in Proc. of the First Workshop on Cyber-Physical Education (CPS-Ed),
in conjunction with CPS week, April 8, 2013, Philadelphia, Pennsylvania.
http://cps-vo.org/group/edu/workshop

Virtualizing Cyber-Physical Systems: Bringing CPS to Online Education

Jeff C. Jensen
National Instruments
Berkeley, CA 94704
Email: jjensen@ni.com

Edward A. Lee
University of California, Berkeley
Berkeley, CA 94720
E-mail: eal@eecs.berkeley.edu

Sanjit A. Seshia
University of California, Berkeley
Berkeley, CA 94720
E-mail: sseshia@eecs.berkeley.edu

***Abstract*—The advent of the massive open online course promises to bring world-class education to anyone with internet access. Instructors use *blended models* of education to deliver course content via video, text, interactive assignments, exams, wikis, and discussion forums. Courses with largely theoretical content are readily adapted to blended models for online audiences, but significant challenges arise when incorporating project-based learning and interactive experiments. Cyber-physical systems courses commonly include experiments that explore the interplay between computation and physics and are especially subject to the challenges of bringing experimentation and project-based learning to online audiences. We describe technical aspects of embedded and cyber-physical systems laboratory exercises used at the University of California, Berkeley, and investigate avenues for adapting this content to a massive open online course.**

I. Introduction

A massive open online course (MOOC) [1] is an online course designed to scale to a large number of participants who do not need to be registered at an academic institution. Models for developing MOOCs vary dramatically and the landscape is still largely uncharted. Organizations leading the development of MOOCs include edX (a nonprofit founded by MIT and Harvard) [2], Coursera (a private organization with roots in Stanford University) [3], and Udacity (a private organization also with roots in Stanford University) [4]. An example of the scale of a MOOC comes from edX, where its inaugural MITx course 6.002x in introductory circuits [5], [6] drew more than 150,000 registrants [7], 28% of which completed the course [6]. The University of California, Berkeley, recently became a partner in edX [8].

The course EECS 149, "Introduction to Embedded Systems" [9] at the University of California, Berkeley, is a course taken by graduate students and undergraduate juniors and seniors. The course follows the textbook Lee & Seshia, *Introduction to Embedded Systems: A Cyber-Physical Systems Approach* [10], and lecture recordings are published online [11]. The textbook is available as a free PDF download and is designed specifically for reading on desktops and tablets, featuring extensive use of hyperlinks [12]. Laboratory exercises give students experience with three distinct levels of embedded software design, namely bare-metal programming (software that executes in the absence of an operating system), programming within a real-time operating system, and model-based design for cyber-physical systems. In each case, students are taught to think critically about technology, to probe deeply the mechanisms and abstractions that are provided, and to understand the consequences of chosen abstractions on overall system design [13], [14]. Laboratory exercises follow Jensen, Lee, & Seshia, *An Introductory Lab in Embedded and Cyber-Physical Systems* [15], a textbook companion that is available as a free PDF download along with all accompanying software and documentation.

Over the last six years we have developed and refined the lectures, textbook, and laboratory exercises. Our focus now turns to adapting the course to a MOOC. Initial results have been obtained for automatically generating problems and solutions for textbook exercises and auto-grading [16]; however, more must be done to bring online the laboratory exercises that are central to the course experience. Here, we addresses this challenge, beginning with a technical overview of key laboratory exercises and then evaluating potential adaptations for an online audience.

II. Architecture of the On-Campus Labs

The Cal Climber [17], [18] is a cyber-physical system based on a commercially available robotics platform derived from the the iRobot Roomba autonomous vacuum cleaner. The off-the-shelf platform is capable of driving, sensing bumps and cliffs, executing simple scripts, and communicating with an external controller. The Cal Climber is comprised of the iRobot Create, a National Instruments Single-Board RIO (sbRIO) embedded microcontroller, and an Analog Devices ADXL-335 three-axis analog accelerometer. The Cal Climber demonstrates the composition of cyber-physical systems, where a robotics platform is modeled as a subsystem and treated as a collection of sensors and actuators located beyond a network boundary.

The problem statement is as follows [15]:

> *Design a Statechart to drive the Cal Climber. On level ground, your robot should drive straight. When an obstacle is encountered, such as a cliff or an object, your robot should navigate around the object and continue in its original orientation. On an incline, your robot should navigate uphill, while still avoiding obstacles. Use the accelerometer to detect an incline and as input to a control algorithm that maintains uphill orientation.*

Students employ model-based design [17], [19] to develop software for controlling the Cal Climber, making use of the LabVIEW Robotics Environment Simulator by National Instruments. The simulator is based on the Open Dynamics Engine [20] rigid body dynamics software that can simulate robots in a virtual environment (Fig. 1). Students design their controller in both C and LabVIEW following the Statecharts model of computation, and execute their controllers both in simulation and on the real device. Student controllers execute within a hardware abstraction layer responsible for simulating sensors and actuators when executing within the simulator and linking to sensor and actuator drivers when deploying to the embedded target (Fig. 2). Control software can be migrated between simulation and deployment with little or no modification. The result is a complete end-to-end model-based design flow (Fig. 3).

Source files distributed with the Cal Climber laboratory are designed so that students need modify only a single C source file or a single LabVIEW Statechart file to complete the exercises. The template C source file is structured as a Statechart that receives as arguments the most recent values of the accelerometer and robot sensors and returns desired wheel speeds. The LabVIEW Statechart receives the same inputs and produces the same outputs as the C source file. In both C and LabVIEW, timing is governed by the rate at which the robot sensors are updated, and inputs and outputs use the same units and binary representation. Timing is simulated by sampling the simulated environment at the same rate as the real robot samples its environment, and quantization error is simulated by converting physical quantities in simulation to the same binary representation used by the robot. The hardware abstraction layer and robotics simulator interface are written in structured dataflow [21], which can execute C code that has been compiled into a dynamic-linked library (DLL), or natively execute a LabVIEW Statechart.

LabVIEW Robotics Environment Simulator provides a graphical interface for customizing environments, including adding, removing, and moving obstacles, changing the initial position of the robot, adjusting sensor locations, and adding multiple robots to the environment. Robots modeled in CAD software such as SolidWorks and Google Sketch can be imported and customized with sensors and actuators. The configuration of the environment, including dimensions and placement of objects and robots, is stored in an XML file.

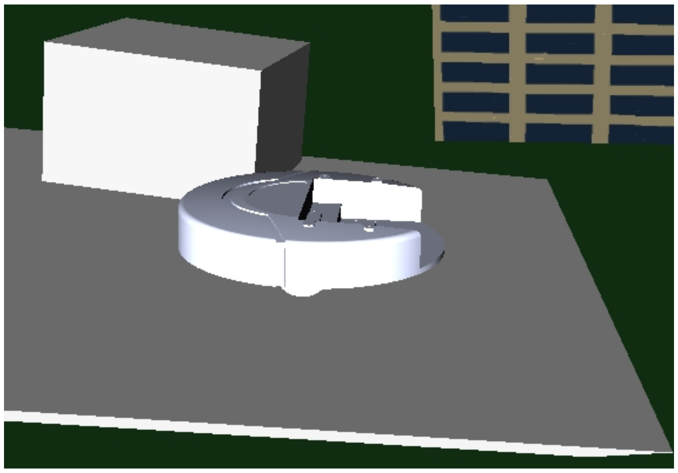
Fig. 1. Cal Climber in the LabVIEW Robotics Environment Simulator.

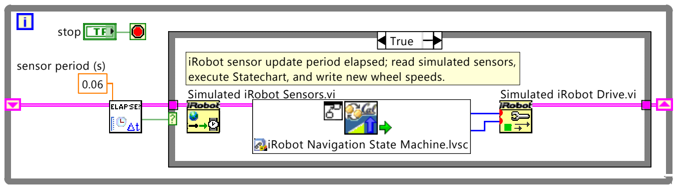
stop
TF
sensor period (s)
0.06
ELAPSED
True
iRobot sensor update period elapsed; read simulated sensors, execute Statechart, and write new wheel speeds.
Simulated iRobot Sensors.vi
iRobot Navigation State Machine.lvsc
Simulated iRobot Drive.vi

Fig. 2. Cal Climber simulator interface in LabVIEW. “Simulated iRobot Sensors” is the hardware abstraction layer for robot sensors, “iRobot Navigation State Machine” is the controller in LabVIEW Statecharts (replaceable by a library call to a DLL produced by C source), and “Simulated iRobot Drive” is the hardware abstraction layer for the robot actuators.

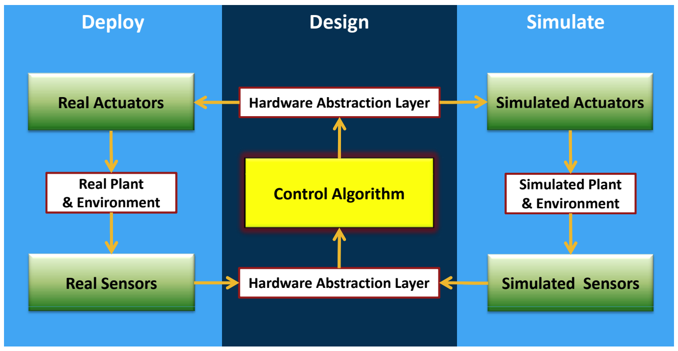

Fig. 3. Model-based design flow used in the Cal Climber exercises.

III. Cost of Virtualization

In an ideal world, we would provide an infrastructure where students can log in remotely to a computer which has been preconfigured with all development tools and laboratory exercises. In fact, such an infrastructure is already becoming a reality. For instance, the iLab project [22] at MIT has for several years sought to make real laboratories accesible through the Internet. In the setting of our course, each virtual computer would connect wirelessly to a real robot in a real environment, with feedback provided by robot sensors, a robot-mounted camera, and one or more cameras with visibility of the environment.

There are few technical challenges to such an implementation of the laboratory: only the cameras and remote access are added to the existing implementation. Scaling to 100,000 students, however, poses significant logistical challenges. Assuming each student has a dedicated robot and environment, each environment has modest dimensions of 5 feet wide by 10 feet long, and each robot costs $500 US dollars, the combined space needed for the environments would exceed that of the Boeing Everett Factory (the largest footprint of any building in the world) and the combined cost would exceed $50 million for the robots alone! Time-sharing is implausible as even the most optimal utilization of resources would result in a still sizable fraction of these costs. Then consider incapacitated robots, dead batteries, and other practical concerns faced by real-world laboratories. MITx 6.002x uses circuit simulation software instead of physical hardware due to similar concerns [6]. For a MOOC with cyber-physical laboratories, only simulation appears viable.

Having solved – or sidestepped – the hardware question, now consider software. The Cal Climber exercises require a gcc compiler for development of the C controller and LabVIEW for the simulation environment and development of the dataflow controller. If students install laboratory software on their personal computers, they absorb cost of licensing commercial software and need computers capable of running modern design and simulation software, which seems counter to the open model of a MOOC. Alternatively, a virtual server maintained by the institution would provide students remote access to virtual instances of a computer that has been preconfigured with all necessary software. Students need not purchase, install, or configure software, and computers need only the capability to access the virtual server over a decent internet connection. Using a virtual server alleviates support burdens and student costs while incurring costs on the institution including any license agreements with commercial software and operating system vendors, setup and maintenance costs of institution-operated servers, and usage fees for a third-party service provider ("cloud" services).

Consider a student remotely connecting to a virtual computer running a Microsoft Windows operating system. The virtual server is hosted on Amazon Elastic Compute Cloud (EC2) services and users connect using Microsoft Remote Desktop Connection. A pilot of LabVIEW running on a virtual Windows server in EC2 is in progress, and early results are promising [23]. We extended this pilot with the simulated labs and verified performance and behavior are sufficient when executing in the cloud. Further evidence of the potential of EC2 is BerkeleyX CS161.1x, which enables students to replicate a virtual computer that has been preconfigured with all development software. Pricing for the LabVIEW EC2 service, at the time of this publication, is \$0.18 per hour for software licenses, \$0.23 per hour for EC2 usage, and \$0.12 for each gigabyte of data transferred [24]. Students typically take between 9 and 12 hours to complete the Cal Climber exercises. Microsoft Remote Desktop Connection will likely use more than 100 kilobytes of data per second during the use of graphical design environments and 3D simulations [25]. The net cost of using Amazon EC2 cloud services for hosting the Cal Climber exercises for 100,000 students is approximately \$492,000 in usage fees and \$51,840 in data transfer fees. Clearly this is is an unreasonable cost for the institution, but the cost per student is convincing:

$$\left[\frac{\$0.41}{\text{hr}} + \left(\frac{\$0.12}{\text{gB}} \times \frac{100\text{kB}}{\text{sec}}\right)\right] \times \frac{12\text{hr}}{\text{student}} = \frac{\$5.44}{\text{student}}. \quad (1)$$

Given the relatively low cost per student and ease of support, we plan to use virtualization to bring the course online.

IV. Automatic Grading

Manual grading of laboratory solutions for massive courses is difficult. Some courses, mostly non-technical, use crowd-sourcing. Technical courses such as the software engineering courses BerkeleyX CS169.1x and CS1692.x employ automatic grading. The model-based design framework of the laboratory exercises aids the creation of automatic grading software, particularly because the controller is modular and independent of the simulator. The concept is to develop grading software that can easily enumerate simulation environments and student solutions.

A. Grading via Simulation

One implementation of the grading software would be a modified simulator that executes the environment simulator with the submitted controller, tests for success and failure conditions, reports, and then permutes the environment and initial conditions. A simple approach would be to manually build a number of set of test cases, where each test case includes an environment, initial conditions, and a set of desired final positions for the robot. Automatic generation of test cases may be possible through the use of simulation-based falsification tools such as S-TaLiRo [26] or Breach [27], [28]. Formal verification tools such as model checkers may be used to decide whether a solution is correct if relatively a simple model of computation (e.g., finite state machines) is used for the student controller. Design requirements can be formally stated in mathematical notation such as temporal logic, allowing the use of techniques for efficient run-time monitoring of such requirements [27]. Temporal logic is a topic covered in our course, and we envision sharing requirements in terms of temporal logic to encourage students to learn through the grading process.

Test cases should be organized in such a way that they first test simple, independent requirements, then advance to more sophisticated requirements, complete test courses, and "torture tests". For example, first test cases would assert "the robot maintains orientation" and "the robot navigates to the top of an incline" in an environment without obstacles. Test cases advance to asserting more complex requirements like "the robot maintains orientation while avoiding obstacles", then sophisticated assertions like "the robot maintains orientation and avoids obstacles on level ground and navigates uphill when on an incline". Torture-tests can introduce environments where the robot may encounter fixed obstacles while on level ground or on an incline, or moving obstacles such as another robot. We imagine test cases where two robots navigate within the same environment, with one robot controlled by student code and the other controlled by a solution submitted by a fellow student or instructor – a test case that challenges student controllers to be robust against a potentially competitive environment.

B. Feedback Generation

Automatic grading serves the purpose of deciding whether a submitted answer is correct, but perhaps more importantly, generating useful feedback to the student when an answer is incorrect or inelegant. One kind of feedback is to identify the design steps in which the student went wrong. This is a form of debugging, in the following sense: given an error trace showing the failure of a test, automatic grading software must identify the steps that lead to the failure of a requirement. This problem has been studied in the computer-aided verification and software engineering literature under the terms "error localization" or "bug localization."

Several error localization methods operate on source code, but these can be limited by the expressiveness of the models of computation used and the size of the programs they analyze. More promising are error localization and explanation techniques that operate on execution traces [29], [30], and those that make use of "fault" models that codify common mistakes [31]. An open technical challenge is to lift these methods to operate on the rich hybrid dynamical models needed in the design and analysis of cyber-physical systems.

C. Local versus Global Implementation

One possible implementation of automatic grading is *local grading* where students students run grading software that tests their solutions, displays results, and optionally submits results to an online database. When students are satisfied with their design, they run tests and submit the results to a remote grading server. If students are able to pass some test cases, but not all, they may still submit their results for partial credit. A secure results database must be maintained by the institution, and care must be taken to prevent students from submitting fraudulent results. Local grading leverages the existing virtual computing framework and gives immediate feedback, but the grading process is vulnerable to hacking. Local grading software is straightforward to implement and does not require any additional virtual computing frameworks beyond a simple secure grading results database.

One benefit of local grading is the instructional value of testing. Students, in a sense, know the answer before they begin: design a controller that satisfies the problem statement. It is the design methodology and the steps leading to the answer that they must learn. With local testing, as test cases are enumerated, students can watch the video of their controllers executing within each simulated test case. This gives them immediate and valuable insight into where design changes should be made. Tests can be run one at a time to focus on a specific case, or in sequences that ensure modifications directed at more advanced cases do not result in the failure of more primitive tests – in essence, a unit test framework.

An alternate implementation of automatic grading is *global grading* where students submit their solutions to a grading server that tests solutions, records results on a local database, and sends the student a report via e-mail. Global grading is more secure since students do not have visibility into the grading software, but grading may not be immediate and a separate grading computer (or cluster of computers) must be virtualized and maintained by the institution. Students do not receive immediate feedback as tests are run, but students can be given the test environments to execute on their own. Grading machines may need to be overprovisioned or capable of spawning additional instances should demand be high – such as the hours before exercises are due. Since testing is performed on a secure server, fraudulent results are difficult to submit, but the burden on the institution is high. Global grading would also require custom software to manage queues, report results, and record grades.

D. Detecting Cheating

In both local and global grading, steps must be taken to prevent students from sharing solutions. Many techniques exist for automatic detection of textual code plagiarism [32], however we have been unable to find methods for similar detection of graphical code plagiarism. A naive method for detecting plagiarism is to compare simulation traces, however this may not be practical (or even possible). The ODE solver may not always produce equal traces for sequential executions of the same controller, especially if any form of noise is modelled. If the active states of the solution Statechart are output along with control signals, then an execution trace can be produced by the grader and checked against previously submitted execution traces. Automatic detection of textual code plagiarism, as well as execution trace correlation, place additional burdens on the institution, but we are not far enough along in development to predict what will be the impact and cost. We leave this as future work.

E. Execution on Hardware

As previously discussed, making real hardware available to several thousand students is impractical. However, the code that students write for use in a simulation-based virtual environment must also run on real hardware. This is an important requirement that ensures that the students learning experience is as "close" as possible to the one they would get in a real hardware laboratory. Executing the best student solutions on real hardware is also a nice validation for the students to see their work in action in the real world. Students who have the wherewithal to purchase and use their own hardware may also be able to deploy their solutions on that hardware and submit, as extra evidence, video of their code running on the hardware.

V. Conclusion

We give evidence that some laboratory exercises in cyber-physical systems can be adapted for use by a MOOC. Hardware replication and resource sharing appear infeasible both in cost and maintainability; however, simulation software such as the LabVIEW Robotics Environment Simulator and virtual computing services such as Amazon EC2 are attractive and cost-effective alternatives that are readily maintained. While the experience of a physical lab environment can never be fully replaced, the combination of simulation software and virtual computing enables online students to experiment, learn, and achieve the laboratory objectives of our course.

Acknowledgements

This work was supported in part by NSF CAREER award #0644436, an Alfred P. Sloan Foundation Fellowship, and the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley, which receives support from the National Science Foundation (NSF awards #0720882 (CSR-EHS: PRET), #0931843 (CPS: Large: ActionWebs), and #1035672 (CPS: Medium: Ptides)), the Naval Research Laboratory (NRL #N0013-12-1-G015), and the following companies: Bosch, National Instruments, and Toyota.

References

[1] *The Year of the MOOC*, New York Times, November 2012. [Online]. Available: http://www.nytimes.com/2012/11/04/education/edlife/massive-open-online-courses-are-multiplying-at-a-rapid-pace.html

[2] edX. [Online]. Available: http://edx.org

[3] Coursera. [Online]. Available: http://coursera.org

[4] Udacity. [Online]. Available: http://udacity.com

[5] A. Agarwal, G. Sussman, and P. Mitros. MITx 6.002x: Circuits and Electronics. Massachusetts Institute of Technology. [Online]. Available: https://6002x.mitx.mit.edu/

[6] P. Mitros, K. Afridi, G. Sussman, C. Terman, J. White, L. Fischer, and A. Agarwal, "Teaching Electronic Circuits Online: Lessons from MITx's 6.002x on edX," in *International Symposium on Circuits and Systems (ISCAS)*. Beijing, China: IEEE, May 2013 (to appear).

[7] D. Chandler, "MIT and Harvard launch a 'revolution in education'," MIT News Office, May 2012. [Online]. Available: http://web.mit.edu/newsoffice/2012/edx-launched-0502.html

[8] "UC Berkeley Joins edX," edX, July 2012. [Online]. Available: https://www.edx.org/press/uc-berkeley-joins-edx

[9] E. A. Lee and S. A. Seshia. EECS 149 course website. University of California, Berkeley. [Online]. Available: http://chess.eecs.berkeley.edu/eecs149

[10] ——, *Introduction to Embedded Systems - A Cyber-Physical Systems Approach*. Berkeley, CA: LeeSeshia.org, 2011. [Online]. Available: http://LeeSeshia.org

[11] ——, "EECS 149 lecture webcast," University of California, Berkeley, 2012. [Online]. Available: http://www.youtube.com/playlist?list=PL62BE418C34C3B2BD

[12] S. A. Seshia and E. A. Lee, "An Introductory Textbook on Cyber-Physical Systems," in *Workshop on Embedded Systems Education (in conjunction with ESWeek)*, Scottsdale, AZ, October 2010.

[13] J. C. Jensen, E. A. Lee, and S. A. Seshia, "Teaching Embedded Systems the Berkeley Way," in *Workshop on Embedded Systems Education (in conjunction with ESWeek)*, Tampere, Finland, October 2012.

[14] ——, "An Introductory Capstone Design Course on Embedded Systems," in *International Symposium on Circuits and Systems (ISCAS)*. Rio de Janeiro, Brazil: IEEE, 2011, pp. 1199–1202.

[15] ——, *An Introductory Lab in Embedded and Cyber-Physical Systems*. Berkeley, CA: LeeSeshia.org, 2012. [Online]. Available: http://LeeSeshia.org/lab

[16] D. Sadigh, S. A. Seshia, and M. Gupta, "Automating exercise generation: A step towards meeting the MOOC challenge for embedded systems," in *Workshop on Embedded Systems Education (in conjunction with ESWeek)*, Tampere, Finland, October 2012.

[17] J. C. Jensen, D. H. Chang, and E. A. Lee, "A model-based design methodology for cyber-physical systems," in *First IEEE Workshop on Design, Modeling, and Evaluation of Cyber-Physical Systems (CyPhy)*, Istanbul, Turkey, 2011. [Online]. Available: http://chess.eecs.berkeley.edu/pubs/837.html

[18] J. C. Jensen, "Elements of model-based design," Master's thesis, University of California, Berkeley, February 2010. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-19.html

[19] P. Derler, E. A. Lee, and A. Sangiovanni-Vincentelli, "Modeling cyber-physical systems," *Proceedings of the IEEE (special issue on CPS)*, vol. 100, no. 1, pp. 13–28, 2012.

[20] R. Smith. Open dynamics engine. [Online]. Available: http://ode.org

[21] J. Kodosky, J. MacCrisken, and G. Rymar, "Visual programming using structured data flow," in *IEEE Workshop on Visual Languages*. Kobe, Japan: IEEE Computer Society Press, 1991, pp. 34–39.

[22] Massachusetts Institute of Technology (MIT), "The iLab Project," Last accessed: March 2013. [Online]. Available: \url{https://wikis.mit.edu/confluence/display/ILAB2/Home}

[23] National Instruments, "LabVIEW 2012 AMI," January 2013. [Online]. Available: https://aws.amazon.com/marketplace/pp/B00B04MUFI

[24] Amazon Web Services. (2013, January) Amazon EC2 Pricing. [Online]. Available: http://aws.amazon.com/ec2/pricing/

[25] Microsoft, "Remote Desktop Protocol Performance," January 2010. [Online]. Available: http://www.microsoft.com/en-us/download/details.aspx?id=23236

[26] Y. Annpureddy, C. Liu, G. E. Fainekos, and S. Sankaranarayanan, "S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems," in *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2011, pp. 254–257.

[27] A. Donzé, "Breach: A Toolbox for Verification and Parameter Synthesis of Hybrid Systems," in *Computer-Aided Verification*, 2010, pp. 167–170.

[28] X. Jin, A. Donzé, J. Deshmukh, and S. A. Seshia, "Mining requirements from closed-loop control models," in *Hybrid Systems: Computation and Control (HSCC)*, 2013, to appear.

[29] W. Li, A. Forin, and S. A. Seshia, "Scalable specification mining for verification and diagnosis," in *Design Automation Conference (DAC)*, 2010, pp. 755–760.

[30] W. Li and S. A. Seshia, "Sparse coding for specification mining and error localization," in *Proceedings of the International Conference on Runtime Verification (RV)*, September 2012.

[31] R. Singh, S. Gulwani, and A. Solar-Lezama, "Automated feedback generation for introductory programming assignments," in *Programming Languages Design and Implementation (PLDI)*, 2013, to appear.

[32] M. Mozgovoy, T. Kakkonen, and G. Cosma, "Automatic student plagiarism detection: Future perspectives," in *Journal of Educational Computing Research*, vol. 43, no. 4, 2010, pp. 511–531. [Online]. Available: http://baywood.metapress.com/openurl.asp?genre=article&id=doi:10.2190/EC.43.4.e