A Preliminary Report of Integrating Science and Computing Teaching Using Logic Programming

Yuanlin Zhang,[1] **Jianlan Wang,**[1] **Fox Bolduc,**[1] **William G. Murray,**[1] **Wendy Staffen**[2]
[1]Texas Tech University, [2]Laura Bush Middle School, USA
{y.zhang, jianlan.wang, Fox.Bolduc, William.G.Murray}@ttu.edu, wstaffen@lcisd.net

Abstract

This paper presents a framework to integrate Science and Computing teaching using Logic Programming. We developed two modules: one for chemistry and the other for chemistry and physics. They are implemented in an elective course for 8th graders. Through clinical interviews, video taped class observations, exit interviews and our own experiences with the class, Logic Programming based approach is accessible to the students.

Introduction

There is an urgent need for a well-prepared workforce in STEM and Computing in this century. Computing is not only a STEM discipline in its own right but also a discipline integral to the practice of all other STEM disciplines. (STEM education act, 2015). There is consensus on the need of integrating computing in STEM teaching and learning. However, we know very little about how best we can teach Computing and how to integrate it with STEM disciplines to improve STEM and Computing learning.

The mainstream programming systems used in K-12 are based on tinkering oriented visual languages such as Scratch (Resnick and Rosenbaum 2013). This approach has demonstrated some benefits of integration. However, more research is needed to understand how thinking occurs as students are tinkering (Guzdial 2004). Second, text-based languages have the advantage over visual languages of "taking students deeper into both programming and science" (Sengupta et al. 2015; DiSalvo 2014). But very little is known about integration using text-based languages.

In this paper we propose to use Answer Set Programming (ASP), a modern version of Logic Programming (LP), to integrate computing into science teaching in middle school. In the rest of the paper, we will first give a description of our integration methodology and argue why it will facilitate learning of science and computer science. We will then share our experience on developing and implementing modules on chemistry and physics for 8th graders.

Integration of Computing and Science Teaching

To integrate Science and Computing teaching, we employ a methodology with two (often iterative) sequential components:

1. **Problem Description.** Teach students a new science topic (problem) or review a learned topic (problem). Students are expected to answer basic questions in this topic and understand why their answers are correct.
2. **Modeling.** Ask students to build a computer model using LP. Note that we use LP and ASP interchangeably in this paper. The model is expected to answer the questions in the problem descriptions.

We will use food chain as an example to illustrate both the methodology and LP.

Problem description. Food chain is a science topic in middle school. Consider a chain with carrots, rabbits, snakes and eagles. Typical questions include "Q1: do eagles eat snakes?" and "Q2: what would happen to eagles if snakes become extinct?" Students are expected to review or learn food chains and how these questions can be answered.

Modeling. To design a computer model to answer the questions above, we follow an LP modeling methodology which consists of two steps.

1. Identify *objects* and *relations* in the problem.
2. Identify *knowledge* in the problem and write *LP rules* for this knowledge. The final LP rules, also called a *program*, form the model of the problem.

Objects. The objects here are four species of organisms, which can be represented in LP by the following sort declaration:

```
#species = {eagle, snake, rabbit, carrot}.
```

Note that each species is taken as an object here. `#species` is called a *sort name*.

Relations. According to question Q1, we identify a relation of the form $feedsOn(X, Y)$ which means that *members of species X feed on those of species Y* . In question Q2, we introduce a relation $extinct(X)$ which means that *species X is extinct*.

Knowledge and LP Rules. In this part, we explicate the science knowledge needed to answer the questions in English and then "translate" that knowledge into LP rules. The *declarative nature* of LP allows for a natural translation. For example, in the given food chain, we know that "rabbits feed on carrots", which can be translated, using the relation introduced earlier, into

$r1 : feedsOn(rabbit, carrot)$.

which is called a *fact*, a simplest form of *LP rule*. $r1$ is the label of the rule which may be referred to later. Similarly, we have the knowledge that "snakes feed on rabbits" and "eagles feed on snakes" which are translated respectively into facts:

$r2 : feedsOn(snake, rabbit)$.

$r3 : feedsOn(eagle, snake)$.

The collection of rules above forms an *LP program* which can be used to answer question Q1. A query $feedsOn(rabbit, X)$, where X is a variable (in the standard sense of a variable in algebra/math), asks the program to find an organism that the rabbits in the chain feed on. The correct answer is carrot. Figure 1 gives an idea of *onlineSPARC*, an online LP programming environment (http://goo.gl/ukSZET) (Reotutar et al. 2016). Area 1 (in red ellipse) is an editor containing the program above, and area 2 contains the query $feedsOn(rabbit, X)$. When the "submit" button is pressed, the answer shows in area 3.

To answer question $Q2$, we add the knowledge that snakes are extinct which is represented as

$r4 : extinct(snake)$.

We also need some more general knowledge: "a species will be extinct if what it feeds on is extinct." This knowledge can be represented by an LP rule of the form:

$r5 : extinct(X)$:- $feedsOn(X, Y), extinct(Y)$

where the symbol ":-" is understood as "if." The rule is read as, from left to right, for any species X, X is extinct if X feeds on Y and Y is extinct. (Note the rule is an accurate representation of the knowledge in food chains, but needs to be refined when a food web is modeled.) With these newly added rules, the LP program concludes that eagles are extinct too. We have covered almost all major constructs of ASP. We hope the examples above demonstrate the simplicity of ASP and the naturalness of the modeling and show how students are focused on domain knowledge. One can also see that the lesson design method is a direct result of the LP modeling methodology, and produces a seamless integration of Science and Computing.

The Integration Facilitates Science and Computer Science Learning

We will argue how the LP based integrative curriculum may facilitate learning of Science and Computer Science at middle school level. Note that the learning outcomes are achieved by both the Science content and its LP based computer models. We do not argue that LP by itself creates transfer learning in other domains (Klahr and Carver 1988).

Model-based learning is well accepted in science education. It is anticipated to help students' "attainment of 'conceptual understanding' in science at a level that goes beyond memorized facts, equations, or procedures." (Clement 2000). It is well recognized that building computer models or STEM problems helps STEM education, too (Harel and Papert 1990; Guzdial 1994; Wilensky and Reisman 2006; Council and others 2011; Repenning, Webb, and Ioannidou 2010; Sengupta et al. 2013; Jona et al. 2014) In fact, Harel and Papert (Harel and Papert 1990) pointed out that learning computing together with another subject can be more effective than learning each separately.

To illustrate how LP-based integration will facilitate Science learning, we use the framework for K-12 science education(Council and others 2012). The framework articulates a vision of the scope and nature of K-12 education in science, engineering, and technology. It has been implemented by NGSS (Next Generation Science Standards) which had been adopted by 16 states. NGSS are in line with TEKS – Texas Essential Knowledge and Skills – which will be followed in our implementation in Texas.

The framework divides the fundamental, core skills for science and engineering into eight practices. SP1: asking questions and defining problems. SP2: developing and using models. SP3: planning and carrying out investigations. SP4: analyzing and interpreting data. SP5: using mathematics and Computational Thinking. SP6: constructing explanations and designing solutions. SP7: engaging in argument from evidence. SP8: Obtaining, evaluating, and communicating information.

Our integration is able to cover the majority of the eight practices. As shown in our integration example above (and our curriculum in next section), students have to ask and answer questions before building a computer model. Hence, $SP1$ is in a prominent position in our curriculum. Our curriculum is driven by developing computer models for science problems and thus $SP2$ will be practiced intensively under our integration methodology. Instead of mainly through tinkering (Resnick and Rosenbaum 2013), students are encouraged to identify the knowledge and represent it into rules of computer models. Hence, $SP3$ and $SP6$ (solution design) are addressed in our integration. When testing and debugging their computer models, students have to re-examine the program and apply the logical reasoning to explain the program behavior. Hands-on programming creates an environment greatly motivating students' discussions where explanations and arguments are heavily involved. Therefore, $SP6$ (explanation) and $SP7$ are well represented in our integration. As required in our modeling methodology, students have to identify the knowledge used in modeling, represent it into English and then translate the English description into rigorous rules. Our integration provides explicit and rigorous training of students in terms of $SP8$. In fact, LP-based integration facilitates students' learning of mathematics and computer science (see below for details). Students have abundant opportunities to practice $SP5$ when building computer models.

As for mathematics, our integration helps address some core practices as identified in the Common Core State Standards for Mathematics (Initiative and others 2010). MP2: reason abstractly and quantitatively. MP3: construct viable arguments and critique the reasoning of others. MP6: attend to precision. As argued before, when developing and test-

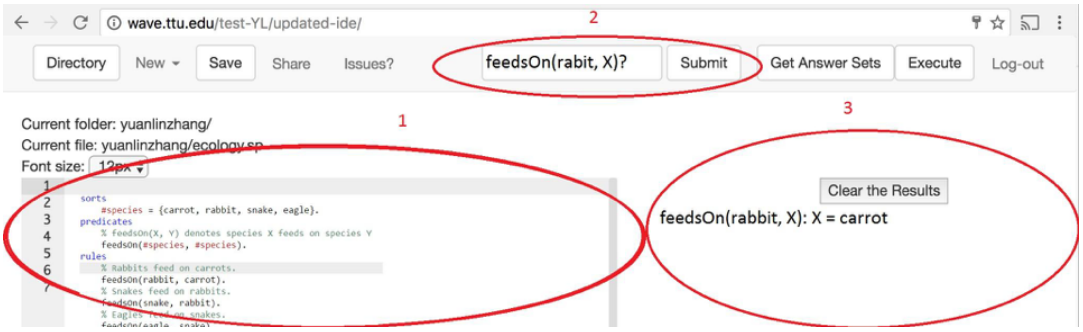

Figure 1: Screenshot of onlineSPARC

ing the computer models (e.g., that for food chain), all these practices are explicitly involved.

As for computer science, LP provides a great way to teach core computing practices of *abstracting, problem solving, programming* and *communicating*, as defined in AP Computer Science Principles (CollegeBoard 2017) and K-12 Computer Science Framework (2017) which is followed by CSTA standards (2017).

The identification of relations and knowledge and the translation of knowledge into rules are a clear practice of *abstracting*. As a well-established programming paradigm, LP offers a setting for students to learn and practice all aspects of *programming*: design the model (program), edit the program, learn the (informal yet rigorous to a great extent) syntax and semantics, coding, testing and debugging. As shown in our methodology and example, model development starts from problem description. Hence *problem solving* is at the core of our integration. In our modeling methodology, students are required to identify (and write down) the knowledge needed to solve the problem. They get an explicit training on *communicating* ideas and knowledge.

Appropriateness for Middle School Students

According to Piaget's (1972) theory of cognitive development, for example, children at age 11 to 15 demonstrate logical use of symbols related to abstract concepts. At this age, students have sufficient experience in life and STEM disciplines to allow for flexibility with regard to problem selection. The value of LP in teaching children has been recognized since 1980s (Kowalski 1982; Guzdial 2004) and practiced by Kowalski and colleagues to teach children at age 10 to 12 (Kowalski 1982).

Curriculum Description

We develop two integration modules for 8th graders. Module 1 is on chemistry, and module 2 on chemistry and physics.

Module 1

Module 1 has 8 lessons (50 minutes a lesson). We assume students have learned periodic table.

Lesson 1 introduces computer science and computer models. It consists of two parts. Part 1 contains the videos on motivating students on computing and its applications. One video is *Computer Science is Changing Everything* by code.org (2016). The videos are followed by a discussion on computer science and our daily life to further motivate students' interest in computer science. In part 2, by asking students questions about their classroom, school and family, we introduce the concept of models that human beings may use to answer questions. Using human thinking as an analogy, we introduce the concept of computer models for problem solving. We then provide students an LP model for the family in the earlier discussion. Students play with the model by asking the computer the questions they were asked earlier and extending the model with new knowledge. Through this hands-on experience, students are expected to develop an understanding of computer models.

Lesson 2 introduces LP concepts of *relations, facts*, and *queries*. It first reviews the chemical symbols for elements. It shows students how to add facts using relations to an existing model provided to them. For example, to develop a model for the symbol for Hydrogen, students need to expand the given model with a comment about the knowledge:

```
% The symbol for Hydrogen is H
```

and then the fact representing it:

```
symbolFor(hydrogen, h).
```

Queries are introduced to answer questions to the model. For example, for the question "is H the chemical symbol of Hydrogen?" We type the query in onlineSPARC

```
symbolFor(hydrogen, h)?
```

Students then extend the model with knowledge from other elements including carbon and phosphorous, and test the model using queries.

Lesson 3 reviews new topics of *atomic numbers* and *mass numbers*. Students are expected be able to answer questions on the atomic number and mass number of Hydrogen and other elements. It then introduces relations needed to answer these questions. Students expand the given model with facts about the new knowledge on atomic numbers and mass numbers, and then test the model with queries.

Lesson 4 first introduces students to variables using queries. E.g., for question "what is the chemical symbol for element silicon?", we need a query

```
symbolFor(silicon, What)?
```

where `What` is a variable. The students also learn the answer to the question is of the form `What = si`. Students

practice variables by writing queries for similar questions about other elements. This lesson then introduces the relation $protonsOf(E, N)$ which denotes that the number of protons of the atom of element E is N. The lesson concludes with challenging the students to extend a given model with facts representing the knowledge of the protons of hydrogen.

Lesson 5 introduces the concept of rules. It first reviews knowledge relating the number of protons of an atom to its atomic number: The number of protons of the atom of an element E is N if N is the atomic number of the element E. It then shows the rule for this knowledge:

```
protonsOf(E, N) :- atomicNumber(E,N).
```

Students will extend a given model by this rule. Then they are asked to write a rule for the knowledge that derives the atomic number from the number of protons. Throughout the lesson the students are tasked with testing models with queries.

Lesson 6 and 7 introduce more complex rules. Students review domain knowledge on how the number of neutrons of an atom relates to the atom's mass number and protons. An English description of this knowledge is: N is the number neutrons of an atom E if M is the mass number of the atom E, and $N = M - P$. The lesson then shows the rule

```
neutronsOf(E, N) :- massNumber(E, M),
    protonsOf(E, P), N = M - P.
```

The students extend a given model with the rule above and further by a rule defining the mass number of an element using its number of neutrons and protons.

Lesson 8 continues the practice of writing rules. Students review domain knowledge of obtaining the number of electrons of an atom from that of the protons of the atom or the atomic number. Students are asked to write a rule for the English description of that knowledge such as "N is the number of electrons of an atom E if N is the number of protons of E." The students also review the rule for getting the neutron number from the number of protons and mass number of the atom. The students are expected to test the model with queries.

Module 2

Module 2 uses the same chemistry content in module 1, but aims to teach students to write complete programs (instead of expanding existing programs). It also introduces a physics topic for students to model. It has 6 lessons.

Lesson 1 teaches student to create and save a file in OnlineSPARC (so that they can develop a program across sessions). Then it teaches students how to define a sort using an example. Students are then expected to add new objects to an existing sort and finally to create a sort on their own. Predicate declarations are then introduced in this lesson.

In lesson 2 students learn to declare new predicates on their own and add facts to their programs using the methodology they learned in module 1. Students are then instructed to write more facts for the predicates that they declared earlier in the lesson.

Lesson 3 Students learn to write a rule for this knowledge: *the number of protons of the atom of an element E is N if N is the atomic number of element E.* The rule is written as

```
protonsOf(E, N) :- atomicNumber(E, N).
```

Students are asked to test the rule using a query.

Lesson 4 evaluates students' learning by asking them to write new sorts, predicates, and rules to model the electron number of an element. Students are given minimal guidance during this lesson.

Lesson 5 introduces students to motion, a new science topic to model. Students are expected to create the program themselves. The first half of the lesson introduces the concepts in motion through a physical demonstration. They include *distance change* and *motion of an object relative to another one*. Students define sorts and declare predicates needed to model the problem.

Lesson 6 continues the modeling of motion by adding facts. The teacher starts by asking students to write a fact for answering questions such as "Has the distance between Bob and Clara changed?". Students add more facts on distance change according to the situation in the physical demonstration in the previous lesson. Students then test the model by asking queries. Finally we evaluate students' learning by asking them to relate distance change to motion of an object relative to another one, and create predicates and rules to model this relation.

Experience

We offered module 1 in fall 2017 to an 8th grade class, at Laura Bush Middle school, with 22 students (5 of whom are female and 10 of whom are Hispanic or African American). Module 2 was offered in spring 2018 to the (almost) same class with 21 students (4 of whom are female and 9 of whom are Hispanic or African American). 19 students are in both classes. Module 1 was taught from Oct 23 to Nov 14 2017 and module 2 from from Apr 18 to May 8. Both classes are for an elective course with name of *STEM* and are taught by Staffen, one of the authors. In general, the course *STEM* could cover any topics of STEM that is not covered by the science and math course. Our integration of science and computing fits this course well. The class meets for 50 minutes every two week days. Some sessions are canceled due to school schedule, and one session in spring 2018 was skipped due to the unavailability of Staffen. The last session in 2018 was used for interview. Staffen had one year teaching experience in middle school Science and STEM before fall 2017. We will share our experience in curriculum development, its implementation and student learning below.

Curriculum Development

Module 1 is the first integration module we ever developed. When designing the module, we assume the students have learned the concepts related to periodic table. (The 8th graders started to learn periodic table in fall 2017 in their Science class.)

For the content of periodic table, the author Wang and Zhang consult Staffen and other science teachers of the same school, the TEKS standard on learning outcomes of chemistry for 8th graders, STAAR (State of Texas Assessments of Academic Readiness) tests on science, and a textbook following TEKS. By performance data on STAAR, students performed poorly on some problems related to periodic table. For example, a general question is: *what is the total*

number of protons, neutrons and electrons in the atom of an element that has a given mass number? An instance of this question on cadmium with mass number of 112 appears in STAAR. Only 35% students answer it correctly (Lead4ward 2017).

For Logic Programming, we resort to our research and teaching experiences on it and textbooks on Logic Programming such as (Gelfond and Kahl 2014; Clocksin and Mellish 2003).

We discover that almost all problems on periodic table in the references we use can be modeled effectively by ASP. We built ASP models for major concepts in chemistry at 8th grader level from periodic table to reaction equations. The models help us to develop a holistic view of chemistry and of its computer modeling. As a result, we hope our module, although covering only periodic table, will be laying a good foundation for students for their future study of chemistry and its modeling. We also document in details our understanding of the chemistry concepts and computer modeling for future development of the integration module. We then start to develop slides for module 1.

With the established structure for module 1 and experience in integration, the development of module 2 is much faster.

Implementation

Teacher Preparation. In the module development phase, for each lesson, we get feedback on all aspects of the lesson such as appropriateness of content in terms of students' capacity and time constraint from Staffen. Staffen goes through the slides, workbook and all programming steps, and meet Wang and Zhang, authors of this paper, before each lesson.

Classroom Teaching. The teaching occurs at a computer lab where each student has a desktop computer. The lab also has a projector, a white board and enough space for students to sit around the instructor. The instructor uses *Google classroom* to make all information available to student.

For fall 2017, Staffen teaches the first four sessions while Wang and Zhang assist (and observe) in class. Staffen co-teaches with Zhang on next three sessions and with Wang on the last session. In co-teaching, Wang and Zhang are on content and Staffen on class management. After each session, Wang and Zhang meet to identify problems and make revisions for the next session. They finalize the revision after meeting with Staffen.

For spring 2018, Staffen teaches all sessions except one which is co-taught with Zhang.

The general class activities can be taken as an iteration of a sequence of science component and computing component. When reviewing the science topics, we usually ask the students to sit around the instructor and make sure they are able to answer the questions and understand the reasoning behind the answers. For the computing part, continuous refinements are made on how to teach the computing concepts and how to carry out the hands-on programming.

Hands-on programming seems to help engage students well and thus is a substantial component of every session. There are a few challenges related to hands-on programming.

1) How to balance the teaching of computing concepts and hands-on programming. We tried different balances. One extreme is: the instructor demonstrates programming while explaining the computing concepts, and the students follow the demonstration on their own computer. A better balance, agreed among the instructors (Staffen, Wang and Zhang), is the following. We ask students to sit around the instructor. We introduce, in a period of 5 to 10 minutes, some computing concept(s) to the students and make sure they can answer questions related to these concepts. For example, after we show students that $symbolFor(hydrogen, h)$ can be used to represent the knowledge"the symbol for Hydrogen is H," we ask them to represent "the symbol for Carbon is C." We will then further ask students to tell us some similar knowledge and then represent it into ASP rules. Then, students will get back to their computers and start coding and testing the rules just discussed. In this way, students have a better understanding of the concepts, and is more effective in programming.

2) We note a big variation in students' programming performance during class. Some students can finish and go beyond given programming tasks. For example, they can write tens of facts on the symbols of elements in the first lessons while some students are lost without knowing what to do (partly due to lack of attention and partly due to the numerous details needed for programming). This variation can severely interrupt the flow of the class and the learning of the students. To overcome this challenge, we introduce workbook which is organized by programming activities and detailed information is provided for each activity. For example, students usually need to copy and paste some given program segments into their editors. An activity is designed for this task. In this activity, the link of given program segments and instructions on how to copy and paste are given. When lost, a student is pointed to the right activity and then follows the instructions. To reduce the performance variation, we also introduce groups where students can help each other and compete for credits. The groups seem to work well.

From Staffen and our interview with some students, students of this class are less motivated and disciplined than those of an average class. We employ some standard methods to enhance class management. The use of groups also seems to help to engage more student who are not able to focus on class otherwise.

Student Learning

Clinical Interview – Fall 2017. To assess how well students understand computing, Staffen selects two students from each performance group: above average, average, and below average. The student performance is decided by Steffen's class observation. We conduct a clinical interview for each student on Dec 1, 2017. The interview is done by a researcher (one of authors: Bolduc, Wang and Zhang) and a participant. Participants are asked to work out one problem while thinking aloud. Researchers ask probing questions in order to prompt participants to explain and clarify their thought processes. Clinical interviews last 30 minutes or un-

til the participants complete their task – whichever come first. Clinical interviews are video recorded. The problem used in the interview can be found in the appendix. It tests student's capacity in using the onlineSPARC tool, abstraction at three levels (1: relation on concrete objects; 2: variables; and 3: rules), and programming (including syntax and debugging). The problem consists of two parts. The first part is about periodic table and is closely related to the class discussion, while the second part is about a family which is only touched in the first two lessons. The second part aims to test if students have a deeper understanding of computing in a relatively new problem domain.

All students are doing well in using the onlineSPARC environment, in abstraction at levels of relation on concrete objects and variables, and programming, except the following cases. One student P (in below average group) did not do well on abstractions at all three levels and on syntax. The student mentions that he missed several sessions. One student Q (in average group) did not do well on abstractions at the levels of variables and rules. All students show satisfactory understanding in abstraction at rule level except students P and R (in below average group).

In summary, students demonstrate a good understanding of the computing concepts. Students are weaker in abstraction at the rule level. There are three reasons. First, the time span from the end of the intervention to the interview date is long, about two weeks due to scheduling problems. Second, abstraction at rule level is tested in a new domain (not periodic table). Finally, students did not have experience on solving a problem in a formal setting such as a clinical interview.

Video-taped Class Observation. We filmed two groups for each class in spring 2018. Group 1 consists of two above average students and Group 2 consists of two average students. They are selected by Staffen. We were not able to film the first lesson.

By the end of lesson 2, both groups understand objects and sorts. They are able to add additional objects to a sort as well as declare entirely new sorts. Both groups are also able to add new facts for a given predicate.

During lessons 3 and 4 students begin encountering more errors related to predicate declarations. Both groups make mistakes by writing predicate declarations like rules

```
protonsOf(#element, #number) :-
  atomicNumber(#element, #number).
```

One possible explanation for this error is that students are confused between the declaration of a predicate and the definition of a predicate using rules.

When either group encounters any error, their usual response was to press the "Get Answer Sets" button which provides a more specific error message. The students would then check the line number provided in the error message. Despite often not knowing what the error message was saying, they usually find the error on the line they are directed to.

By lesson 5 both groups are extremely competent at defining new sorts. Group 1 even gets this done before being asked to model motion - the new science topic. Both groups are also able to correctly declare a simple predicate. Group 1 is able to get a fully functioning program working and tested by the end of class. Group 2 has a syntactically correct program, however it does not make an attempt to test it.

In lesson 6 Group 2 is able to write a rule involving negation:

```
-motion(alice, daniel) :-
  -distanceChange(alice, daniel).
```

However, both groups are still making the mistake of declaring predicates like rules

```
motion(#object, #object) :-
  distanceChange(#object, #object).
```

Group 1 also demonstrates creativity in trying to represent that the distance between Bob and everyone else changes. But they fail to represent it as a rule. Instead, they try the following:

```
distanceChange(bob, O),
O = {bob, alice, chair, clara, daniel}
```

Exit Interview. We also conducted an interview with group 1 and 2 in the class on May 10, 2018, and it was video recorded. Here is a summary of key points we learn from this interview. All participants mention that students of this class are much below average in terms of the their attitude and discipline. However, they mention that one student in the class is able to get everything done. The participants believe that the continuation of the study using the same chemistry content as in the previous semester helps to improve their understanding of the computing significantly. When asked if using science topics to learn computing is useful, they mention that the familiarity of the science topics help their computer modeling. Two of them mention that in this manner, they see the meaningful application of computing to Science topics. One said that it is more interesting when applying what they are learning in science to programming. As for if programming helps them understand the science topics better, one mention that it offers a new way to look at things [science topics], one mention that programming makes one to think back [in science topics], and one mention that programming makes learning Science concepts more interesting. During the interview, it is mentioned that LP programs in Science context make much more sense than programs they have learned in JavaScript [partly because of the topic and partly because of the programming language used]. Some mention that writing complete program in this semester helps them better understand LP-based modeling. Some mention their early confusion between predicates and rules and now a better understanding of the three sections of a program: sorts, predicates and rules.

Related Work

The mainstream programming systems used in K-12 are based on tinkering oriented visual languages such as Scratch (Resnick and Rosenbaum 2013). These languages are very successful in reaching a large audience. However, more research is needed to understand how thinking occurs as students are tinkering (Guzdial 2004). Furthermore, text-based languages have the advantage over visual languages of "taking students deeper into both programming and science." (Sengupta et al. 2015; DiSalvo 2014). To make the application of computer science to STEM authentic (Shaffer and

Resnick 1999), the languages are supposed to allow students to write programs to solve problems in their STEM study. However, there is little research on text-based languages' integration in middle schools, because of the dilemma that most widely used languages in K-12 are algorithm-based and there is a gap between the language constructs and the scientific concepts (Sengupta et al. 2015). It should also be noted that learning algorithm-based languages is time consuming (Sherin, diSessa, and Hammer 1993).

To address the challenge above, we propose to investigate ASP, a text-based language, for the integration. There is a natural correspondence between the ASP (logical) constructs and the knowledge needed to solve a STEM problem. To build a computer model for a STEM problem, students only need to use ASP to specify the subject-matter knowledge needed without creating algorithms. Since ASP is also simple, students can use it to solve non-trivial STEM problems. As we argued in earlier sections, the ASP based integration has the potential to improve students' learning outcomes in both STEM and CT.

However, LP has been largely overlooked during the last two decades. In fact, in the 1980s, PROLOG, a representative of the classical LP, had been studied rather intensively for teaching because it is supposed to allow a declarative (i.e., logical) reading and understanding of a program (and thus easy for children) (Mendelsohn, Green, and Brna 1991). Unfortunately, PROLOG is not purely declarative and has a strong algorithmic (also called procedural) feature which requires comprehension of a difficult notional machine (Taylor and Du Boulay 1986) by learners. A major breakthrough in LP research (Gelfond and Lifschitz 1988; Gebser, Kaufmann, and Schaub 2012) in the last three decades was the establishment of ASP, which is purely declarative and removes the procedural feature of PROLOG. Thus, LP merits revisiting in the teaching context. This project starts a new research line on LP-based integration. The proposed research will advance our knowledge on how well LP-based approach improves STEM and computing learning outcomes. There is some recent work that studies LP in K-12 (Scherz and Haberman 1995; Stutterheim, Swierstra, and Swierstra 2013; Beux et al. 2015), but none of it applies purely declarative LP to STEM.

The benefits of declarative feature of a language were demonstrated by existing work including Kowalski (1982) where the declarative feature of PROLOG was emphasized, and recent integration BOOTSTRAP (Schanzer et al. 2015) where only "side-effect-free" functions and variables (which are also classified as declarative) are allowed in programming but procedural components such as mutable variables and assignment are forbidden. (BOOTSTRAP employs functional programming (FP). Although both FP and LP are declarative, LP offers a more straightforward support of knowledge representation and reasoning for many STEM problems. For example, the simple LP program in Figure 1. can be used to answer many interesting questions including "what does rabbit feed on," which is not the case for FP.)

Conclusion

In this research, we propose a methodology for developing modules to integrate the teaching of science and computing through developing computer models. Logic Programming plays an essential role in making this integration seamless and natural, and also makes it accessible to middle school students. We have developed and implemented two modules for 8th graders. Through clinical interviews, video taped class observations, exit interviews and our own experiences with the class, Logic Programming based approach is accessible to the students. Majority of the students are able to use the onlineSPARC environment, to abstract at different levels, and to program. The current study is limited. In the future, we plan to carry out more rigorous experiments and analysis on how LP based integration facilitates students' learning in science and computing.

Acknowledgment

We thank Michael Gelfond and Michael Strong for numerous discussions on this topic, and Edna Parr and Jeremy Wagner for their support in our implementation.

Appendix

Clinical Interview Problem

Q1-Q3, interviewer will copy the program testProgram-chemistry to online SPARC.

Answer the following questions

Q1. We know symbolFor(E, S) denotes that the symbol for element E is S. Ask the question "is na the chemical symbol for sodium?" to the model.

Q2. We know atomicNumber(E, N) denotes that the atomic number for element E is N. Ask the question "What is the atomic number for sodium?" to the model.

Q3. Extend the model by including the following knowledge: the atomic number of silicon is 14. Test if your model works.

Q4 - Q6, interviewer will copy the program the program testProgram-family to online SPARC.

Q4. We know mother(X, Y) denotes that person X is the mother of Y. Ask the question "is Joann the mother of John?" to the model. (Optional question to ask the model "Who is the mother of John?")

Q5. We know mom(X, Y) denotes that person X is the mom of Y. Extend the model by including the following knowledge: X is mom of Y if X is the mother of Y.

Q6. Test your model by asking the question Is Joann the mom of John?

References

Beux, S.; Briola, D.; Corradi, A.; Delzanno, G.; Ferrando, A.; Frassetto, F.; Guerrini, G.; Mascardi, V.; Oreggia, M.; Pozzi, F.; et al. 2015. Computational thinking for beginners: A successful experience using prolog. In *CILC*, 31–45.

Clement, J. 2000. Model based learning as a key research area for science education. *International Journal of Science Education* 22(9):1041–1053.

Clocksin, W. F., and Mellish, C. S. 2003. *Programming in Prolog: Using the ISO standard*. Springer Science & Business Media.

code.org. 2016. Computer science is changing everything. Video retrieved from https://www.youtube.com/watch?v=QvyTEx1wyOY on September 4 2018.

CollegeBoard. 2017. Ap computer science principles: course and exam descriptions. Retrieved from https://apcentral.collegeboard.org/pdf/ap-computer-science-principles-course-and-exam-description.pdf on September 4 2018.

Common Core State Standards Initiative. 2010. Common core state standards for mathematics. *http://www.corestandards.org/assets/CCSSI_Math%20Standards.pdf*.

CSTA. 2017. Csta k-12 computer science standards. Computer Science Teachers Association.

DiSalvo, B. 2014. Graphical qualities of educational technology: Using drag-and-drop and text-based programs for introductory computer science. *IEEE computer graphics and applications* 34(6):12–15.

Gebser, M.; Kaufmann, B.; and Schaub, T. 2012. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence* 187:52–89.

Gelfond, M., and Kahl, Y. 2014. *Knowledge representation, reasoning, and the design of intelligent agents: The answer-set programming approach*. Cambridge University Press.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *ICLP/SLP*, volume 88, 1070–1080.

Guzdial, M. 1994. Software-realized scaffolding to facilitate programming for science learning. *Interactive Learning Environments* 4(1):001–044.

Guzdial, M. 2004. Programming environments for novices. *Computer science education research* 2004:127–154.

Harel, I., and Papert, S. 1990. Software design as a learning environment. *Interactive learning environments* 1(1):1–32.

Jona, K.; Wilensky, U.; Trouille, L.; Horn, M.; Orton, K.; Weintrop, D.; and Beheshti, E. 2014. Embedding computational thinking in science, technology, engineering, and math (ct-stem). In *future directions in computer science education summit meeting, Orlando, FL*.

K-12 Computer Science Framework. 2017. http://www.k12cs.org. Retrieved on September 4 2018.

Klahr, D., and Carver, S. M. 1988. Cognitive objectives in a logo debugging curriculum: Instruction, learning, and transfer. *Cognitive Psychology* 20(3):362–404.

Kowalski, R. A. 1982. Logic as a computer language for children. In *ECAI*, 2–10.

Lead4ward. 2017. http://lead4ward.com/docs/resources/rt/science/rt_science_gr_08.pdf. Retrieved on September 4 2018.

Mendelsohn, P.; Green, T.; and Brna, P. 1991. Programming languages in education: The search for an easy start. In *Psychology of programming*. Elsevier. 175–200.

National Research Council. 2011. *Report of a workshop on the pedagogical aspects of computational thinking*. National Academies Press.

National Research Council. 2012. *A framework for K-12 science education: Practices, crosscutting concepts, and core ideas*. National Academies Press.

Piaget, J. 1972. Intellectual evolution from adolescence to adulthood. *Human development* 15(1):1–12.

Reotutar, C.; Diagne, M.; Balai, E.; Wertz, E.; Lee, P.; Yeh, S.-L.; and Zhang, Y. 2016. An online logic programming development environment. In *AAAI*, 4130–4131.

Repenning, A.; Webb, D.; and Ioannidou, A. 2010. Scalable game design and the development of a checklist for getting computational thinking into public schools. In *Proceedings of the 41st ACM technical symposium on Computer science education*, 265–269. ACM.

Resnick, M., and Rosenbaum, E. 2013. Designing for tinkerability. *Design, make, play: Growing the next generation of STEM innovators* 163–181.

Schanzer, E.; Fisler, K.; Krishnamurthi, S.; and Felleisen, M. 2015. Transferring skills at solving word problems from computing to algebra through bootstrap. In *Proceedings of the 46th ACM Technical symposium on computer science education*, 616–621. ACM.

Scherz, Z., and Haberman, B. 1995. Logic programming based curriculum for high school students: the use of abstract data types. In *ACM SIGCSE Bulletin*, volume 27, 331–335. ACM.

Sengupta, P.; Kinnebrew, J. S.; Basu, S.; Biswas, G.; and Clark, D. 2013. Integrating computational thinking with k-12 science education using agent-based computation: A theoretical framework. *Education and Information Technologies* 18(2):351–380.

Sengupta, P.; Dickes, A.; Farris, A. V.; Karan, A.; Martin, D.; and Wright, M. 2015. Programming in k-12 science classrooms. *Communications of the ACM* 58(11):33–35.

Shaffer, D. W., and Resnick, M. 1999. " thick" authenticity: New media and authentic learning. *Journal of interactive learning research* 10(2):195–216.

Sherin, B.; diSessa, A. A.; and Hammer, D. 1993. Dynaturtle revisited: Learning physics through collaborative design of a computer model. *Interactive Learning Environments* 3(2):91–118.

Stutterheim, J.; Swierstra, W.; and Swierstra, D. 2013. Forty hours of declarative programming: Teaching prolog at the junior college utrecht. *arXiv preprint arXiv:1301.5077*.

Taylor, J., and Du Boulay, B. 1986. *Studying novice programmers: why they may find learning Prolog hard*. School of Cognitive Sciences, University of Sussex.

Wilensky, U., and Reisman, K. 2006. Thinking like a wolf, a sheep, or a firefly: Learning biology through constructing and testing computational theoriesan embodied modeling approach. *Cognition and instruction* 24(2):171–209.