with more palatable syntax tailored to the project at hand, easing the transition from algorithms to programs. For a course in programming languages, students can extend a language with new constructs, thereby reinforcing their knowledge of grammars and semantics. Students in a compiling course can be challenged with discovering the strengths and weaknesses of syntax macros by implementing a minimal language using *iGen* and then comparing the relative difficulty of extending their language using syntax macros versus modifying the interpreter. The implementation of the syntax macro facility in *iGen* can be used to reinforce

good software engineering concepts involving design and reuse as discussed in [7]. Last, the *iGen* macro facility could be used in a computational theory course to extend a minimal language using macros, as discussed in [3] to investigate language hierarchies.

References

- [1] Robert Adams. Take command: The M4 macro package. *Linux J.*, 2002(96):6, April 2002.
- [2] T. E. Cheatham. The introduction of definitional facilities into higher level programming languages. In *Proceedings of the November 7-10, 1966, Fall Joint Computer Conference, AFIPS '66 (Fall)*, page 623–637, New York, NY, USA, 1966. Association for Computing Machinery.
- [3] Martin Davis and Elaine Weyuker. *Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science*. Academic Press, San Diego, California, 1983.
- [4] D. Kevin Layer and Chris Richardson. Lisp systems in the 1990s. *Commun. ACM*, 34(9):48–57, September 1991.
- [5] B. M. Leavenworth. Syntax macros and extended translation. *Commun. ACM*, 9(11):790–793, November 1966.
- [6] Larry Morell. Algorithms in Genesis. In *Proceedings of the 2nd Annual Conference on Mid-South College Computing, MSCCC '04*, page 5–16, Little Rock, Arkansas, USA, 2004. Mid-South College Computing Conference.
- [7] Larry Morell. Design of an extensible interpreter using information hiding. *J. Comput. Sci. Coll.*, 26(5):130–136, May 2011.
- [8] David Parnas. Designing software for ease of extension and contraction. In *Proceedings of the 3rd International Conference on Software Engineering, ICSE '78*, page 264–277. IEEE Press, 1978.

A New Face for Old Moses: An Exercise in Swift and C Interoperability*

Robert England
Computer Science Program
Transylvania University
Lexington, KY 40508
rengland@transy.edu

Abstract

This paper describes the author's continuing experiences and revelations related to the development of an iPad app in Swift to provide a graphical interface for an existing C software package, the Moses Operating System environment simulator.

1 Introduction: Overview of Moses System

Moses is an acronym for Microcomputer Operating System Environment Simulator. It is a software system developed by the author for use in teaching the primary concepts of Operating Systems through programming in upper level Computer Science courses. The Moses system and its features are described extensively in earlier papers [5, 2, 4, 3]. A key characteristic of the Moses system is that it is written entirely in the C programming language. While C affords students who work on Moses projects the opportunity to delve deeply into low level systems programming, directly manipulating values and addresses as bits and bytes, it is not particularly conducive to a user friendly interface.

The Moses software package given to students who will work on Moses projects consists of two major components. The first component implements Moses hardware simulation. Students working on Moses projects write a small kernel program to run on this simulated hardware platform. The simulated

*Copyright ©2020 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

hardware includes five general purpose registers and a program status word (PSW) that consists of a program counter, a timer, and a variety of individual bit flags. As would be the case in a real system, the program counter holds the address of the next instruction to execute, the timer can be used by the kernel to implement preemptive timeslice-based scheduling, and flag settings can be checked by the kernel to discover what sorts of interrupts and system calls occur. A special flag bit can also be set to have the processor execute in user mode or supervisor mode.

The other major component of the Moses package is a collection of simulated user processes. Each of these processes is specifically designed to be run by a Moses kernel simulator. The user process simulator package only performs activities, generates simulated interrupts, and makes system calls that a correctly implemented Moses kernel can handle. Thus, the Moses project that a student writes is a kernel simulator that is sandwiched in between the Moses simulated hardware below, and the collection of simulated user processes above [8].

The interface development project presented in this paper has thus far been implemented for Moses Project 1. In Project 1, students write a kernel that implements user process management, including process scheduling. To this end, a student's kernel must implement and maintain a simple round-robin process scheduling queue data structure, context switching among user processes, and a process table data structure. Each process table entry corresponds to one user process, and it includes save areas for general register values, the program counter value, the timer value, and several other process specific accounting details such as the process name and its current state.

In addition to implementing process scheduling, a Moses Project 1 kernel must also recognize and handle a variety of simulated interrupts and system calls, such as timer interrupts and user process requests for output.

Current State of the Moses Interface

As of this writing, the most recent stable release of Moses is still strictly a console application, as shown in Figure1. Any input or output communication whatsoever between a running Moses system and a user / student developer appears as plain text in a console window. A student's kernel project must print all of its own debugging info about current hardware status — such as the general register values and the various fields of the PSW — and current process status, such as the process name. While running Moses as a console application is not necessarily a problem in itself (students don't complain, much), the technical details of retrieving and formatting the wealth of pertinent hardware and user process data values for screen display eats up precious time in a one semester class, and thus it hampers development of more advanced kernel functionality.

```
Kamb — a.out — 91x37
Loop entered. Current is 2.
Shared hardware restored from process CHANDLER.
    Timer is set to 116.

PC is set to 3024450.
Control being handed to process CHANDLER...

DEBUG: Before dec_timer, timer is 116
DEBUG: Leaving dec_timer, timer should be 103
Control has been returned.

Shared hardware saved to process CHANDLER.

CURRENT STATE OF SCHEDULING QUEUE:
[2 -> 1 -> 0 -> 5 -> 4 -> 3 -> ]

Checking interrupts...
    Which interrupt is it?
    Interrupt 9
    get_syscall() returns 10.
    System Call 10.

*****
                Getting first element of Squeue*: 2.
Loop entered. Current is 2.
Shared hardware restored from process CHANDLER.
    Timer is set to 103.

PC is set to 3024530.
Control being handed to process CHANDLER...

[begin MOSES output:]
    CHANDLER: I put 40 in Tx, 50 in Rx, and Tx*Rx in Ux. Ux current value: 0x7d0
[:end]

[press Enter]
```

Figure 1: The Moses console interface

2 Project Goal: Build a Better Interface

The goal of the upgrade currently under development is to provide a modern, intuitive GUI dashboard for automatically displaying the hardware and user process details for a running Moses system. This would be an invaluable tool for students to use as they test and debug their Moses kernel projects, freeing them to focus less on the tedious formatting of output and more on the functionality germane to the domain of Operating Systems services. However, the Moses simulator components and student-written kernel are all entirely written in C, which is not good for developing GUIs.

Thus, a wholly new approach to an interface for the Moses system is now in the works: Using Swift, the language of Apple iOS apps, the author is developing an iPad app that serves as a GUI dashboard shell on top of the

Moses C code.

To fully understand the nature of the undertaking of designing and implementing a dual-language software system, especially when the languages are so very different, we must first consider the development philosophies of the languages involved.

The C Programming Language

As higher level languages go, C is very low level [7]. It is completely comfortable with direct manipulation of data as bits and bytes. It considers numeric addresses of memory locations as just another data type, and it has no problems with the access and use of these addresses in code. For this reason, from a more modern higher level language perspective, C is not considered “safe”. It allows code to go on search and destroy missions throughout memory, either accidentally or maliciously, as long as the requisite addresses are known.

The Swift Programming Language

Swift is still a quite new language [6]. It was introduced in 2014 by Apple as a modern alternative to the older Objective-C language for use in the development of apps for Apple devices. As such a new language, Swift embodies a great many of the solid design principles and paradigms of good development practice that have emerged over the years. For example, Swift culture is fully aware and supportive of the Model-View-Controller design pattern for software development. Relevant to the endeavor described in this paper, Swift also insists on use of its somewhat abstract layer of protection against raw data and memory address manipulation. In that sense, the designers of Swift worked hard to ensure that Swift would be a “safe” language.

Swift software development philosophy, generally speaking, seems to prefer that a programmer not be aware that bits and bytes and memory addresses actually exist. For example, even though Swift does in fact support a data type for pointer-to-integer, `UnsafeMutablePointer` (notice the implicit rebuke in the name!), this type is actually a structure rather than a C-style fundamental data type, and the actual hexadecimal pointer value itself is a field within this structure, hidden from direct access by Swift code. According to the Swift paradigm, all functionality should be accomplished through a much higher level of abstraction with respect to the actual computer hardware.

However, Swift’s insistence that development be safely bounded within its high level syntax and use its extensive and elaborate set of available prepackaged frameworks make it a fantastic language for its intended purpose: The rapid development of GUI based applications. Swift is fully at home and comfortable with interface-centric apps for GUI mobile devices such as iPhones and

iPads [1].

3 Worlds Collide: Swift and C, Together

Arising from the very different coding philosophies and indeed, even different levels of abstraction with respect to hardware, the author has encountered several challenges in accomplishing interoperability between the two languages Swift and C in a single Moses software system.

Data Communication Concerns

The first major issue, at no surprise to the author, was how to pass data between the two languages. Swift is strongly typed, and mixing and matching data values of different types is strictly disallowed. In sharp contrast, C views the types that appear in variable definitions to be little more than indications of the number of bytes that these variables should occupy in memory.

After several frustrating trial and error experiments to communicate data back and forth between Swift and C, the following (somewhat hacky) protocol was established: Moses C defines two different arrays of bytes: one to hold values that are to be viewed as numbers, and another one to hold values that represent textual data (i.e., letters and words). The Moses C code passes the locations of these two arrays to Swift whenever an update to the GUI dashboard on the iPad is desired. This communication connection of Moses code to Swift code is accomplished via exactly one function call from a C function in Moses to one Swift function on the Swift side. These two functions, one on the C side and one on the Swift side, handle all of the communication between the two subsystems.

On arrival in the Swift communication function, the data in the two arrays from C is immediately copied into two different arrays on the Swift side. Having all of this data sent from Moses stored locally in arrays that are native to the Swift code and declared with Swift syntax makes it easier to pick through the data and convert pieces of it to the data types expected by the various view controllers on the Swift side. In the MVC vernacular, the pair of copied data arrays sent from Moses C to Swift forms the entire base of data used as the Model on the Swift side.

The very first of the number array elements indicates one of several different types of updates for the iPad dashboard display, based on the type of event that has just occurred and been recognized by a kernel program in the Moses C code. On the Swift side, when such a call from Moses C is received, the Cocoa Frameworks “Notification Center” is used to broadcast throughout the Swift side that a particular type of event has occurred back on the C side. Any component part of the Swift side that has preregistered its interest in a

particular type of event receives this notification sent out by the Notification Center, and then it digs through the number array and text array for the data it needs to update its part of the dashboard display.

For example, suppose an event occurs that involves a change to the simulated timer on the Moses C side. Then when the Moses communication function calls the Swift communication function, the timer view controller (and possibly others) on the Swift side would be notified that this event had occurred, and then the timer view controller would look through the Model data arrays for the values it needs to update its part of the dashboard display.

CPU Control Concerns

A less expected snag that had to be dealt with was the transfer of CPU control back and forth between the C code and the Swift code. The original plan was for the C side Moses kernel program to occasionally call a printf-like function in Swift whenever the iPad dashboard needed to be updated because of some event. This approach would have been as simple as printing to the console, and it would have left the specifications for the development of Moses C kernel code that students write for a project assignment largely unchanged. Unfortunately, while the idea is quite simple and the test implementation was straightforward, it just doesn't work.

The C side wants to run to completion, and Swift can't hang on to CPU control on its own after the iPad display has been updated — not even to wait for a simple “Continue” button touch on the dashboard. The Swift view controllers immediately return CPU control to the iOS runtime, which in turn immediately returns to the C side. On the Swift / GUI side, a multicolor pinwheel spins, waiting for a GUI touch event that it never receives, as the Moses C code runs all of its user processes to completion behind the scenes.

The “Alert” facility provided by Swift through its standard iOS framework seemed to offer some hope for stopping control on the Swift side before returning to C. However, the designers of Alerts locked down almost all aspects of the appearance and behavior of an Alert within an app, so there was no way to map this behavior to a simple, unobtrusive “Continue” button in the corner of the iPad screen. Any Alert used must appear smack in the middle of the screen, with the screen behind the Alert view grayed out. This obstruction of the hardware status and current process status completely defeated the purpose of the dashboard app, and the hopes for using Alerts to solve the CPU control problem were scrapped.

The solution to this maddening problem, for now, is that a Moses kernel must be partitioned into separate functions itself, with each function invoked in sequence from the Swift side. At the end of each of these function calls, the dashboard is updated. In other words, the entire interaction model between

the C side and the Swift side of the Moses system had to be inverted from the original plan. This Swift- calls-C approach is more invasive into the existing Moses C code than the author had hoped would be necessary, but at least it does work well.

The once single-function main program on the Moses C kernel side, in order to facilitate the necessary control over the back-and-forth execution path with Swift, presently consists of four separate functions, each corresponding to a primary stage in the execution cycle of the kernel:

- Stage 0 initializes the Moses system. The C function that implements this stage is only called once at the beginning of a Moses session.
- Stage 1 restores the hardware context of the next user process that will get CPU control, as selected by the scheduler.
- Stage 2 passes CPU control to the user process whose context was restored in Stage 1. On return from this user process, Stage 2 code saves the context of the process for its next CPU burst.
- Stage 3 determines what sort of interrupt or system call event led to the return from the process that just returned CPU control, and then Stage 3 calls the appropriate handler for the interrupt or system call.

Rather than calling these stage function directly (and thus itself having to keep up with where the C kernel currently is in its execution cycle), the Swift side calls the single function `callToMosesC` — a single entry point into the C code from the Swift side — leaving the responsibility of keeping up with the current stage to the C side. By design, this function is tiny and simple:

```
static FNPTR nextMainStage = main_stage_00;
void callToMosesC (void) {
    nextMainStage();
    allHdwrUpdate();
}
```

When `callToMosesC` is called by Swift, the `nextMainStage` function passes control to the next kernel stage to be executed, whose address is held in the static, function address variable `nextMainStage`. Each stage function is responsible for updating this global `nextMainStage` variable to the address of the next stage function just before it exits, with Stage 3 updating `nextMainStage` to the address of Stage 1's function, thus repeating the kernel cycle indefinitely until all user processes have terminated. The `allHdwrUpdate` function initiates the sequence of function calls that updates each of the views of the dashboard. For each dashboard view update, the two communication arrays are filled with the appropriate hardware status data on the C side, and then the C side calls back Swift to allow the Swift side to access the communication arrays and update the corresponding view.

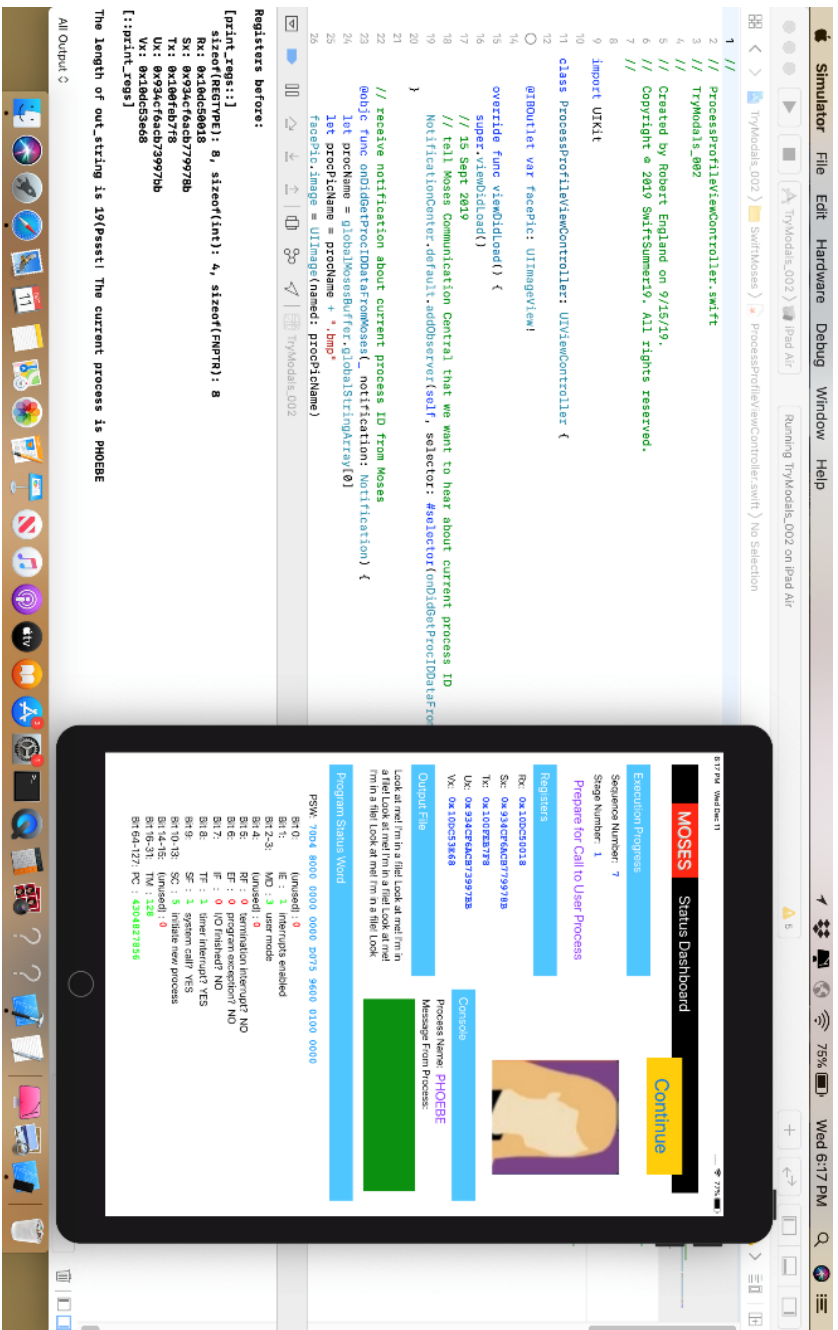


Figure 2: The Moses iPad dashboard interface, on an iPad simulator

```

1 //
2 // ProcessProfileViewController.swift
3 // TryMoses_002
4 //
5 // Created by Robert England on 9/15/19.
6 // Copyright © 2019 SwiftSummer19. All rights reserved.
7 //
8
9 import UIKit
10
11 class ProcessProfileViewController: UIViewController {
12
13     @IBOutlet var facePic: UIImageView!
14
15     override func viewDidLoad() {
16         super.viewDidLoad()
17
18         // Is Sept 2019
19         // tall Moses Communication Control that we want to hear about current process ID
20         NotificationCenter.default.addObserver(self, selector: #selector(onDidDerProcIDdataFrom
21
22         // receive notification about current process ID from Moses
23         @objc func onDidDerProcIDdataFromMoses(_ notification: Notification) {
24             let procName = globalMosesBuffer.globalStringArray[0]
25             let procIDName = procName + ".idm"
26             facePic.image = UIImage(named: procIDName)
27
28         }
29     }
30 }
31
32 Registers before:
33 [print_regs:]
34 sizeof(REGTYPE): 8, sizeof(int): 4, sizeof(FPTR): 8
35 Rk: 0x1d4dc0018
36 Sk: 0x93ac765cb779979b
37 Tk: 0x1d8f1e0718
38 Uk: 0x926c0a0c3997979b
39 Vx: 0x1d4dc0018
40 Wz: 0x1d4dc0018
41 [::print_regs]
42
43 The length of out_string is 19(Pesst) The current process is PHOEBE

```

All Output 3

4 Future Work

The iPad Moses status dashboard itself is now complete and working to a level satisfactory to the author, but it has yet to be used in an Operating Systems course. The students will have final say on the quality and usefulness of the new interface, of course. The main goal of this project remaining for the author is to find a way to connect the Moses O/S code to the Swift iOS interface so that students' Moses C kernel code does not have to be custom designed and written to enable communication with the Swift interface.

5 Conclusion

The original plan was to combine two major Moses subsystems, each written in a different language, into an integrated whole. The low level, systems programming subsystem would all be in C; the colorful candy shell iPad interface would be written in Swift. Each language would contribute what it does best to the goals of the whole system. The mixed language system as it stands now does work, but not without some compromises to the original plan and design.

References

- [1] Joe Conway and Aaron Hillegass. *iOS Programming: The Big Nerd Ranch Guide*. Addison-Wesley Professional, 2012.
- [2] Robert England. Teaching principles of shared resource management with the Moses2 microcomputer operating system environment. *Journal of Computing Sciences in Colleges*, 21(5):46–52, 2006.
- [3] Robert England. The PNTFS file system in the Moses2 operating system environment simulator. *The Journal of Computing Sciences in Colleges*, page 97, 2010.
- [4] Robert E England. The virtual machine and user process model used in Moses2: a microcomputer operating system environment simulator. *Journal of Computing Sciences in Colleges*, 17(2):301–309, 2001.
- [5] Robert E England. Teaching concepts of virtual memory with the Moses2 microcomputer operating system environment simulator. *Journal of Computing Sciences in Colleges*, 20(6):84–91, 2005.
- [6] Matthew Mathias and John Gallagher. *Swift Programming: The Big Nerd Ranch Guide*. Pearson Technology Group, 2016.
- [7] Dennis M Ritchie, Brian W Kernighan, and Michael E Lesk. *The C programming language*. Prentice Hall Englewood Cliffs, 1988.
- [8] Abraham Silberschatz, Peter B Galvin, and Greg Gagne. *Operating system concepts*. John Wiley & Sons, 2005.

Resources for Building Knowledge Base Files for Use With Named Entity Resolution/Disambiguation Tools Such As TIMBER*

Anthony D. Davis
Computer Science Department
Lyon College
Batesville, AR 72501
anthony.davis@lyon.edu

Abstract

TIMBER is a tool designed to assist scholars in Digital Humanities tag texts for entities such as persons and places. This article briefly demonstrates how TIMBER can assist such scholars by creating simplified knowledge base files compared to other industry Named Entity Resolution/Disambiguation tools. Specifically, this article provides a look at three different sources of information that a scholar may have readily available for such a task: 1) a General Index for a Multi-Volume Set; 2) Book Indices found in the domain, and; 3) Domain Specific Dictionaries. With information found in any of these three resources, scholars have all that they need to create a knowledge base for TIMBER and the ability to have a computer automatically tag persons and places in their corpora.

Introduction

Often humanities scholars desire to analyze people and places in their domain of texts. Many of these domains are made up of a large corpora of texts. Without the aid of digital humanities tools, scholars are left with manually reading

*Copyright ©2020 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

and analyzing these texts looking for connections and drawing conclusions. Fortunately, machine learning techniques such as TIMBER (Tensor Induced Multilingual knowledge Base for Entity Resolution) provide an accurate and efficient method for tagging persons and places in a large corpora [5]. While this may seem daunting to those without a computer science background, TIMBER is designed to use a simple knowledge base for named entity resolution and disambiguation. In fact, one may use an index or a collection of indexes to create their own domain specific knowledge base. This article provides three different methods of converting an index into a knowledge base suitable for use with TIMBER.

What is TIMBER?

TIMBER is the product of the dissertation titled, *Combing Texts: A Quest to Increase the Timeliness and Accuracy of Geotagging Multilingual Toponyms and Tagging Persons in Large Corpora Using Tensors for Disambiguation* [6]. This dissertation demonstrates how one can use simple knowledge bases to accurately tag domain specific persons and places in raw text. Using a comma delimited file of entities and connections, TIMBER tokenizes every word in the raw text files, connects each entity token based on co-occurrence – or how close each entity token is to each other in the text – and creates a rank-3 tensor of connections. These connections are used alongside a bayesian approach for disambiguation of entities with more than one possible entry in the knowledge base file. Figure 1 shows a visual depiction of this process. Raw text files are converted to tokenized word objects. On the left, each entry in the knowledge base file is converted to an authority token object. If the tokenized word object matches an authority token object, then it is considered a match. When there are multiple possible authority token matches for the same text token, the tensor on the bottom center is used to determine which entity has the highest likelihood of being the correct match.

In the named entity resolution/disambiguation field, the most common standard of measurement is known as an F-Score. For instance, one will find examples below where the F-Scores obtained from the TIMBER tests are compared to other named entity resolution tools. This provides an objective measurement of the accuracy between various tools in the field. Typically, they are provided in one of two formats: 1) as a percentage score, or; 2) as a floating point number that can be converted to an percentage score. The F-Score is determined by a set of equations based on how each entity token was assigned. The final equation utilizes the two other equations known as Precision and Recall. In these equations, the value “tp” is the number of true positives found when tagging the text, “fp” is the number of false positives tagged in the text,

and “fn” is the number of false negatives tagged in the text. The equations for all three are as follows [8]:

$$Precision = |tp| / (|tp| + |fp|)$$

$$Recall = |tp| / (|tp| + |fn|)$$

$$F\text{-Score} = 2 \times Precision \times Recall / (Precision + Recall)$$

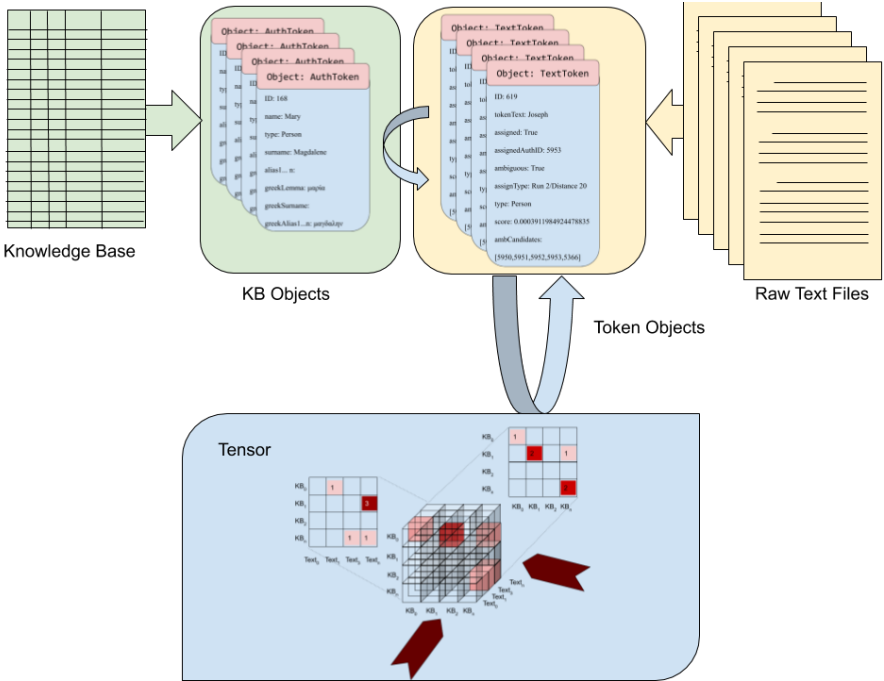


Figure 1: TIMBER Matching Entities in Raw Text with the Knowledge Base

What Is a Knowledge Base?

There are many different types of knowledge bases in named entity resolution. A knowledge base essentially functions as the brain of the process; the known entities of the domain for which one is looking. In Named Entity Resolution one typically finds very large knowledge bases used for Named Entity Disambiguation.

tion. For instance, a typical knowledge base uses Wikipedia, DBpedia entries and/or Linked Open Data formats such as Resource Description Framework (RDF) using the SPARQL query language [4]. While these knowledge bases are excellent for many projects, they may not offer the best information for domain specific texts for two reasons: 1) the large knowledge bases are heavily contemporary, and; 2) RDF knowledge bases are very technical in their formatting and usage.

For the first drawback, most Named Entity Resolution programs are interested in tagging persons and places in the contemporary world. Again, DBpedia and others are a great resource for this. However, one runs into issues when dealing with non- contemporary texts. As an example, a group of researchers considered using some typical state-of-the-art Named Entity Resolution/ Disambiguation tools on “entities in the 17th century depositions obtained during the 1641 Irish Rebellion” [8]. The researchers wanted to test current tools and knowledge bases on a very specific domain, the 1641 Irish Rebellion. To do this, they used GERBIL, which combines multiple Named Entity Resolution/Disambiguation tools into one annotator [2]. The best results from all of the tools tested presented an F-Score of around 60. The authors’ conclusion was pretty blunt: Based on this corpus and the results obtained from the General Entity Annotator Benchmarking Framework [GERBIL] we observe that the accuracy of existing Entity Linking systems is limited when applied to content like these depositions. This is due to a number of issues ranging from problems with existing state-of-the-art systems to poor representation of historic entities in modern knowledge bases [8].

The second drawback is the sheer complexity of most popular knowledge bases, such as RDF. TIMBER is designed to aid digital humanities professionals in tagging their specific corpora. Most scholars in digital humanities may not have experience in coding, XML tagging, etc. Therefore, a specification like RDF may be more than they would need or desire to learn for their domain of texts. For instance, here is an example of an RDF entry for person 8 in the digital reference portal for Syriac literature, culture, and history known as the Syriac Reference Portal:

As one can see, creating a knowledge base in RDF requires knowledge of XML and the syntax of RDF. While this is an exceptional format for a knowledge base, it may not be necessary for many domain specific projects in digital humanities. If the project has technical staff, expertise, and time, then RDF would be an excellent choice. However, TIMBER is designed to offer an alternative to scholars looking to tag their corpora using a simple knowledge base created from a comma delimited flat file. Since many scholars – regardless of their technical skill – are familiar with spreadsheet formats such as Libre Office Calc or Microsoft Excel, it is safe to assume that more scholars could create a


```

<rdf:RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#" ...
<skos:Concept rdf:about="http://syriaca.org/person/8">
  <rdf:type rdf:resource="http://lawd.info/ontology/Person"/>
  <rdfs:label xml:lang="en">Miles , bishop of Sus</rdfs:label>
  <rdfs:label xml:lang="syr">    </rdfs:label>
  <lawd:hasName>
    <rdf:Description>
      <lawd:primaryForm xml:lang="en">Miles , bishop of
Sus</lawd:primaryForm>
    </rdf:Description>
  </lawd:hasName>
  <lawd:hasName>
    <rdf:Description>
      <lawd:variantForm xml:lang="en">Miles , bishop of
Sus</lawd:variantForm>
    <lawd:hasAttestation rdf:resource="http://syriaca.org/bibl/4"/>
    </rdf:Description>
  </lawd:hasName>

```

Figure 2: Example of an RDF entry from the Syriac Reference Portal [3]

knowledge base in this fashion than using RDF, etc.

Resources for a Knowledge Base

What are some ways that a scholar can build a knowledge base from scratch? There are typically three very good resources for most historical domains: 1) a General Index for a multi-volume set of texts; 2) indices found in books, and; 3) a domain specific dictionary. Each resource provides opportunities and challenges, but most domains can start with one – or a combination – of these resources for seeding their knowledge base.

Option One: General Index for a Multi-Volume Set

One of the greatest sources of information when starting your knowledge base is the use of a General Index for a multi-volume set. Typically, an index for a multi- volume set will contain a wide variety of useful information. For instance, the General Index in the Ante-Nicene writings contains an alphabetical index of People and Places found in each of the ten volumes of the set. Furthermore, each entry in the index contains the following information: 1) volume

number(s) where the named entity appears; 2) page number(s) where the entity appears, and; 3) many connected entities mentioned in conjunction with the listed entity [1]. A General Index provides concise information on each entity and a clear view of the various connections. Although creating the knowledge base is still a manual process, this option provides the quickest route of the three options described in this article for seeding your initial knowledge base.

Option Two: Book Index

Notwithstanding the fact that a single book index is not as useful as the multi-volume set with connected entities, the typical index found in any book still contains a wealth of information that one can use to build or supplement a domain specific knowledge base. In order to make the connections, the user has two options: 1) the user may read the book and determine a connection either through co-occurrence or entity interaction manually, or; 2) the user may determine the co-occurrence programmatically based on two entities occurring on the same page or other delimiter. Obviously, the most efficient method is number two whereby the user places the entity into the knowledge base and makes a connection between two entities based on both occurring on the same page. The benefits with this case are that one can quickly and programmatically produce a knowledge base with connections for use with TIMBER. The challenge is that the granularity of the connections are based on the two occurring on the same page. What if one entity was the last word on page 22 and the other entity is the first word on page 23? In this case, there would be no connection – assuming that was the only case of the two occurring in the book.

Option Three: Dictionaries

The third scenario for building or supplementing your knowledge base is a domain specific dictionary. This option usually provides the most granular information and has a wealth of information for determining connections. Each entry has specific information on each entity and one can determine numerous connections for each based on the entries. One can even supplement the connection with more information such as gender, marital status, parents, children, etc. The benefits with this method are that there is a wealth of detailed information. This information can increase the number of connections one can use for seeding the knowledge base as well as provide information for other types of analysis. The downside to this approach is that this method takes more time to determine connections through the manual process. One dictionary entry may be multiple paragraphs long and contain a lot of entity connections. Therefore, a scholar may get bogged down with too many connections. An example for early Christian texts would be Easton’s Illustrated Bible Dictionary [7]. As

with the Ante-Nicene Fathers mentioned above, this book is out of copyright so this is a further benefit to using it for your knowledge base.

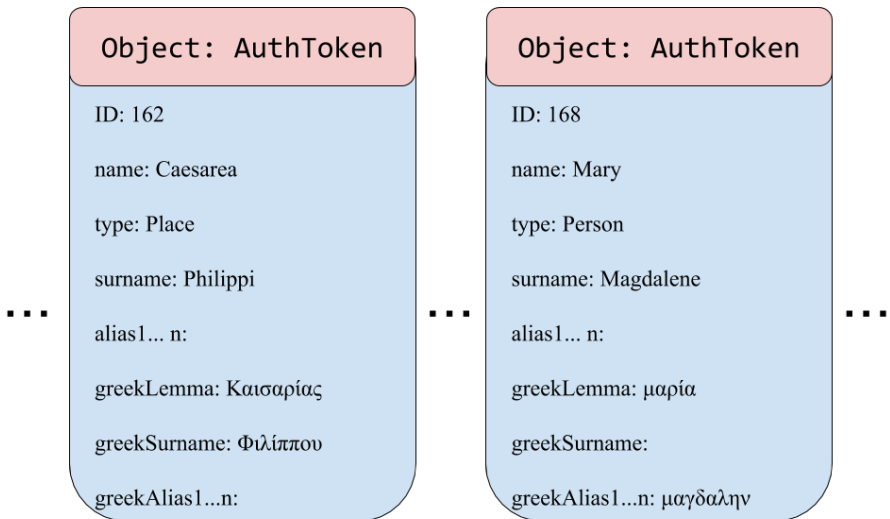
Scenario Demonstration

Precision	Recall	F-Score
0.98535	0.92737	0.95548

Table 1:
The F-Score of Volume One of the Ante-Nicene Fathers Using TIMBER

As one saw earlier, Munnely, et al compared many state-of-the-art tools on their domain specific texts. Named The best tool provided an F-Score of around 0.60. Using TIMBER, our research is able to demonstrate that a simple knowledge base is able to produce an F-Score of around 0.95, as seen in Table 1, on Volume One of the Ante-Nicene texts containing 645,579 tokens [5]. These results were obtained by creating a simple comma delimited file out of entries found in a General Index of the ten volume set Ante-Nicene Fathers: The Writings of the Fathers down to A.D. 325 [4]. This corresponds with option one mentioned above. A General Index seems to be one of the better resources to use when building a domain specific knowledge base due to the concise nature of the entries and the concise mentions of various entity connections.

After creating the knowledge base from the General Index, TIMBER generates tokens of each entity from which to compare each token in the raw text. In Figure 3, one sees two examples of entities found in the General Index, their placement into a simple comma delimited file, and a graphic depiction of the knowledge base converted into objects. This method, again, using option one above, provides an efficient and easy way for a scholar of early Christian texts to tag persons and places. However, one can use the same method on their own respective domains to tag entities in any raw text.



ID	Name	Type	Description	Surname	Alias1	Greek	GreekSurname	GreekAlias
168	Mary	Person		Magdalene		μαρία	μαγδαλην	
162	Caesarea	Place	Peter's confession	Philippi		Καισαρίας	Φιλίππου	

Figure 3: Depiction of the Comma Delimited Knowledge Base Turned Into Objects

Conclusion

Given that most domains in the humanities have numerous sources of index information, the ability to use machine learning to tag a corpus is within reach for anyone with proficiency in creating spreadsheets. One does not have to use technical formats such as RDF to create their knowledge base and each domain can find at least one of the three examples provided above to create their knowledge base for use with TIMBER. If one can find a General Index for a multi-volume set, book indices for reference works in their domain, or a dictionary for their domain, then one has all that they need to use machine learning techniques on their texts.

References

- [1] Christian classics ethereal library. <https://ccel.org/fathers> Accessed: 27-May-2018.
- [2] GERBIL — Agile Knowledge Engineering and Semantic Web (AKSW). <http://aksw.org/Projects/GERBIL.html> Accessed: 20-Jun-2019.
- [3] RDF repository for all syriaca.org data. This repository is based on the development branch of <https://github.com/srophe/srophe-app-data> and should be considered beta. : srophe/srophe-data-rdf. Syriaca.org: The Syriac Reference Portal, 2019.
- [4] Marcelo Arenas and Jorge Pérez. Querying semantic web data with SPARQL. In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 305–316, 2011.
- [5] Anthony D. Davis. TIMBER - tensor induced multilingual knowledge base entity resolution. <https://github.com/davisgis/TIMBER> Accessed November 24, 2019.
- [6] Anthony D Davis. *Combing Texts: A Quest to Increase the Timeliness and Accuracy of Geotagging Multilingual Toponyms and Tagging Persons in Large Corpora Using Tensors for Disambiguation*. PhD thesis, University of Arkansas at Little Rock, 2019.
- [7] Matthew George Easton. *Illustrated Bible Dictionary, and Treasury of Biblical History, Biography, Geography, Doctrine, and Literature*. T. Nelson, 1894.
- [8] Gary Munnely and Séamus Lawless. Investigating entity linking in early english legal documents. In *Proceedings of the 18th ACM/IEEE on Joint Conference on Digital Libraries*, pages 59–68, 2018.

A Unified Representation for Teaching Bottom-up and Top-down Parsing*

Larry Morell and David Middleton
Computer and Information Science
Arkansas Tech University
Russellville, AR 72801
`{lmorell,dmittleton}@atu.edu`

Abstract

A trie-based model for context-free grammars is described that supports teaching both LL(1) and LR(1) parsing. In the model, grammar rules and parse states are embedded in a *trie graph*, which is modified version of a characteristic finite state machine (CFSM) used in the design of LR(1) parsers.

1 Introduction

Courses on compiling typically discuss top-down and bottom-up parsing. On the surface these approaches appear to have much in common. Each is grammar-based, matching the stream of input tokens against the language's constructs. Each stores the progress of parsing on a stack. Each can be implemented via hand- or table-driven techniques. Unfortunately, the differences that lurk just below the surface can incline a student to conclude that they have little in common. The stack for a top-down parser captures the parse tree that is being built from the top down, requiring the parser to choose which rule to expand when matching the next input token. For example, the stack for a bottom-up parser consists of parse states which record progress through multiple rules, thereby delaying the decision as to which rule has been parsed. Additionally, table-based representations of top-down and bottom-up differ significantly in

*Copyright ©2020 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

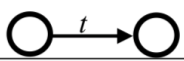
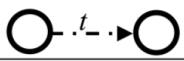
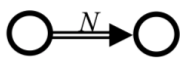
structure and content and, especially in the case of bottom-up, are not easily invertible to recover the original grammar.

These differences are exacerbated by the fact that a student can begin constructing a parser using a top-down strategy only to discover it fails and therefore a bottom-up strategy must be followed, perhaps requiring a portion of the grammar to be rewritten, data structures to be redone and a different parsing algorithm to be implemented. This paper presents a way of teaching parsing by via a unified method for representing a grammar for both bottom-up and top-down parsing.

An approach for modelling grammars via a trie graph is first described and then applied to both LL and LR grammars. An algorithm is then presented for parsing using trie graphs.

2 Modeling Parsing

Presented here is a model that allows LR and LL grammars to be processed via a single parser. The representation is called a *trie graph* because it shares aspects from both tries and graphs. Each node in the graph denotes a parser state that represents the degree to which a rule or a collection of rules has been parsed. Arcs between states direct the parser’s next action, based on the current input symbol. The parser stores active states on a stack, which is called the *parser status*. Thus, given the parser status and the current input, the parser advances to the next state based upon the action dictated in the current state, modifying the parser stack appropriately. The following table illustrates the notations used in a trie graph.

Shift arc		Advance to the next state when the input is token t , consuming the input
Advancement arc		Advance to the next state without consuming the token t
Reduction arc		Advance to the next state when a rule for the non-terminal N has been parsed and reduced

To minimize the complexity of the trie graph both the shift and advancement arcs may be labeled with a set of terminals to minimize the number of states. Both LL(1) and LR(1) grammars can be represented using only these three forms of arcs, as discussed below.

2.1 Representing LL(1) Grammars via Trie Graphs

Consider the rules:

- 1: $\langle \text{Stmt} \rangle ::= \langle \text{Print} \rangle$
- 2: $\langle \text{Stmt} \rangle ::= \langle \text{while} \rangle$
- 3: $\langle \text{Stmt} \rangle ::= \langle \text{Assign} \rangle$
- 4: $\langle \text{Print} \rangle ::= \text{print } \langle \text{Exp} \rangle$
- 5: $\langle \text{while} \rangle ::= \text{while } \langle \text{Cond} \rangle \text{ do } \langle \text{Stmt} \rangle$
- 6: $\langle \text{Assign} \rangle ::= \text{id } := \langle \text{Exp} \rangle$

Figure 1: Stmt Trie

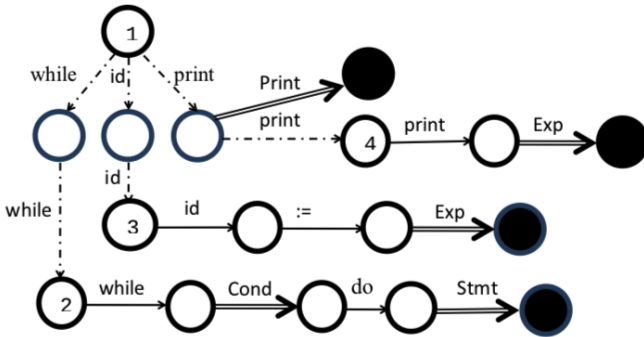
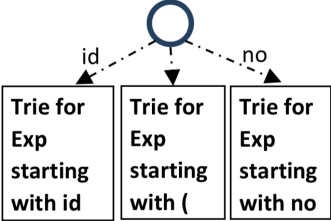


Figure 1 shows a portion of the trie graph that comes from these rules, where nodes represent parse states and filled circles indicate the end of a rule where a reduction can occur. The numbered nodes are the roots of tries that represent the rules. Parsing for a $\langle \text{Stmt} \rangle$ begins at node 1 where the decision is made for choosing to parse using either rule 1, 2, or 3. If the input is *print*, then the next state consists of the single item [$\langle \text{Stmt} \rangle ::= . \langle \text{Print} \rangle$], since *print* is only in the first set of $\langle \text{Print} \rangle$. The period in the item indicates that a $\langle \text{Print} \rangle$ must be parsed next. To do so, parsing progresses to the trie for $\langle \text{Print} \rangle$ located at rule 4. There the *print* is consumed, and the parse state is advanced to [$\langle \text{Print} \rangle ::= \text{print} . \langle \text{Exp} \rangle$]. Parsing continues at the trie for $\langle \text{Exp} \rangle$ (shown in Figure 3). After parsing an $\langle \text{Exp} \rangle$, the reduction arc for *Exp* is taken to the state [$\langle \text{Print} \rangle ::= \text{print } \langle \text{Exp} \rangle .$]. Since a $\langle \text{Print} \rangle$ has been recognized, a reduction takes place. As in a CFMSM, reducing entails backing up to the state that triggered parsing $\langle \text{Print} \rangle$. Control therefore returns to the state that contains [$\langle \text{Stmt} \rangle ::= . \langle \text{Print} \rangle$]. From there the

Print reduction arc is followed, advancing to the state that contains [$\langle \text{Stmt} \rangle ::= \langle \text{Print} \rangle .$]. Another reduction occurs there, and control returns to the state that triggered a parse for $\langle \text{Stmt} \rangle$ (not shown).

To extend this trie graph for $\langle \text{Cond} \rangle$, $\langle \text{Stmt} \rangle$, and $\langle \text{Exp} \rangle$, a subgraph is created for each rule and is linked into the existing graph with a dashed arrow, for each member of the first set of the right-hand side of the rule(s). For example, if $\text{first}(\text{Exp}) = \{id, (, no\}$ then each instance of $\langle \text{Exp} \rangle$ in any trie is linked to the trie summarized in Figure 2. The three arcs direct the parser to the state appropriate for the current input. In an LL(1) grammar any (input, state) combination can select at most one successor state. In terms of a trie graph this means that no node will have two solid arcs labeled with the same input. Instead, the target states must be merged.

Figure 2: Exp Trie



Any common prefix for two rules with the same left-hand side will overlap in the trie graph. For example, if the following rules are added to the grammar

```

<If> ::= if <Cond> then <Stmt>
<If> ::= if <Cond> then <Stmt> else Stmt
    
```

then the trie graph for $\langle \text{If} \rangle$ is enhanced to include:



An LL(1) table-driven parser does not accommodate common prefixes; trie graphs handle them nicely.

2.2 Representing LR(1) Grammars via Trie Graphs

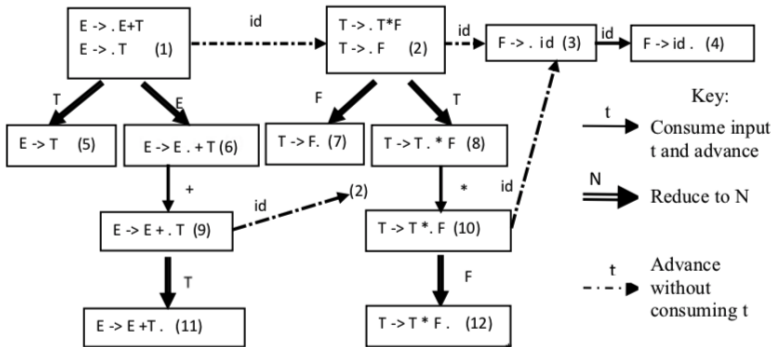
In an LR(1) grammar a state represents the degree to which one or more rules have been parsed. A canonical LR(1) state has two parts: the core and the closure. The core consists of the rules that need to be parsed and the position within those rules that indicate what may be parsed next. The closure consists of rules which may be parsed next from the positions indicated in the core. Items in LR states are stratified into separate states in the trie graph, linked by one of the three arc types mentioned earlier.

Figure 3 shows part of the trie graph for the grammar below when the input is *id*.

```

E ::= E + T | T
T ::= T * F | F
F ::= id | (E)
    
```

Figure 3: Trie Graph for Recursive Expressions



The twelve nodes in the figure 3 are labeled with items. Each *item* is a rule marked with a dot to indicate how much of the rule has been parsed already by the time a parser arrives at that state. A state with more than one item indicates multiple rules may be parsed. If the dot in every item precedes token *t*, then there is a shift arc for token *t* from this state. If a non-terminal *N* is to be parsed, then the state includes an advancement (dashed) arc to the trie for *N*, labeled with the one or more terminals that are in *first(N)*. Additionally, there is a reduction arc for *N* from this state. A sequence of dashed arcs labeled with *t* must terminate in a state where *t* is consumed.

Consider parsing a^*b using the trie graph in Figure 3. Starting in state 1, the parser advances to node 2 and then state 3, without consuming a . The a is then consumed, moving to node 4, where a reduction to F occurs. The parser unwinds to node 2, where it can advance across an F , taking it to node 7. There a reduction to T occurs and the parser unwinds to node 2 again. The parser must now decide between advancing across the T to state 8 or returning T to state 1. Since the next input is a $*$, the parser should advance to state 8, consume the $*$, and move to state 10. The parser again parses b as an F (in the same manner as it did for parsing a), and then moves to state 12, advancing the F . Reducing to a T , the parser unwinds to state 2. Again, it must decide whether to advance with the T or return it to state 1. Since there is no incoming $*$, the parser returns the T to state 1. Advancing the T leads to state 5, where an E is recognized and returned to state 1. Since there a $+$ sign is not upcoming, parsing completes having recognized a^*b as an E .

2.3 Parsing using a Trie Graph

This section demonstrates how to parse both LL(1) and LR(1) grammars via a recursive algorithm directed by a trie graph. The arcs emanating from each node select the next transition to be performed by the parser. Parsing using a trie graph is guided by three observations. First, a shift arc is taken when its label matches the next input. Second, when reducing a rule for non-terminal N , an empty parse tree for N is returned and populated as the parser unwinds. Third, after a reduction to N , the parser backs up until it can successfully advance N in a state.

```

Tree parse(Symbol sym, State s) {
  Tree tree, component
  if there is a solid arc with label sym from s to s' then
    tree = parse(readSym(), s') // get the portion of tree after sym
    Insert sym as first child of tree under the root
    return tree
  elseif there is a dashed arc with label sym from s to s' then
    component = parse(sym, s') // get the tree anchored at Symbol sym
    N = non-terminal at the root of component
    if N cannot be advanced in State s
      return component
    else
      tree = parse(readSym(), t) // where s ==> t for N
      Insert component as first child of tree under the root
      return tree
  elseif State s has a single reduce item [ A -> alpha . ] then
    return a tree with A at the root and no children
}

```

The parser proceeds top-down and left-to-right through the grammar, as directed via the trie graph. The recursion of the first and second if-block drives the parsing downward and laterally across the grammar until a rule for a non-terminal N is completely parsed. The calls to `readSym()` move the parser through the input. As the recursion unwinds, the parse tree for N is built, and then the lateral movement again ensues via the first or second if-block.

Determining whether N can be advanced involves seeing if the next input can follow N in each state as the recursion unwinds.

This algorithm handles shift/reduce conflicts by favoring a shift; it does not handle reduce/reduce conflicts. We agree with the GNU.org comment [1] that such conflicts “usually indicates a serious error in the grammar” and should be fixed. Context for a reduce, if needed, can be passed forward as a third parameter to parse. This helps students to understand how lookaheads are used to resolve conflicts and makes an excellent student enhancement to the algorithm.

3 Related Approaches and Discussion

In his first book on data structures [7], Wirth uses syntax diagrams as a means of representing a grammar via a graph to support top-down parsing. Slivnik describes a system called LLLR that can switch between LL and LR parsing by modifying underlying LL(1) tables. He indicates that previous approaches to mixing top-down and bottom-up are not popular “because of the confusing order in which semantic actions are triggered” [5]. To achieve this his principal parser is LL; when an LL conflict is encountered a small LR parser is invoked. Morell uses tries to implement a parser generator that supports LL(1) but not LR(1) parsing [2]. Each non-terminal has a corresponding trie, which is build as the grammar rules are read. Parsing is conducted via a recursive algorithm that traverses the collection of tries top down.

Trie graphs enhance the characteristic finite state machine (CFSM) used in LR parsing. The principal differences involve the representation and processing of states. CSFM states have two components: the core and the closure. The core consists of items which specify a goal for the parser when it reaches this state. The closure consists of items which can be used to progress through one or more of the core items. In trie graphs the closure collections are stratified into separate states and the resulting states are connected using three different links to represent advancing, shifting and reducing. The associated parsing algorithm is reduced to performing the appropriate action given the current state and input. Since the parser proceeds from the bottom up, it mimics the behavior of a recursive-ascent parser [3, 6].

Trie graphs elucidate the relation between top-down and bottom-up techniques by developing a trie-based model to represent LL(1) and LR(1) grammars. Such a model helps students better understand parsing before mastering complications introduced by traditional presentations of LL and LR tables. Additionally, trie graphs maintain a property the makes recursive-descent parsers popular: a grammar directly maps to its trie graph and vice versa. It is not easy to make such a mapping in table-driven approach, especially with LR tables.

A thorough coverage of parsing requires significant time and is generally only possible in a course on compiling. Using trie graphs reduces the amount of time needed to demonstrate top-down and bottom-up parsing, enabling parsing to be covered in other courses such as data structures, algorithms, programming languages, software development, or theory of computation.

4 Conclusion

Trie graphs provide a uniform model for teaching both LL(1) and LR(1) grammars. A grammar readily maps to a trie graph which can be used to direct parsing. Additionally, trie graphs address one of the most common problems of LL(1) parsing, namely, eliminating common prefixes. They help demystify parsing.

A trie graph model provides a basis for discuss issues related to processing grammars such as computing *first* sets, *follow* sets, and *derives* [4]. Trie graphs provide a convenient model for discussing the need for these concepts and how they are used to resolve conflicts. For example, it is customary for overlapping *first* sets to be discussed when an LL(1) table is built. Such a conflict will be discovered earlier when producing a trie graph. Similarly, the need for *follow* sets arises in the algorithm for trie graphs when processing a reduction.

Trie graphs are an initial model for a grammar without committing to particular data structures and algorithms for parsing. This allows an instructor to cover parsing early and explore possible implementations, including traditional LL(1) and LR(1) table-driven approaches. Seeing the tables as implementations of tries may help explain their meaning.

References

- [1] GNU Bison - the yacc-compatible parser generator. <http://www.gnu.org/software/bison/manual/> Retrieved 23 Nov, 2019.
- [2] Larry Morell. Design of an extensible interpreter using information hiding. *Journal of Computing Sciences in Colleges*, 26(5):130–136, 2011.
- [3] Larry Morell and David Middleton. Recursive-ascent parsing. *Journal of Computing Sciences in Colleges*, 18(6):186–201, 2003.
- [4] Larry Morell and David Middleton. Using information flow to analyze grammars. *The Journal of Computing Sciences in Colleges*, page 21, 2010.
- [5] Boštjan Slivnik. Llr parsing. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1698–1699, 2013.
- [6] Wikipedia. Recursive ascent parser. https://en.wikipedia.org/wiki/Recursive_descent_parser Retrieved 7 Nov. 2019.
- [7] Niklaus Wirth. *Algorithms + Data Structures = Programs*. Prentice Hall PTR, USA, 1978.

Toward the Creation of a Personal Device Security Testbed to Aid Student Learning Objectives*

*Charles Walter and Charles Fleming
Computer and Information Sciences
University of Mississippi
University, MS, 38677
{cwwalter,fleming}@olemiss.edu*

Abstract

In this paper, we propose the creation of a testbed that will allow instructors to effectively teach security of personal devices, including wearables and smart home computing devices. To enable the construction of this testbed, we explore low cost, commercial-off-the-shelf devices that allow students to examine the security of a large variety of devices, each separated in an RF-free environment. We then show the feasibility of this method with a small-scale experiment.

1 Introduction

Personal devices, like wearables, have exploded in popularity in recent years. In 2018, 17.2 million wearables were shipped [19], making them some of the most common personal computing devices available in the world. It is predicted that, by 2021, 928.8 million wearables will be active around the world [4]. While some research has been done into the security of wearables [6][24], much of the work has been slowed by the sheer number of different wearables and the lack of consistency across the devices.

Personal devices, while individually different, do share some standard features. Most communicate through either Wi-Fi, Bluetooth, Zigbee, or Z-Wave,

*Copyright ©2020 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

with most of these communicating in the 2.4GHz range. The devices almost always include a debug port left open, allowing for attackers with physical access to get debug information and, potentially, push their own firmware [6]. Additionally, users are likely to not worry about the security of specific devices, often assuming the security is taken care of by the device manufacturer. Even when the manufacturer does focus on security in some way, there are often issues with over-privilege that are inherent in personal devices, especially those on a home wireless personal area network (WPAN) [13]. While there have been some recent attempts to secure these devices, such as [10], these methods are not in widespread use.

In this paper, we discuss the issues with performing security analyses on personal devices, such as wearables, home automation devices, and smart speakers. These issues are wide-ranging, from the startup cost to a lack of understanding of how prevalent security holes are across devices. These issues make it difficult both for the researcher to discover new security issues and experiment with potential security fixes, as well as teachers that wish to make their students aware of the security issues of their own devices. We show a proof-of-concept testbed that allows users to perform security analysis on a wide variety of wearable devices, featuring existing tools capable of attacking Bluetooth communication, the primary communication mechanism used by wearables.

2 Background

The security of wearable devices has been extensively studied, both from the vulnerability and defense perspective, though only on a limited number of devices and with limited attack vectors. A good survey of this work can be found in [18].

One of the earliest testbeds for home computing devices was the “Ubiquitous Home”, which was focused on how smart home devices improved life, rather than their security [25]. A similar testbed was proposed in [12] for more general types of mobile devices, but again, primarily with a human-centric focus, through volunteer end user evaluation. Crowdlab [11] had a similar design, utilizing volunteer end users, but gave researchers control of low-level wireless state. The DRIVE testbed [19] was built with more of a networking focus, but was specifically for vehicular area networks and devices, rather than wearables or home automation devices. A similar testbed for automotive security was proposed in [14].

One of the first security-centric testbeds for wireless networks was introduced in [16], however the authors were interested in ad-hoc local area network security. A testbed for wearable medical devices was described in [9], but this testbed was designed to measure the energy use of adaptive security defenses,

rather than serve as a general testbed for device security. A more general Internet of Things (IOT) testbed was developed in [8]. This test bed included some limited security testing features, but was built for the Web of Things applications, and only tested basic authentication protocols.

Two very comprehensive testbeds for wearable security are proposed by Siboni et al. in [21] and [20]. While some of the functionality of these testbeds overlaps with our work, they are primarily designed to provide comprehensive testing of new devices, rather than the development of new attacks or for educational exercises. Pass-IoT [7] is an IoT testbed designed specifically to test lightweight encryption algorithms. In [23] the authors propose another IoT testbed which primarily captures wireless data from the devices. While not strictly a testbed, IoTScanner [22] similarly passively captures IoT transmissions, though in a mobile manner that allows for scanning of any space of interest.

A slightly different approach to the problem is to provide “meta-testbeds” that are amalgamations of smaller testbeds. This approach is taken in [15], the IoTbed system. A similar approach is taken in [24], with respect to self-adaptive systems.

3 Project Goals

The project described and explained in this paper is to create a method for teaching and research of personal use devices, such as wearables and home automation. Because of the rapid increase in market share these devices have enjoyed recently, and are likely to continue to enjoy, there is increased need to be able to examine possible security vulnerabilities and to teach the methods of security analysis of these devices to students just beginning their careers as computer scientists. Currently, the only method of examining a personal use device is to purchase it and all hardware providing possible methods of attacking it. Just attacking a single Bluetooth device can get expensive quickly and become impossible to use to teach a course.

In order to attack the device, one must purchase the device and attempt to attack it. If it uses a simple pairing method, one may be able to attack the device with a form of Man-in-the-Middle attack. This works well for devices that have low security methods, but becomes more difficult when there is either a unique method of connection (similar to the Apple Watch, which uses an image on the screen to generate a key), devices that pair using Out-Of-Band pairing (where a secondary form of communication is used to establish keys), or devices that require user input and confirmation (which may not be possible if not all devices have the same capabilities). To attack these devices, researchers must turn to eavesdropping attacks to get access to the data communication.

Currently, the best method to eavesdrop on a device communicating with Bluetooth is using an Ubertooth One [5], a device costing around \$115 as of this writing. Unfortunately, even with an Ubertooth, there is no guarantee that all packets will be intercepted or that a pairing packet (containing keys) will be intercepted, allowing for decryption.

Even if an attack vector is identified with one device, there is no way to test against a large number of other devices quickly and cheaply. Without a centralized testbed, accessible by both students and researchers, breaking into personal use device security work is difficult.

We propose the creation of a testbed designed to allow instructors and researchers access to a range of personal use devices and the methods to attack them. The instructors and researchers may either purchase the components themselves or, when complete, connect to our centralized personal use device testbed remotely to examine the security of personal use devices. In this paper, we focus on the creation of this testbed and the descriptions of the attacks that are enabled. With a fully realized testbed, instructors will become capable of demos and labs focused on the security of personal use devices, including eavesdropping, man-in-the-middle attacks, and analysis of data collected.

4 Methodology

Our design for a testbed has two primary setups for device security: one relies on a large number of devices separated into individual commercially-available Faraday cages containing a base station and its associated personal use device. The architecture of this can be seen in Figure 1. If possible, the base station, connected to the internet, will provide the packets that it is receiving to a central server. These devices will be set up for researchers and students to choose the specific devices they wish to have access to through a web portal. These devices will have their base stations set up as a pass-through to a single external base station, using a Man-in-the-Middle style attack, to simulate a normal user environment. The researcher or student can then examine all the communication flowing from the devices they have chosen to the central base station. With the Man-in-the-Middle pass-through setup, students and researchers can also modify the packets to examine how the devices react to crafted packets, allowing them to act as if they have physical access to the device.

Additionally, we will have a group of devices designed to simulate accurate user environments, all connected to a single user device. The architecture for this can be seen in Figure 2. These devices will be sequestered within a single Faraday cage and, within the cage, devices to eavesdrop on the communication and, if possible, transmit specific data from a secondary device. The device to

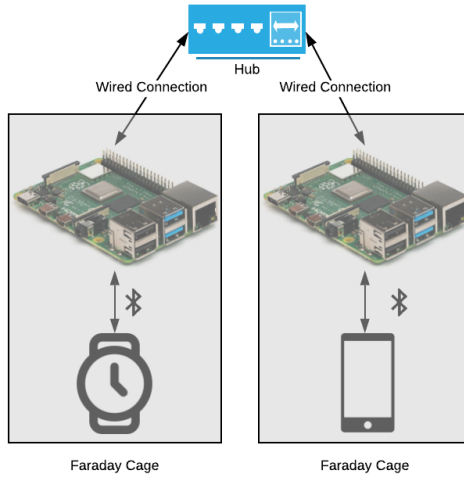


Figure 1: Proposed Testbed Architecture for Separated Devices

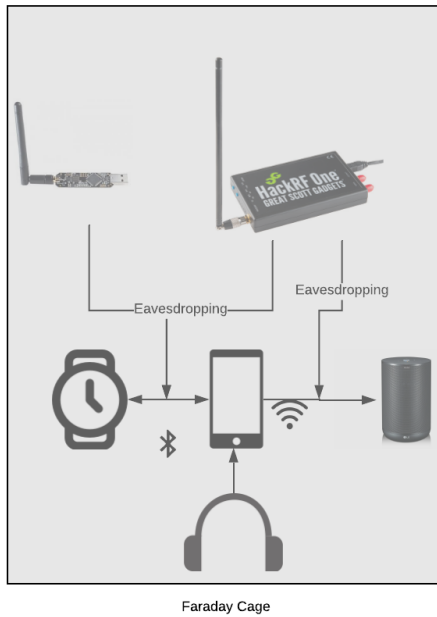


Figure 2: Proposed Testbed Architecture for Simulated User Environments

eavesdrop on the communication will be attached to a computer sending data back to a central server for researchers and students to perform additional analysis on the communication of those devices. The single user device within the cage will be controllable externally, allowing users to simulate having the devices themselves and making requests for data or sending data to the devices.

5 Results

Currently, the proof-of-concept testbed is designed primarily to attack devices owned by the attacker, though this restriction will not be present in the full testbed. To attack the communication, we have both an Ubertooth One [5], a device designed to eavesdrop on Bluetooth communication (both Bluetooth Classic and BLE), and a HackRF One [2], a Software Defined Radio (SDR) designed to monitor RF communication from 1MHz to 6GHz (well within the communication range of personal use devices, usually using either 900MHz or 2.4GHz) as well as transmit signals within that range. Both of these devices are connected to an internet-connected device running Ubuntu, allowing a server to be run for external connections and analysis. Using these devices, our testbed is capable of eavesdropping on Bluetooth communication and, with enough HackRFs, capable of recording and replaying all communication within the range or frequencies used by personal use devices, allowing a user to analyze the communication, eavesdrop on Bluetooth packets, and replay recorded communication or specifically crafted signals.

One advantage of using the HackRF One is that it allows some measure of understanding to be gained of the communication methods used by personal use devices that may choose to use a proprietary communication method. With the communication range pre-defined and available for view through the FCC, it is trivial to detect a signal being sent from the device and to perform basic analysis to discover the data being sent. From there, the data can be analyzed to determine if it is encrypted or unencrypted as well as if replay attacks are possible against the device.

We also use existing Bluetooth Classic and BLE Man-in-the-Middle attacks, specifically GATTack.io [17] and BTLEjuice [1] for BLE and btproxy [3] for Bluetooth Classic attacks. While these attack tools are best used by devices that use JustWorks pairing, they allow the recording of the communication as well as the ability to modify or block communication, as desired. Importantly, while there are an increasingly large number of devices moving away from using JustWorks as their default pairing method, there remain a significant number, including headphones and devices prioritizing user simplicity over security. While these attacks do not and will not work on all wearables or personal use devices, they provide additional understanding of the pairing security of personal use devices using Bluetooth.

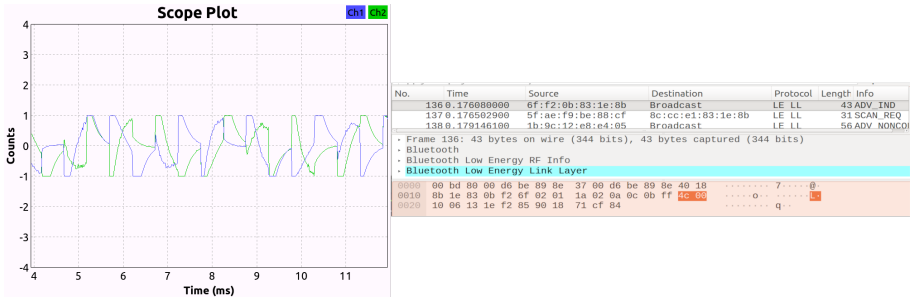


Figure 3: Example captures from HackRF One (Left) and Ubertooth (Right)

With the HackRF One, we have shown a proof-of-concept for a replay attack by recording communication between a user’s car key fob and their car and successfully replayed the communication to cause the car to lock (Figure 3). With the Ubertooth One, we have, with permission, eavesdropped on RF communication, both the communication in and around our campus and to specific devices (Figure 3).

6 Conclusions

The difficulty of both teaching and researching the security of personal use devices, including wearables and home automation technology, cannot be overcome by every instructor or researcher. The cost alone is a major factor in the ability of researchers to examine the area. We propose to solve this with a testbed consisting of real personal use devices that can be attacked with existing attack devices, allowing remote code execution and tests to be run by both researchers and students. We show that, through the creation of a small-scale proof of concept testbed, this method is reasonable moving forward, providing instructors the ability to show existing vulnerabilities and the effectiveness of existing strategies at combating attackers and researcher the ability to both explore potential vulnerabilities of specific devices and test discovered vulnerabilities against a wide range of personal use devices.

Moving forward, we will continue to develop the testbed as well as a robust method of external connection before releasing the full testbed to the public. We will create lesson plans for instructors who wish to incorporate personal use device security into their curriculums, allowing students a better understanding of the security of the devices they are likely to use daily. Finally, we will continue looking into methods of incorporating new devices into the testbed that may not be attackable with traditional eavesdropping, MitM, or replay attacks, including physical attack methods such as connection to active debug ports and power analyses.

References

- [1] BtleJuice Bluetooth Smart (LE) man-in-the-middle framework. <https://github.com/DigitalSecurity/btlejuice>, February 2020.
- [2] HackRF One. <https://greatscottgadgets.com/hackrf/one/>, February 2020.
- [3] Man in the middle analysis tool for Bluetooth. <https://github.com/conorpp/btproxy>, February 2020.
- [4] Number of connected wearable devices worldwide by region from 2015 to 2022.. <https://www.statista.com/statistics/490231/wearable-devices-worldwide-by-region/>, February 2020.
- [5] Ubertooth One. <https://greatscottgadgets.com/ubertoothone/>, February 2020.
- [6] Orlando Arias, Jacob Wurm, Khoa Hoang, and Yier Jin. Privacy and security in internet of things and wearable devices. *IEEE Transactions on Multi-Scale Computing Systems*, 1(2):99–109, 2015.
- [7] Ștefan-Ciprian Arseni, Maria Mițoi, and Alexandru Vulpe. Pass-IoT: A platform for studying security, privacy and trust in IoT. In *2016 International Conference on Communications (COMM)*, pages 261–266. IEEE, 2016.
- [8] Laura Belli, Simone Cirani, Luca Davoli, Andrea Gorrieri, Mirko Mancin, Marco Picone, and Gianluigi Ferrari. Design and deployment of an IoT application-oriented testbed. *Computer*, 48(9):32–40, 2015.
- [9] Yared Berhanu, Habtamu Abie, and Mohamed Hamdi. A testbed for adaptive security for IoT in eHealth. In *Proceedings of the International Workshop on Adaptive Security*, pages 1–8, 2013.
- [10] Christopher Champion, Ilesanmi Olade, Constantinos Papangelis, Haining Liang, and Charles Fleming. The smart² speaker blocker: An open-source privacy filter for connected home speakers. *arXiv preprint arXiv:1901.04879*, 2019.
- [11] Eduardo Cuervo, Peter Gilbert, Bi Wu, and Landon P Cox. CrowdLab: An architecture for volunteer mobile testbeds. In *2011 Third International Conference on Communication Systems and Networks (COMSNETS 2011)*, pages 1–10. IEEE, 2011.
- [12] MP Ponce de Leon, Mats Eriksson, Sasitharan Balasubramaniam, and William Donnelly. Creating a distributed mobile networking testbed environment-through the living labs approach. In *2nd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, 2006. TRIDENTCOM 2006.*, pages 5–pp. IEEE, 2006.
- [13] Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. Security analysis of emerging smart home applications. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 636–654. IEEE, 2016.
- [14] Daniel S Fowler, Madeline Cheah, Siraj Ahmed Shaikh, and Jeremy Bryans. Towards a testbed for automotive cybersecurity. In *2017 IEEE International*

- Conference on Software Testing, Verification and Validation (ICST)*, pages 540–541. IEEE, 2017.
- [15] Md Mahmud Hossain, Shahid Al Noor, Yasser Karim, and Ragib Hasan. IoTbed: A generic architecture for testbed as a service for Internet of things-based systems. In *ICIOT*, pages 42–49, 2017.
 - [16] Fei Hu, Amish Rughoonundon, and Laura Celentano. Towards a realistic testbed for wireless network reliability and security performance studies. *International Journal of Security and Networks*, 3(1):63, 2008.
 - [17] Sławomir Jasek. Gattacking Bluetooth smart devices. In *Black hat USA conference*, 2016.
 - [18] Nataliia Neshenko, Elias Bou-Harb, Jorge Crichigno, Georges Kaddoum, and Nasir Ghani. Demystifying IoT security: an exhaustive survey on IoT vulnerabilities and a first empirical look on internet-scale IoT exploitations. *IEEE Communications Surveys & Tutorials*, 21(3):2702–2733, 2019.
 - [19] S Perez. IDC: Apple led wearable market in 2018, with 46.2 M of the total 172.2 M devices shipped, 2019.
 - [20] Shachar Siboni, Vinay Sachidananda, Yair Meidan, Michael Bohadana, Yael Mathov, Suhas Bhairav, Asaf Shabtai, and Yuval Elovici. Security testbed for Internet-of-Things devices. *IEEE Transactions on Reliability*, 68(1):23–44, 2018.
 - [21] Shachar Siboni, Asaf Shabtai, Nils O Tippenhauer, Jemin Lee, and Yuval Elovici. Advanced security testbed framework for wearable IoT devices. *ACM Transactions on Internet Technology (TOIT)*, 16(4):1–25, 2016.
 - [22] Sandra Siby, Rajib Ranjan Maiti, and Nils Ole Tippenhauer. Iotscanner: Detecting privacy threats in IoT neighborhoods. In *Proceedings of the 3rd ACM International Workshop on IoT Privacy, Trust, and Security*, pages 23–30, 2017.
 - [23] Ali Tekeoglu and Ali Şaman Tosun. A testbed for security and privacy analysis of IoT devices. In *2016 IEEE 13th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pages 343–348. IEEE, 2016.
 - [24] Charles Walter and Rose Gamble. Crossing the adaptation boundaries of distinct testbeds. In *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS* W)*, pages 36–39. IEEE, 2019.
 - [25] Tatsuya Yamazaki. Ubiquitous home: real-life testbed for home context-aware service. In *First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities*, pages 54–59. IEEE, 2005.

Engaging Students with Computing for the Common Good*

Conference Panel

James W. McGuffee¹, Anthony Davis², Mark Goadrich³

*¹Christian Brothers University
Memphis, TN 38104*

jmcguff1@cbu.edu

*²Lyon College
Batesville, AR 72501*

Anthony.Davis@lyon.edu

*³Hendrix College
Conway, AR 72032
goadrich@hendrix.edu*

1 Memphis' Fourth Bluff Project

The Fourth Bluff Project is a partnership between several public entities in the City of Memphis. This project began in 2019 and includes three significant public spaces along the riverfront in downtown Memphis (the historic Cossitt Library, Memphis Park, and Mississippi River Park). The project's goal is to revitalize and transform these public spaces over the next three years with a variety of events, activities, prototypes, enhanced connectivity, signage, and public art, with the hope of redefining the public's expectations of public parks and libraries, and foster socio-economic connections across boundaries in a culturally and environmentally sensitive way.

All computer science majors at Christian Brothers University (CBU) must complete a year long software development project. In cooperation with CBU's Autozone Center for Community Engagement, one of our current computer science seniors is working on a software project in support of the Fourth Bluff Project. Additionally, there have been preliminary discussions between CBU and the Memphis Public Library to expand on the types of projects that might be accomplished in the next academic year.

*Copyright is held by the author/owner.

2 Vesuvian Institute Near Stabiae

Founded by the Restoring Ancient Stabiae Foundation in 2007, the Vesuvian Institute is a multi-functional academic and residential hospitality complex situated overlooking the Bay of Naples and centrally located to the most famous ancient Roman sites in the world. Our focus is to support and broaden the study of the cultural, historic and artistic patrimony of our southern Italian region and partner with researchers and institutions from all over the world. The Vesuvian Institute is an ideal location for study and tourism in the Bay of Naples.

In the summer of 2020, a group of faculty and students from Lyon College will participate in an immersive Vesuvian Institute experience. These students will include computer science majors who will be working with data to support the scientific discovery happening at Stabiae.

3 Non-Profit Web Systems Partnerships

Many non-profit organizations have workflow processes currently tracked with paper and pencil, due to small budgets, limited time, or lack of technical expertise. In 2016, 2018, and 2019, I taught an upper-level course for computer science majors and minors which covered Databases and Web Systems, and included a large term project with a local community partner, such as the United Way of Central Arkansas, Salvation Army, Child Care Aware, and the Conway Human Development Center. Through this course, students learned the course material and created database systems in the context of assisting a community in need.

My students worked in teams of six, meeting with their local organization four times over the semester, first gaining an understanding of the needed system, and then returning multiple times with work-in-progress updates, culminating in a final presentation for the partner of their prototype application. This partnership allowed the students to grasp an understanding of complex issues in service to others and to help the organizations use databases and programming to make the organization more efficient. Students applied their classroom knowledge to create a practical system for a community partner, and in the process, learned about how the partner effects change in the world.

Abstract Syntax BNF Is Not Ambiguous/Inadequate*

Nifty Assignment

Cong-Cong Xing¹, Jun Huang²

¹ Department of Mathematics and Computer Science
Nicholls State University
Thibodaux, LA 70310

`cong-cong.xing@nicholls.edu`

²School of Computer Science
Chongqing Univ. of Posts and Telecommunications
Chongqing, China 400065

`xiaoniuadmin@gmail.com`

Motivation Abstract syntax BNF is commonly used in the teaching of theory classes in computer science. One of the confusing points associated with abstract syntax BNF is that many students think that an abstract syntax BNF is ambiguous or inadequate to deliver unequivocal language constructs, and needs to be “fixed” into a concrete syntax BNF in order to be useful. While this viewpoint is prevalent, it is not the only way to perceive this problem. The purpose of this nifty assignment is to offer an alternative viewpoint on this issue.

Overview We typically say that a grammar G is ambiguous or inadequate if a string in $L(G)$ has more than one parse tree. For any character string, which is supposedly derived from a given BNF grammar, there are two types of information embedded in this string: (1) the textual symbols which constitute the string itself and (2) the *order* in which the string is constructed by those textual symbols. Unfortunately, due to the way strings are represented normally (all characters are written on a single line with fonts of the same style and same size), the order is *invisible* or lost. However, this loss of information is not caused by the abstract syntax itself, rather, it is caused by the way strings are *represented*. More precisely, the abstract syntax BNF itself specifies (in an abstract level) the order to build strings by its production rules,

*Copyright is held by the author/owner.

and we concretely follow this order step by step to build strings out of the abstract grammar. But the footprint of following this order is lost in the final result due to the way strings are normally represented. In other words, had we used a different way to represent the derived strings, the order would have been revealed. One of such ways is the tree representation in which a string is written as a 2-dimensional figure instead of a linear sequence of symbols. Another way is to write a string as a linear sequence of symbols using fonts of different sizes, which is what we introduce in this nifty assignment. We suggest that students use the following simple abstract syntax BNF $M := C|MOM$, $C := 0|1|2|\dots|9$, $O := +|-$ to derive some strings in both traditional way (e.g., 3-4+5) and “different font-size” way (e.g., 3-4+5 and 3-4+5) to realize the intrinsic order that comes with the abstract syntax BNF, and to see how this order gets lost (thus causing the ambiguity issue) in the normal representation of strings, but is preserved (thus nullifying ambiguity issue) if strings are represented by fonts of various sizes. More examples can be given in the handouts to students. We stress the following two points.

- The ambiguity or inadequateness issue of abstract syntax BNF is rooted in the way strings are (normally) represented. This issue does not stem from the abstract syntax BNF itself. Abstract syntax BNF has the order information inherently embedded in its production rules and in the sequence in which these rules are applied. This order could be shown if we choose to represent the resulted strings as a tree or in a font-sized style.
- The font-sized mechanism is equivalent to the tree mechanism.

Classroom observations and possible gains: This assignment was given as homework for a senior-level theory course in computer science. The feedback was mixed – some students appreciated that the font-sized way helped them see the underlying order that is otherwise invisible or lost, but some were still confused. Possible gains from this assignment include

- A clearer and deeper understanding about the nature of the abstract syntax BNF.
- A true comprehension of the “ambiguity or inadequateness issue” associated with the abstract syntax BNF.

We hope that this assignment can be found useful by colleagues.

Increasing Cyber Security Awareness by Creating a Case Study and Video Project*

Nifty Assignment

Luay A. Wahsheh

Department of Computer and Information Science

Arkansas Tech University

Russellville, Arkansas 72801

lwahsheh@atu.edu

Abstract

Writing case studies in undergraduate and graduate courses in cyber security is a great way to learn class material. In this assignment, a student creates a case study and video about a cyber security incident. The idea is to engage the audience and make them curious about the content of the case study and video, thus increasing cyber security awareness. In developing a case study, the student picks an interesting scenario or story that provides an example of a cyber security incident. It can be a real cyber security incident or one that the student envisions. Once the student picks a case, he or she locates reliable sources to provide more information about it, writes a case study, and presents the case study to his or her classmates and instructor in the forms of a short paper (1 page) and video (1 minute). The video is an illustration of the student's case study and should use the same title and scenario. The student will combine text, images, and sound to tell the audience about his or her case study. The student should be present in the video illustrating his or her case study.

1 Deliverables

There are eight milestones throughout the course to help students build their case study and video.

*Copyright is held by the author/owner.

- Milestone 1 – Video: Download, Install, and Learn Video Software.

Students pick, install, and learn a video editing software. There is no deliverable to their instructor for this step. It is a reminder to have the software installed in order to continue with the rest of the project.

- Milestone 2 – Case Study / Video Summary or Abstract.

Students provide a short description of their topic. They explain how the topic is connected to their course on cyber security.

- Milestone 3 – Case Study / Video Title and Preliminary List of References.

Student provide the title of their case study and a preliminary list of recent references.

- Milestone 4 – Case Study Outline and Video Storyboard.

Students develop an outline of their case which shows at a high level what they are going to cover. They should introduce the case in the beginning by describing the situation, problem, or story. They provide more details and may have some aspects of research to prove their points. Then, they discuss lessons learned from the case. Students create a storyboard for their video. This is a plan for their video and is similar to creating an outline. They put together a description of their video in a paragraph or it could be a list of images/scenes if that works better than a paragraph.

- Milestone 5 – Collect Picture, Movies, and Multimedia Content for Video.

Students collect media, pictures, and other content for their video. They combine text, images, and sound to tell the audience about their case. They should be present in the video (live record themselves) illustrating their case study. There is no deliverable to their instructor for this step. It is a reminder to the students to get their material organized.

- Milestone 6 – Case Study Paper First Draft and Video First Version.

Students submit their case study first draft and video first version. When they have finished their first draft and video first version, students are encouraged to assume that they would spend only a couple of hours of revision to produce their final paper and video.

- Milestone 7 – Critique and Collect Feedback.

Students critique and grade one of their classmate’s case study paper first draft and video first version. Students collect feedback on their case study and video from their classmates’ critique.

- Milestone 8 – Case Study Final Paper and Video Final Version.

Students submit their case study final paper and video final version.

2 Tools

In order to provide a consistent format for all the case studies, it is required that students use a conference template to deliver their case study work, which contains information about formatting the case studies. An IEEE conference template is provided to the students by the instructor.

Students could use any video editing software to create their video. The video must be in a common format to be played on a standard Windows personal computer (e.g., Windows Media, Flash, or Quicktime). Students could use their smartphone to create a high quality video using a free product called Videolicious (<https://videolicious.com/>). Other tools include Microsoft Windows Live Photo Gallery and Movie Maker and Microsoft PowerPoint (if they feel comfortable with its multimedia features). Macintosh users can use iMovie for Mac (<https://www.apple.com/imovie/>) or a similar multimedia package designed especially for the Mac platform.

Students could use OpenShot (<https://www.openshot.org/>), a very simple, easy-to-use, and free program. Adobe Creative Cloud applications, which include Premiere and Premiere Rush could be used by students. Premiere is the full-blown video editing software and Rush is the more basic program that includes mobile app versions. Another option is Lightworks, which is available to download for free (the Pro version is not free of charge). Additional tools include Wondershare Filmora9, VideoScribe, Clideo, Davinci Resolve, and PhotoStage.

3 Conclusion

In this assignment, students create a case study and video about a cyber security incident. This assignment is well received by students in undergraduate and graduate courses in cyber security. Students like that this assignment is engaging, hands-on, and creative. This assignment could be easily modified by instructors to be used in undergraduate and graduate courses, no matter the field of study.

The State of Machine Learning*

Conference Tutorial

Dan Brandon
MIS, School of Business
Christian Brothers University
Memphis, TN 38104
dbrandon@cbu.edu

The field of Artificial Intelligence (AI) is certainly the buzz today, and we are deluged by TV and print ads from vendors offering AI hardware, software, and consulting services. This is rightly so as AI has achieved a high degree of success in recent years for addressing many problems in a variety of disciplines including: business, robotics, medicine, sports, and general use as self-driving vehicles.

The more appropriate term here is Machine Learning (ML) which is the scientific study of algorithms and statistical models that computer systems use to progressively improve their performance on a specific task. A machine learning system is trained rather than explicitly programmed. Machine learning algorithms build a mathematical model from sample data, known as “training data”, in order to make predictions or decisions without being programmed. In its application across business problems, machine learning is often referred to as “predictive analytics”.

A popular misconception is that machine learning only employs neural network algorithms. However the field of machine learning is very broad and includes many methods and algorithms. In addition, many problems can be addressed by several different algorithms.

ML can be divided into supervised vs unsupervised approaches. In supervised learning, the data one feeds to the algorithm includes a “label” (the answer or desired output variable). Supervised is further divided into regression where the output is a real value, and classification where the output is a category. Examples of supervised methods include:

Linear regression (MLR)

Logistic regression and other specialized regressions Naive Bayes

*Copyright is held by the author/owner.

Multivariate Analysis of Variance (MANOVA) Linear discriminant analysis
Nearest neighbors
Support vector machines (SVM)
Trees and forests
Neural networks when used in a supervised manner

Unsupervised learning is the training of a machine using information that is neither classified nor labeled and allowing the algorithm to act without guidance to group unsorted information according to similarities, patterns, and differences. In business and medical applications, unsupervised learning is often called “data mining”. Examples of unsupervised methods include:

K-mean clustering and K-nearest neighbor clustering Hierarchical cluster analysis
Visualization and dimensionality reduction
Principal component analysis (PCA)
Kernel methods and Support Vector Machines (SVM)
Canonical Correlation (CC)
Conjoint analysis
Factor Analysis (FA)
Multidimensional Scaling
Structure Equation Modeling (SEM)
Affinity (Market Basket) Analysis
Neural networks when used in an unsupervised manner

A key issue in ML is which algorithms work best for which problems. Key questions are: does the problem need a supervised or unsupervised approach, is this a perceptual or numerical problem, does the output require a regression or classification, and how much training data is available. This tutorial will provide some ML history, describe ML approaches and algorithms, and address those key questions involving the selection of ML algorithms. The tutorial will also discuss the resources needed and available for teaching ML in colleges.

Presenter Background

Dr. Dan Brandon is a Professor of Management Information Systems at Christian Brothers University in Memphis TN where he teaches courses in MIS, programming, database, data analytics, statistics, and project management. He has authored two books and numerous journal articles and conference proceedings including some for CCSC. He has designed and developed modern IT systems for a number of organizations and was formerly the Director of Information Systems at the NASA Stennis Space Center.

Cyber Security Hands-On Learning Using Steganography*

Conference Tutorial

Luay A. Wahsheh

Department of Computer and Information Science

Arkansas Tech University

Russellville, Arkansas 72801

lwahsheh@atu.edu

Abstract

One method that provides more security in computer systems is the use of hidden messages. The hidden messages can be plaintext, ciphertext, or image. This tutorial will give an overview of steganography, which is a technique to hide messages within data. We will install Quick Stego and WinHIP. Then, we will have hands-on exercises that use these tools to hide secret messages.

Description

Steganography is hiding messages within data. This technique makes secret messages (which could be malware) appear invisible to entities. Although steganography algorithms use different formats including image, audio, and video, we focus on hiding messages within images. The image would be the carrier that holds the hidden message and the original content of the image. The hidden messages are embedded in a way that does not significantly change the properties of the original image. Steganography software tools (e.g., WinHip, EzStego, and OpenPuff) allow embedding hidden messages in an image and then extract that information.

Both steganography and cryptography are concerned with preventing unauthorized access to information. One advantage of steganography over cryptography is that a secret message that is generated by using steganography does

*Copyright is held by the author/owner.

not attract attention to itself because no encryption is used. When steganography is combined with cryptography, the security of data increases; in this situation, steganography is used to hide a ciphertext, and if the use of steganography was discovered, then cryptography is used to encrypt the plaintext.

The original image is the one in which the secret message is embedded. The payload is the secret message that will be embedded in the original image. The stego image is the final image that resulted from embedding the payload in the original image. Figure 1 shows an example of an original image, payload, and stego image. We used WinHip steganography software tool to produce the stego image. The human naked eye cannot detect the difference between the original image and the stego image.

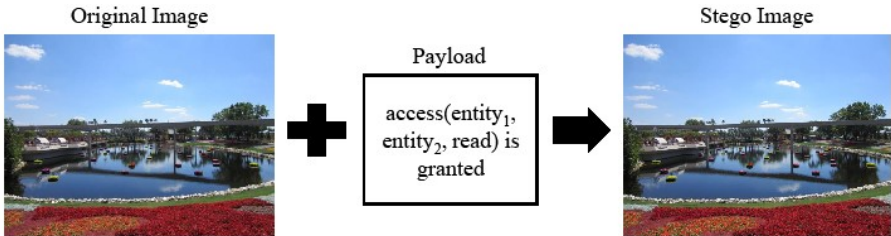


Figure 1: An Original Image, Payload, and Stego Image

Introduction to Jetstream - A Research and Education Cloud *

Conference Tutorial

Sanjana Sudarshan and Jeremy Fischer
Research Technologies
Indiana University
Bloomington, IN 47401
{ssudarsh, jeremy}@iu.edu

Abstract

Jetstream is the first production cloud funded by the National Science Foundation (NSF) for conducting general-purpose science and engineering research as well as an easy-to-use platform for education activities. Unlike many high-performance computing systems, Jetstream uses the interactive Atmosphere graphical user interface developed as part of the iPlant (now CyVerse) project and focuses on interactive use on uniprocessor or multiprocessor. This interface provides for a lower barrier of entry for use by educators, students, practicing scientists, and engineers. A key part of Jetstream's mission is to extend the reach of the NSF's eXtreme Digital (XD) program to a community of users who have not previously utilized NSF XD program resources, including those communities and institutions that traditionally lack significant cyberinfrastructure resources. One manner in which Jetstream eases this access is via virtual desktops facilitating use in education and research at small colleges and universities, including Historically Black Colleges and Universities (HBCUs), Minority Serving Institutions (MSIs), Tribal colleges, and higher education institutions in states designated by the NSF as eligible for funding via the Established Program to Stimulate Competitive Research (EPSCoR).

While cloud resources won't replace traditional HPC environments for large research projects, there are many smaller research and education projects that would benefit from the highly customizable, highly configurable, programmable

*Copyright is held by the author/owner.

cyberinfrastructure afforded by cloud computing environments such as Jetstream. Jetstream is a Infrastructure-as-a-Service platform comprised of two geographically isolated clusters, each supporting hundreds of virtual machines and data volumes. The two cloud systems are integrated via a user-friendly web application that provides a user interface for common cloud computing operations, authentication to XSEDE via Globus, and an expressive set of web service APIs.

Jetstream enables on-demand access to interactive, user-configurable computing and analysis capability. It also seeks to democratize access to cloud capabilities and promote sharable, reproducible research. This event will describe Jetstream in greater detail, as well as how its unique combination of hardware, software, and user engagement support the "long tail of science." This tutorial will describe Jetstream in greater detail, as well as how its unique combination of hardware, software, and user engagement support the "long tail of science." Attendees will get a greater understanding of how Jetstream may enhance their education or research efforts via a hands-on approach to using Jetstream via the Atmosphere interface.

Tutorial Description

This tutorial requires two to three hours.

- Prerequisites: Basic Linux command line knowledge a plus (but not required)
- Required: Laptop, modern web browser (Chrome, Firefox, Safari)
- Targeting: Educators, Researchers, Campus Champions/ACI-Ref Facilitators, Campus research computing support staff

This tutorial will first give an overview of Jetstream and various aspects of the system. Then we will take attendees through the basics of using Jetstream via the Atmosphere web interface. This will include a guided walk-through of the interface itself, the features provided, the image catalog, launching and using virtual machines on Jetstream, using volume-based storage, and best practices.

We are targeting users of every experience level. Atmosphere is well-suited to both HPC novices and advanced users. This tutorial is generally aimed at those unfamiliar with cloud computing and generally doing computation on laptops or departmental server resources. While we will not cover advanced topics in this particular tutorial, we will touch on the available advanced capabilities during the initial overview.

Tutorial Program

This is a sample tutorial program. Time required for this tutorial is approximately 3 hours.

- What is Jetstream?
- Q & A and what brief hands-on overview
- Getting started with Jetstream, including VM launching
- Break
- Accessing your VM, creating and using volumes
- Customizing and saving images, DOIs
- Cleaning up
- Final Q & A