LP Based Integration of Computing and Science Education in Middle Schools

Yuanlin Zhang
Texas Tech University
y.zhang@ttu.edu

Jianlan Wang
Texas Tech University
jianlan.wang@ttu.edu

Fox Bolduc
Texas Tech University
Fox.Bolduc@ttu.edu

William G. Murray
Texas Tech University
William.G.Murray@ttu.edu

ABSTRACT

There is a consensus on integrating computing with STEM teaching in K-12. However, very little is known about the integration. In this paper, we propose a novel framework for integrating science and computational thinking teaching using Logic Programming. We then develop and implement two 8-session integration modules on chemistry and physics for 6th and 7th graders. Pre- and post- tests, class observations and interviews show the feasibility of the framework in terms of 1) development and implementation of the modules, and 2) the students' learning outcomes on science content and Computational Thinking, and their acceptance of the integration.

CCS CONCEPTS

• **Social and professional topics** → **Computational thinking**; **K-12 education**; • **Computing methodologies** → **Logic programming and answer set programming**.

KEYWORDS

Logic Programming, Middle School Science, Computing

ACM Reference Format:
Yuanlin Zhang, Jianlan Wang, Fox Bolduc, and William G. Murray. 2019. LP Based Integration of Computing and Science Education in Middle Schools. In *ACM Global Computing Education Conference 2019 (CompEd '19), May 17–19, 2019, Chengdu, Sichuan, China.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3300115.3309512

1 INTRODUCTION

There is consensus on the need of integrating computing, integral to the practice of all other STEM disciplines, in STEM teaching and learning in K-12 (Kindergarten to 12th grade in the US education system) (see, e.g., [31, 41]). However, little is known about how best Computational Thinking (CT) can be taught and how to integrate it with STEM disciplines to improve STEM and CT learning in K-12 in general and middle schools in particular.

CompEd'19, May 17 – 19, 2019, Chengdu, Sichuan, China

ACM ISBN 978-1-4503-6259-7/19/05…$15.00
https://doi.org/10.1145/3300115.3309512

To develop effective, integrative curriculum, it is desirable to have frameworks on how CT can be integrated to STEM education to support both CT and STEM curricular topics and students' learning outcomes.

In this paper we propose a Logic Programming (LP) based framework for integration, called *LPK12*. LPK12 achieves a deep integration of CT and STEM education by building computer models for STEM problems through Answer Set Programming (ASP) [15] – a modern LP paradigm. LPK12 is based on the following arguments. First, LP has low floor and high ceiling [32]. It allows students to start developing computer models for interesting, non-trivial STEM problems after a very short introduction and yet it is a full-fledged programming paradigm. Second, LP facilitates a unified treatment of the fundamental skills and topics in STEM and Computing thanks to the fact that LP is based on discoveries and ideas of Logic which forms an important base for learning and problem solving in all STEM disciplines. The LP modeling methodology allows a natural and seamless connection of subject-matter concepts and reasoning to computer model development. Thirdly, middle school students are cognitively ready for LP based approaches. By Piaget [33] and Vygotsky [44], children at age 11 to 15 demonstrate substantial knowledge of natural language and the logical use of symbols related to abstract concepts. Finally, for STEM, LPK12 facilitates students to develop fundamental skills, as defined in next generation science standards (NGSS) [30], such as *asking questions* and *defining problems*, *constructing explanations*, *engaging in argumentation*, and *communicating information*. For Computing, students will get abundant opportunities to learn and practice *various levels of abstraction*, *problem solving*, *programming* and *communication* as identified in the K-12 Computer Science Framework [22].

The rest of the paper is organized as follows. We present the LPK12 framework in Section 2 before the discussion of the related work in Section 3. In Section 4, we introduce the design of the study of the feasibility of LPK12 and the procedure of data collection and analysis. In Section 5, we present the data in alignment with our research questions. The paper is concluded by the last section.

2 THEORETICAL FRAMEWORK

2.1 LP Based Integration of Computing and Science Teaching

To integrate STEM and Computing teaching, we employ a methodology with two (often iterative) sequential components: (1) **Problem Description.** Teach students a new or learned STEM topic

(problem). Students are expected to answer basic questions in this topic and understand why. (2) **Modeling.** Ask students to build a computer model using LP. 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 chains are a science topic taught 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 of the food chain problem. 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 in the food chain problem. From question Q1, we identify a relation of the form $feedsOn(X, Y)$ meaning 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 an *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 the facts: $r2 : feedsOn(snake, rabbit)$ and $r3 : feedsOn(eagle, snake)$. 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 (X) 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) [34]. 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 is shown 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 from left to right as *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 demonstrate the simplicity of ASP and the naturalness of the modeling and how the modeling focuses on domain knowledge. One can also see that LP, together with its modeling methodology, produces a seamless integration of Science and Computing.

2.2 LPK12 Facilitates STEM and CT Learning

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" [5]. It is well recognized that building computer models for STEM problems helps STEM education too [10, 16, 19, 21, 35, 40, 45]. In fact, Harel and Papert [19] 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 STEM learning, we use the framework for K-12 science education[11]. 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 has been adopted by 16 states in US.

The NGSS framework divides the fundamental, core skills for science, engineering and technology 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.

LPK12 is able to cover the majority of the eight practices. As shown in our integration example in Section 2, students have to ask and answer questions before building a computer model. Hence, $SP1$ is in a prominent position in LPK12. LPK12 is driven by developing computer models for STEM problems and thus $SP2$ will be practiced intensively under our integration methodology. In LPK12, as shown in Section 2.1, students are encouraged to identify the knowledge and represent it as rules for 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 logical reasoning to explain the program behavior. Therefore, $SP6$ (explanation) and $SP7$ are well represented in our integration. As required in LP modeling methodology, students have to identify the knowledge used in modeling, express it in English and then translate it into rigorous rules. Hence, our integration helps students to practice $SP8$ (communicating information). Practice $SP5$ will be elaborated below on mathematics and computing separately.

As for mathematics, our integration helps address some core practices as identified in the Common Core State Standards for Mathematics [20]. 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

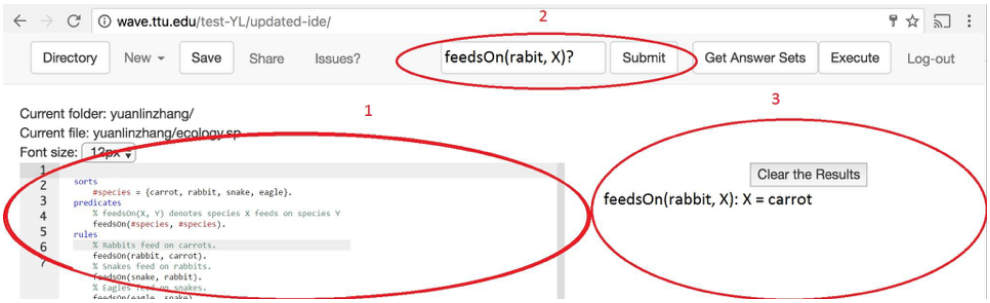

Figure 1: Screenshot of onlineSPARC

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

As for computing, LP covers the core practices of *abstracting, problem solving, programming* and *communicating*, as defined in an AP course [8] and standards ([12, 22]). The identification of relations and knowledge and the translation of knowledge into rules are a clear practice of *abstracting*. As a programming paradigm, LP offers the practice of all aspects of *programming*: model (program) design, program editing, (informal yet rigorous to a great extent) syntax and semantics, coding, testing and debugging. As shown in Section 2.1, model development starts from problem description. Hence *problem solving* is at the core of our integration. As argued for STEM, *communicating* is well covered by LPK12.

2.3 Appropriateness for Middle School Students

By Piaget [33] and Vygotsky [44], children from 11 to 15 demonstrate logical use of symbols related to abstract concepts. LPK12 also supports learning progression across multiple years as required in both STEM and CT [2, 9, 39] because of the easy integration with STEM topics. LPK12 also lends itself to well-accepted pedagogies such as *scaffolding* (because LP methodology explicates the knowledge and skills needed in problem solving in STEM) and *inquiry based learning* (because LP integration is driven by building computer models for answering questions – see Section 2 and 4.2). Due to space limitation, we are not able to elaborate on the above here.

3 LITERATURE REVIEW

Logic Programming Research and Its Use in Education. Born in the 1960s, Logic Programming is a meeting point of Thinking, Logic and Computing [24, 26]. It has been studied for teaching children since the 1980s [25] because it is supposed to allow a declarative (i.e., logical) reading and understanding of a program (and thus easy for children) [17, 29]. Unfortunately, the strong *procedural component* of classical LP systems such as PROLOG [29, 43] has prevented LP from reaching a wider audience although there have been efforts in the last two decades to include it in high school curriculum [3, 38, 42] and undergraduate teaching [27]. A major breakthrough in the last two decades is the establishment of Answer Set Programming – a *purely declarative* LP paradigm [15]. ASP is now a major paradigm in Knowledge Representation and Logic Programming community [24] with numerous applications across many areas [14]. This breakthrough merits a revisiting of LP in teaching because it eliminates the *procedural component*.

Programming Systems in K-12. The mainstream systems used in K-12 is based on visual programming environments such as Scratch and Alice [23, 28]. The majority of systems encourage and facilitate tinkering and fit the needs of students who prefer *tinkering* to *logic and planning* [32, 36]. These languages and environments have been very successful in reaching a large K-12 population. However, more research is needed to understand how CT occurs as students are tinkering while using visual programming languages [17, 28]. The majority of the teaching modules, based on the visual programming languages, adopt open-ended contexts such as game design and storytelling for ad hoc STEM topics [2], which makes the alignment with curricular topics difficult.

Integration of Computing and STEM Teaching. Some challenges for integrating computing into STEM in K-12 are how to align with STEM and CT curricula topics and how to support students' learning progression (usually across multiple years) in both STEM and CT [2, 9, 39]. To develop effective integration curriculum, it is desirable to have integration frameworks to support curricular alignment and learning progression. One of the few works in this direction is [40] which proposes the use of agent-based computation to integrate CT and science. We also note a rigorous study on how the integration may improve students' learning of a specific topic in mathematics [37]. However, this study does not present a general framework on the integration.

Our Contribution. As far as we are aware, we are the first to propose the use of LP, a purely declarative programming paradigm, to integrate computing and STEM. The majority of integration (including [40]) is based on *imperative programming paradigms*. We note functional programming, also declarative, is used in integration, e.g., [37]. However, the work in [37] is on a specific topic but not on a general framework. The main advantages of LP over imperative languages are as follows: 1) LP is simple in both syntax and semantics (see Section 2); 2) the fact that LP is based on discoveries and ideas of *Logic* which also forms an important basis for learning and problem solving in all other disciplines provides a more straightforward connection between LP based models and STEM problem solving; and 3) the high level of abstraction of LP allows one to "hide" many machine-related details when solving a problem. It is also observed that text-based languages have the advantage over visual languages of "taking students deeper into both programming and science" [13, 39]. LP is a text-based language.

4 STUDY DESIGN

To study the feasibility of LPK12, we developed two LP-integrated modules and applied the exploratory case study method to pilot the exploration of the efficacy of the modules [1].

4.1 Participants and Context

This study took place in an elective course, with the name "STEM", of a middle school with 900 students in grades 6th - 8th which is located in a middle-south city in the United States. The participants were one STEM teacher and her four sections of 96 6th-graders and three sections of 71 7th-graders. Among the 6th (and 7th respectively) graders, there were 61 (and 51) males and 36 (and 20) females. The ethnographic composition of 6th graders was 60 Whites, 28 Hispanics, 5 Blacks, 2 Asians, and 1 American Indian; and that for 7th graders was 43 Whites, 22 Hispanics, 3 Blacks, and 3 Asians.

4.2 LP Based Integration Modules and Implementation

We developed a chemistry module (periodic table) for 6th graders and physics module (motion) for 7th graders.

4.2.1 Chemistry Module. Lesson 1 introduces computer science in general and computer models in particular. Part 1 includes motivating videos such as *Computer Science is Changing Everything* by code.org (2016) and discussions following these videos. In part 2, by asking students questions about their classroom, school, and a 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 LP based computer models for problem solving. Students would interact with a model by asking the same questions they were asked in the lesson and extending the model with new knowledge.

Lesson 2 introduces LP concepts of *relations, facts*, and *queries* using examples. It first reviews the chemical symbols for elements. Students will then extend a given model by adding facts, e.g., on the symbol for Hydrogen. They first type a comment "`% The symbol for Hydrogen is H`" and then the fact "`symbolFor(hydrogen, h).`" *Queries* are introduced to answer questions to the model. 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 number* and *mass number*. The teacher will introduce relations needed to answer questions in those topics. Students will then expand a given model with facts about the new knowledge on atomic number and mass number, and then test the model using queries.

Lesson 4 introduces *variables* using queries. E.g., for question "what is the chemical symbol for the element silicon?', we need a query "`symbolFor(silicon, What)?`" where `What` is a variable. Students practice variables by writing queries for similar questions about other elements. A new relation $protonsOf(E, N)$ is introduced to denote that the number of protons of the atom of element E is N. Finally, students are challenged to extend a given model with facts representing the knowledge of the protons of hydrogen.

Lesson 5 introduces *rules*. It reviews knowledge relating proton number to 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 representing it:

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

Students extend a given model by this rule and test it. They then practice by writing a rule for knowledge on getting atomic number from proton number and to test it.

Lesson 6 and 7 introduce more complexity to the rules. It reviews domain knowledge relating the number of neutrons to mass number and proton number: N is the number neutrons of an atom E if M is the mass number of the atom E, and $N = M - P$. It is represented as

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

where "," between relations means conjunction. The students extend a given model with the rule 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 and reviews domain knowledge relating the number of electrons of an atom to that of protons or the atomic number. Students are asked to write and test a rule for this knowledge. Another exercise is to represent the knowledge on getting neutron number from proton and mass numbers.

4.2.2 Physics Module. This 8-lesson module has a similar structure to the chemistry one. Due to lack of space, here we only give the physics problem involved and its modeling information. The physics concepts covered are *target object*, *reference object*, *distance change* between two objects and the target object's *motion relative to* the reference object. They are introduced by an experiment where one student will move a chair with another student on it. During and after the experiments, questions about the concepts are asked, discussed and explained. To model the problem, we introduce the relations $isTarget(X)$ (i.e., X is a target object), $isReference(X)$ (i.e., X is a reference), $distanceChange(X, Y)$ (the distance changes between X and Y), and $moving(X, Y)$ (X is moving relative to object Y). One piece of knowledge used is: an object X is moving relative to object Y if the distance between X and Y changes. (This knowledge may need to be refined in a rigorous setting or multiple dimensions.) We also introduce the classical negation $\neg$ in this module.

4.2.3 Implementation. Each class session consists of one or several cycles. Each cycle consists of two components: concept understanding (by lecturing and discussion with a duration of 5-10 minutes) and programming practices. Slides are designed to facilitate the lecturing and discussion, and workbooks are designed to contain detailed information to guide the students on their programming practices. Programming involves too much information and any ignorance of any information by students will frustrate them and interrupt the class flow. The workbooks make it easy for students to review or find information they need.

4.3 Research design

This study lasted for four weeks of totally 8 50-minute long periods in spring 2018. Due to the exploratory nature of this study, we selected topics that the participating students had learned prior to the intervention. The goal is to measure students' learning outcomes in science content and computational thinking.

4.4 Data collection and analysis

We administered pre- and post-surveys to examine the learning outcomes. They contain multiple-choice questions assessing students' scientific content knowledge of interest and computer science skills. The science questions have been previously validated. Since LP is new in teaching CT, the questions are designed by the researchers by following the guide in [4]. Each question is graded as either 1 (correct) or 0 (incorrect). The total score is the sum of scores from all question. The questions from the pre- and post-surveys are substantially the same and vary only in different context. For instance, the questions in the pre-survey for the 7th-graders are about a boy on a swing and those in the post-survey are about a car leaving a garage. The purpose of not using the same questions in the post-survey is to suppress the possibility of students answering questions through memorization.

Here are some questions for physics. Pre-question. A boy is playing on a swing in a park. His mom stands still by the swing. There are several other kids playing on a slide in the same park. In reference to the boy, is the swing moving? A. Yes, B. No, C. Not sure, more information is needed. Post-question. A boy is sitting in a car. His mom is reversing from the garage. His dad is standing still in the garage and saying bye to them. In reference to the boy, is the car moving? A. Yes, B. No, C. Not sure, more information is needed. Here is a question for measuring abstraction in Computing. We know `protonNumber(E, N)` means that the proton number for element `E` is `N`, `electronNumber(E, N)` means that the electron number for element `E` is `N`. Write a rule to represent the following knowledge: the electron number of `E` is `N` if the proton number of `E` is `N`. A. the electron number of E is N if the proton number of E is N. B. `protonNumber(E, N) :- electronNumber(E, N).` C. `electronNumber(E, N) :- protonNumber(E, N).` D. None of the above.

Besides descriptive statistics, we applied paired t-tests on the surveys for the pre-post comparison. Meanwhile, we calculated Cohen's d [7] and normalized gain [18] to measure the effect size. Cohen's d describes the width of the impact on students from the integration modules (small 0.2, medium 0.5, large 0.8) and normalized gain describes the magnitude of the impact in terms of the ratio of the actual progress made by students to the maximum progress that students could make (small <0.3, medium 0.3-0.6,large >0.7).

We also carried out one post group interview for each grade to gauge the students' feedback to their experience with the integration modules. The interviews were semi-structured. We recruited volunteer 6th and 7th graders with a diverse background in terms of gender, race, science content knowledge and CT understanding (so that they are representative to the greatest extent – See Table 1).

Table 1: Background information of the interviewees

	S7-1	S7-2	S7-3	S7-4	S7-5	S7-6	S7-7	S7-8	S6-1	S6-2	S6-3	S6-4	S6-5	S6-6	S6-7
G	7	7	7	7	7	7	7	7	6	6	6	6	6	6	6
S	M	F	M	M	M	M	F	M	M	M	M	M	F	F	M
R	A	W	W	W	H	W	W	W	W	H	W	W	H	W	W
Sc	3	4	1	2	3	4	4	3	2	3	3	2	2	1	1
CT	3	3	3	2	3	4	4	3	2	3	2	4	3	2	3

G: grade; S: sex; R: race; M: male; F: female; Sc/CT: Science/CT score out of 4; S7-i, S6-i: pseudo code for students; A: Asian; H: Hispanic; W: White.

The interviews were audio-recorded and transcribed later. Codes were developed to identity students' feedback to the integration modules. We coded the interviewees' comments from three perspectives: *module being interesting*, *computing being impactful to science learning*, and *science being impactful to computing learning*. For each category, we coded an interviewee's response as "Yes" (i.e., admitting that statement), "No" (denying that statement), and "N/A" (i.e., not mentioning it in the interview).

5 FINDINGS

5.1 Q1. How did the integration modules affect the students' learning?

We summarized in Table 2 the data about pre-post comparison. Both the 6th- and 7th-grade students developed their CT (abstraction) significantly with Cohen's d being large and normalized gain being medium, which indicates that the modules have helped most of the participating students make considerable progress in CT skills. Similarly, the 7th graders' physics content knowledge increased significantly. The effect size indexes (Cohen's d=1.15,<g>=0.61) suggest that most students developed their understanding about the relative nature of motion to a considerable extent after the integration modules. However, the 6th graders' chemistry content knowledge decreased from 1.55 to 1.35. This decrease might happen by chance because it is not statistically significant. Some possible reasons might be that the students had different statuses while answering those questions, and students may be more serious to answer the questions in pre-survey. Some extra instruments may be needed to figure out the reasons in the future study.

Table 2: Paired t-tests of assessments on science and CT

	Pre	SD	Post	SD	|t|(df)	p	C-d	<g>
6-C	1.55	0.95	1.35	0.88	0.89(73)	ns	-0.22	-0.08
6-CT	1.14	1.02	2.51	0.99	10.46(70)	***	1.36	0.48
7-P	1.62	1.37	3.07	1.16	6.71(55)	***	1.15	0.61
7-CT	1.73	1.17	2.61	1	4.37(56)	***	0.82	0.39

ns: not significant; ***: p<0.001; Pre and Post: mean score with a max score of 4; 6-C: chemistry; 7-P: physics; C-d: Cohen's d; <g>: normalized gain

The quantitative data does not cover students' learning of programming and communication. They are examined by class observations and the interviews. For programming, students are expected to repeat given models and extend them with new knowledge and rules during class. According to the class observation by the teacher and researchers, a majority of the students are able to complete both types of tasks. We will discuss next the interview results on the programming and communication aspect of CT.

From the interviews, students show positive experience with programming and LP. For example, "*I think it was really easy. Because you're basically just saying it in English, but just like a different. Well.. It's like pretty much the same thing*" by S7-7. "*I really liked the questions for the query because if you get your answer yes your like hard work paid off or if it didn't you need to like go through all your steps again*" by S6-5 (and similarly S6-4). The favorite part of the class is "*the coding part*" by S7-3 (and agreed by S7-1, S7-4, S7-5 and S7-6). "*My favorite part is I think just learning new coding styles in general is cool to me because we went from symbol forward to atomic*

number to number of neutrons to protons and I'm just excited for what we learned" by S6-7. When comparing other programming paradigms such as drag and drop, "*this time around, we actually we had to learn what to say to do it. I mean in fourth grade all we did was pretty much move like right forward back but here we actually had to learn is like protons, neutrons, electrons stuff like that for sparc [LP system used in our implementation], which I think is actually better on that because we can learn more .... I mean it's a lot harder and more informative if you actually do [LP programming]*" by S6-2.

Debugging is an important element of programming. Students seem to be able to do debugging and appreciate its value. Here are example excerpts. Comment by S6-5 above on programming. "*When it [the model/program] just doesn't give you an answer and then you realize you did something wrong and you go back and you look over it [program]*" by S6-6. "*when you make a mistake you get to go back and then while you're retyping something or redoing something you can look and make sure that you get like if I got let's say lithium and I went back next to and accidentally only put a top or a symbol for it is as LI can go back and put it as li and that helps me more because the more and more I do it and the more I make mistakes the more I will know to put the right or correct answer*" by S6-7. When something is wrong, "*I got through all the steps that I took and if I missed a step or something I would go back and fix it and it would work again*" by S6-6. When asked if they tried to find errors when their program does not answer a query correctly, S7-1, S7-2, S7-3 and S7-4 answered yes while S7-5 no. As for locating errors, "*ask a query of a certain line of code. And if it [LP system] comes up ... the opposite [to the expected] response then ... you can go through that specific line of code and see what's wrong*" by S7-2.

For CT practice communication, in our teaching, we always start from English description of knowledge and then write rule(s) to represent the knowledge. As a result, we observed that majority of students always write English description before writing any rules. When commenting about this methodology "*I think they help ... because you're actually reading it and if you don't understand something you can read it again maybe ask a couple questions*" by S6-6. Another example is the comment by S7-7 above. Given the precision needs of computer models, the writing of the description and translating it into rule(s) will improve students' rigorousness in communication.

5.2 Q2. How the participants reflect on their experience with the LP based integration?

Most of the interviewees commented positively on their experience with the integrated modules. 13 out of 15 interviewees thought that the modules were interesting (S7-7 was coded as "No" and S7-6 as "N/A"). 14 out 15 interviewees believed that computing impacted science learning (S7-3 is coded as "No"), and vice versa (S7-6 is coded as "No"). Thus, the LP based integration of science and CT is accepted by the students. S6-4 commented that science and computing can support each other: "*I think it can be fun just like learning about new like elements that like mean this is technology, but we're also learning about like other stuff in our world not just kind of ... it's like a win-win.*" "*instead of just learning it [motion], but putting it into something you can do like teaching the computer, that makes it more fun*" by S7-7. "*[computing] makes it more interesting because you're not just looking at a whole lot of boxes on a piece of paper [periodic table]*" by S6-5. "*[Computing is interesting] when we had to do the like the number of neutrons equals the number of protons like you add both of those to get the uh, atomic mass*" by S6-3. "*You don't really notice you are learning the chemistry and stuff until you are done with it [modeling] ...*" by S6-1. When talking about modeling chemistry problems, "*you might feel more professional because you're doing something that professionals would do*" by S6-4.

6 CONCLUSION

We propose an LP based framework to integrate computing and STEM teaching in K-12. To test its feasibility, we have developed two integration modules for 6th and 7th grades. By our experience, the framework allows a rather straightforward development of the modules (see Section 4.2). We conjecture that the development of modules for other topics in science, based on our modules, will also be straightforward. Our survey data show that students' learning has been improved on physics and CT (abstraction) significantly. Class observations and interviews show that students are doing well in the other aspects of CT: programming and communication. Interviews also show the LP based integration is accepted by students: the modules are interesting and there is a positive impact of computing and science to each other.

In summary, LP based integration seems to be promising in terms of the development and implementation of the two 8-session modules, the students' learning outcomes.

There are some limitations in this preliminary study. 1)It is difficult to test students' development of problem solving capacity, a CT practice, because the short duration of the project. Our framework allows a curriculum with much longer duration (see Section 2.3). We will plan experiments with longer duration in future. 2) Other variables are not controlled. For example, we did not control the students' access to resources of both science and computational science. Thus, we cannot ascribe the observed pre-post differences solely to the LP-integrated modules as there might be other variables that took effects. 3) Our survey data did not show improvement on students' learning outcome on chemistry. One explanation is that two of the four test questions involve complex relations and calculations. One question is "*How many protons, neurons and electrons are present in an atom of hafnium, Hf, with a mass number of 178, and an electron number of 72?*" We focus on the declarative knowledge, but it may still be challenging for 6th graders to calculate the right answer from their knowledge. In future, clinical interviews will be used to understand students' performance change. 4) We do not have a quantitative measurement on students' learning outcomes on programming and communication of CT practices. In future, we will develop and validate quantitative assessment and rubrics to more rigorous measurement of students' outcomes on these CT practices. Finally, our future work will be exploring how to integrate the LP-integrated method into core STEM courses seamlessly.

ACKNOWLEDGMENTS

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

REFERENCES

[1] Donald Ary, Lucy Cheser Jacobs, Christine K Sorensen Irvine, and David Walker. 2018. *Introduction to research in education.* Cengage Learning.

[2] Satabdi Basu, Gautam Biswas, Pratim Sengupta, Amanda Dickes, John S Kinnebrew, and Douglas Clark. 2016. Identifying middle school studentsâĂŹ challenges in computational thinking-based science learning. *Research and Practice in Technology Enhanced Learning* 11, 1 (2016), 13.

[3] Silvio Beux, Daniela Briola, Andrea Corradi, Giorgio Delzanno, Angelo Ferrando, Federico Frassetto, Giovanna Guerrini, Viviana Mascardi, Marco Oreggia, Francesca Pozzi, Alessandro Solimando, and Armando Tacchella. 2015. Computational Thinking for Beginners: A Successful Experience using Prolog. *Proceedings of the 30th Italian Conference on Computational Logic* (2015).

[4] Philip Sheridan Buffum, Eleni V Lobene, Megan Hardy Frankosky, Kristy Elizabeth Boyer, Eric N Wiebe, and James C Lester. 2015. A practical guide to developing and validating computer science knowledge assessments with application to middle school. In *Proceedings of the 46th ACM technical symposium on computer science education*. ACM, 622–627.

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

[6] code.org. 2016. Computer Science is Changing Everything. Video retrieved from https://www.youtube.com/watch?v=QvyTEx1wyOY on September 4 2018.

[7] Jacob Cohen. 1988. Statistical power analysis for the behavioral sciences. 2nd.

[8] 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.

[9] Thomas B Corcoran, Frederic A Mosher, and Aaron Rogat. 2009. Learning progressions in science: An evidence-based approach to reform. (2009).

[10] National Research Council et al. 2011. *Report of a workshop on the pedagogical aspects of computational thinking.* National Academies Press.

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

[12] CSTA. 2017. CSTA K-12 computer science standards. Computer Science Teachers Association.

[13] Betsy DiSalvo. 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 (2014), 12–15.

[14] Esra Erdem, Michael Gelfond, and Nicola Leone. 2016. Applications of answer set programming. *AI Magazine* 37, 3 (2016), 53–68.

[15] Michael Gelfond and Yulia Kahl. 2014. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents.* Cambridge University Press.

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

[17] Mark Guzdial. 2004. Programming environments for novices. *Computer science education research* 2004 (2004), 127–154.

[18] Richard R Hake. 1998. Interactive-engagement versus traditional methods: A six-thousand-student survey of mechanics test data for introductory physics courses. *American journal of Physics* 66, 1 (1998), 64–74.

[19] Idit Harel and Seymour Papert. 1990. Software design as a learning environment. *Interactive learning environments* 1, 1 (1990), 1–32.

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

[21] Kemi Jona, Uri Wilensky, Laura Trouille, MS Horn, Kai Orton, David Weintrop, and Elham Beheshti. 2014. Embedding computational thinking in science, technology, engineering, and math (CT-STEM). In *future directions in computer science education summit meeting, Orlando, FL.*

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

[23] Caitlin Kelleher and Randy Pausch. 2005. Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers. *ACM Comput. Surv.* 37, 2 (June 2005), 83–137.

[24] Robert Kowalski. 2014. Logic Programming. *Computational Logic, Volume 9 (Handbook of the History of Logic)* (2014).

[25] Robert A Kowalski. 1982. Logic as a computer language for children. In *ECAI*. 2–10.

[26] Robert A Kowalski. 1988. The early years of logic programming. *Commun. ACM* 31, 1 (1988), 38–43.

[27] Hector J. Levesque. 2012. *Thinking As Computation: A First Course.* The MIT Press.

[28] Sze Yee Lye and Joyce Hwee Ling Koh. 2014. Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior* 41 (2014), 51–61.

[29] Patrick Mendelsohn, TRG Green, and Paul Brna. 1991. Programming languages in education: The search for an easy start. In *Psychology of programming*. Elsevier, 175–200.

[30] NGSS Lead States. 2015. Next generation science standards: For states, by states. National Academies Press.

[31] NSF. 2018. STEM+C Program. https://www.nsf.gov/funding/pgm_summ.jsp?pims_id=505006, retrieved on October 10 2018.

[32] Seymour Papert. 1980. *Mindstorms: Children, computers, and powerful ideas.* Basic Books, Inc.

[33] Jean Piaget. 1972. Intellectual evolution from adolescence to adulthood. *Human development* 15, 1 (1972), 1–12.

[34] Christian Reotutar, Mbathio Diagne, Evgenii Balai, Edward Wertz, Peter Lee, Shao-Lon Yeh, and Yuanlin Zhang. 2016. An Online Logic Programming Development Environment.. In *AAAI*. 4130–4131.

[35] Alexander Repenning, David Webb, and Andri Ioannidou. 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*. ACM, 265–269.

[36] Mitchel Resnick and Eric Rosenbaum. 2013. Designing for tinkerability. *Design, make, play: Growing the next generation of STEM innovators* (2013), 163–181.

[37] Emmanuel Schanzer, Kathi Fisler, Shriram Krishnamurthi, and Matthias Felleisen. 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*. ACM, 616–621.

[38] Zahava Scherz and Bruria Haberman. 1995. Logic programming based curriculum for high school students: the use of abstract data types. In *ACM SIGCSE Bulletin*, Vol. 27. ACM, 331–335.

[39] Pratim Sengupta, Amanda Dickes, Amy Voss Farris, Ashlyn Karan, David Martin, and Mason Wright. 2015. Programming in K-12 science classrooms. *Commun. ACM* 58, 11 (2015), 33–35.

[40] Pratim Sengupta, John S Kinnebrew, Satabdi Basu, Gautam Biswas, and Douglas Clark. 2013. Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework. *Education and Information Technologies* 18, 2 (2013), 351–380.

[41] STEM education act. 2015. Public Law No: 114-59.

[42] Jurriën Stutterheim, Wouter Swierstra, and Doaitse Swierstra. 2013. Forty hours of declarative programming: Teaching Prolog at the Junior College Utrecht. *arXiv preprint arXiv:1301.5077* (2013).

[43] Josie Taylor and Ben Du Boulay. 1987. Studying novice programmers: why they may find learning Prolog hard. In *Computers, Cognition and Development: Issues for Psychology and Education*. John Wiley, 153–173.

[44] Lev S Vygotsky and Lev Seminovitch Vygotski. 1987. *The collected works of LS Vygotsky: Volume 1: Problems of general psychology, including the volume Thinking and Speech.* Vol. 1. Springer Science & Business Media.

[45] Uri Wilensky and Kenneth Reisman. 2006. Thinking like a wolf, a sheep, or a firefly: Learning biology through constructing and testing computational theoriesâĂŤan embodied modeling approach. *Cognition and instruction* 24, 2 (2006), 171–209.