Informatics in Education, 2014, Vol. 13, No. 2, 323–332
© 2014 *Vilnius University*
DOI: http://dx.doi.org/10.15388/infedu.2014.11

Teaching Some Informatics Concepts Using Formal System

Sojung YANG, Seongbin PARK[1]
Department of Computer Science Education, Korea University
Anam-Dong, Sungbuk-Ku, Seoul, 136-701, Korea
e-mail: cherry089@korea.ac.kr, hyperspace@korea.ac.kr

Received: January 2013

Abstract. There are many important issues in informatics and many agree that algorithms and programming are most important issues that need to be included in informatics education (Dagienė and Jevsikova, 2012). In this paper, we propose how some of these issues can be easily taught using the notion of a formal system which consists of axioms and inference rules by which theorems can be proved. As is argued in (Dagienė and Jevsikova, 2012), we can introduce important topics in informatics using puzzle-like examples and students do not need to have prerequisites for learning. The materials presented in this paper have been used in a college-level elective class titled Hypertext and Computability in our university since the fall semester of 2008 and we believe that the contents proposed in this paper can be easily used to teach beginner students without technical backgrounds.

Keywords: formal system, informatics education.

Introduction

In this paper, we discuss how concepts in informatics such as computation, modeling, etc can be introduced to students who do not major in informatics-related disciplines using formal systems.

A formal system (Alagar and Periyasamy, 2011) is a structure that consists of the following components:

(1) A decidable set (i.e., there is an algorithm that can tell whether an arbitrary element is a member of the set or not) of expressions called well-formed formulas (wffs).
(2) A decidable set of axioms which are wffs that are assumed to be true.
(3) A set of truth-preserving transformations, called inference rules.

[1] Corresponding author

A theorem is a wff that is either an axiom or recursively constructed by applying inference rules to axioms or previously constructed theorems. A proof is a finite sequence of wffs such that each wff is either an axiom or derived from previous wffs according to inference rules. The set of proofs in a formal system is decidable, but the set of theorems in a formal system is recursively enumerable (Nelson, 1967).

Formal systems have been used in different areas. (Day, 2012) considers evolution as a system defined on the natural numbers and an evolutionary theory is interpreted as a set of inference rules that can derive statements about those numbers. (Tautu, 1976) views a cellular system as a formal system by interpreting cell states as elements of an alphabet and strings over the alphabet as assemblies of cells in various states. (Bittner and Frank, 1997) explains how a geographic space can be formalized using formal systems and discusses an application of a formal theory to implement a geographical information system. While a lot of research has been done on applications of formal systems in various areas, little work has been done on how formal systems can be used to teach some informatics concepts especially for beginners in the field.

Materials presented in this paper have been used successfully in a class, Hypertext and Computability in our university which intends to introduce various issues about modeling, analysis, and problem solving skills to college students as a general elective class since the fall semester of 2008. The class meets twice a week and the duration for each lecture is one and a quarter hour. The entire semester consists of 14 lectures and approximately 5 lectures are used for teaching the materials described in this paper. Most of the students who take this class do not have experiences about computer programming and their majors are not informatics-related disciplines. Based on our experiences in this class, we believe that some concepts in informatics can be easily introduced to students without backgrounds on computer programming and beginner students (Dagienė and Jevsikova, 2012) as well using the notion of a formal system. So the theme of this research is to provide easy-to-use contents in informatics using the notion of a formal system in such a way that beginners or students who have some interests in informatics can easily understand some concepts in informatics.

This paper is structured as follows. Section 2 describes how important subjects in informatics such as computability and programming can be introduced to students using the notion of a formal system. Section 3 explains feedbacks from the students who have taken the class. Finally, section 4 concludes the paper.

2. Possible Topics

There are various topics in informatics that can be introduced using formal systems. In this section, we explain how some of these issues can be taught to college students whose majors are not informatics-related disciplines. Specifically, there are three main subjects that we can teach. First, we teach the notion of computation and related issues such as P vs NP problem and unsolvable problems using simple formal systems. Second, we introduce modeling using formal systems. Third, we explain the issue of completeness in the context of a formal system.

2.1. *Computation*

Informally, computation can be defined as a finite sequence of doing something. While the formal definition of computation can be explained using Turing machine (Linz, 2006), we believe that it can be easier to start with a formal system because students can understand the notions of computation and algorithm using relatively simple examples such as MIU system (Hofstadter, 1999) defined as follows. A wff is any positive length string over {M, I, U}. MI is the only axiom and there are four inference rules:

(1) If a wff ends with I, U can be added at the end of the wff.
(2) If the form of a wff is Mx, where x is any wff, then Mxx can be created as well.
(3) A triple of I (i.e., III) in a wff can be replaced by U.
(4) UU in a wff can be deleted.

Using the MIU system, students can learn what is meant by mechanically deriving a theorem from an axiom. For example, to derive MUI, one needs to start from the only axiom, i.e., MI and by applying the inference rules, one can see that MI becomes MII which becomes MIIII which eventually becomes MUI. This finite sequence (i.e., MI, MII, MIIII, MUI) is an example of computation. Of course, the sequence itself does not mean anything; what we can get across to students with the example is that computation is merely a finite sequence of doing something. So the process of adding a finite number of integers can be an example of computation regardless of the result that we can get. In other words, adding three numbers, 2, 3, and 4 can be done in sequence 2 + 3 and (2 + 3) + 4 and it is possible that the result can be 10 if we make a mistake. This is still an example of computation (i.e., wrong computation) and students can get the image of computation clearly. For example, they can understand that there are many examples of computation in our daily life. To be qualified as computation, it needs to have a starting point, an ending point, and the ending point is reachable from the starting point in a finite sequence.

Using the MIU system, we can also discuss whether all wffs in the MIU system can be derivable inside the system. This is not possible in MIU system since there is a wff such as MU which cannot be derivable from MI and successive applications of inference rules. To demonstrate this directly is not easy because it requires checking all possible inference rules to MI in an exhaustive way, yet it cannot guarantee whether MU is not derivable from MI in a finite number of steps or not. In fact, we can point out that we need to analyze the properties of the inference rules of the system in order to show that any theorem in the MIU system needs to start with M. However, it is only a necessary condition to become a theorem in the system because there is a wff that starts with M such as MU which is not a theorem (Gibson and Méry, 1998).

At this point, we can introduce the famous P vs NP problem which can be defined as follows: does there exist an efficient algorithm that takes an undirected graph as input and tells whether it contains a complete subgraph of half the nodes in the original graph, where an efficient algorithm refers to an algorithm whose running time is bounded above by a polynomial in the input size (Sipser, 1992). One way to introduce the problem using a formal system is to start with a simple formal system where students can see the difference between proving a theorem and verifying that a given proof is correct. Proving a theorem can be cast as a search problem where we are to find a proof in the space defined by axioms and inference rules while verification involves mechanically checking whether each component of the proof in sequence (i.e., a wff) follows from previous wffs in the sequence. This will lead to discussions about what can be describable in a formal system, what can be provable in a formal system, etc. On top of this, students can understand a consequence of a possibility that P is different from NP from the perspective of a formal system; i.e., there are theorems for which proving is harder than verifying (Goldreich, 2012). This can motivate students to imagine a possible formal system where the hardness of proving a theorem is the same as that of verifying a theorem.

Deriving a theorem T is an example of computation in the sense that it is a finite sequence from a starting point (i.e., a theorem) and an ending point (i.e., T). In fact, finding a proof for a theorem is an example of problem solving (Goldreich, 2012) and we can smoothly introduce situations where we cannot have a finite sequence that leads to an ending point from a starting point even though both starting and ending points are clearly defined. A relevant computational problem (i.e., Theoremhood testing problem) can be defined as follows. Given a set of axioms, inference rules, and an arbitrary proposition P, determine whether P can be proved as a theorem or not. For some formal systems, this problem is decidable, but for others, it is undecidable. This means that there does not exist an algorithm that decides whether an arbitrary proposition is a theorem in a formal system or not.

Beginners or students who do not major in informatics may have difficulty differentiating an algorithm and computation and the Theoremhood testing problem can be a good example that illustrates the difference between an algorithm and computation. They can understand that saying that the Theoremhood testing in general is not computable is equivalent to saying that there does not exist an algorithm that can tell whether an arbitrary wff is a theorem or not in a formal system. At this point, we can point out that there are a lot of computational problems which share a common property; i.e., it is not possible to solve these problems algorithmically regardless of how much time or space is given. Once students understand that there are algorithmically unsolvable problems, we can explain the Halting problem which asks whether an arbitrary program P can halt after a finite number of steps when executed with an arbitrary input for P and the reason why there cannot exist a universal anti-virus program that can detect whether a program that executes with an input is a computer virus or not (Lowther and Shene, 2004) using an intuitive argument that can lead to a contradiction if such a program exists.

2.2. *Modeling*

A wff of a formal system can be defined using a grammar (Linz, 2006) and writing a wff is similar to programming (i.e., the activity of writing a computer program) because both a wff and a computer program should have clear meanings and each of these needs to be defined according to a syntactic rule (Fitting, 2007) as well. This analogy can be also explained by comparing the process of figuring out whether a given wff can be derivable as a theorem in a formal system and checking whether a computer program is syntactically correct. Here, the correspondence is that a computer program roughly corresponds to a theorem and the grammar of the programming language in consideration such as Java or C corresponds to the set of axioms and inference rules.

In fact, programming is related to a more general activity, modeling which is one of the fundamental issues in computer science education (Majherová, 2007). Modeling involves identifying important features of objects that are to be modeled. For example, to solve a sudoku problem using a computer program, one needs to identify constraints (Lobo, 2007) and represent them in a way that a program can understand. This again involves two aforementioned issues: i.e., representation of constraints needs to be done in such a way that it is a syntactically correct form and it means what it intends to express.

Designing a formal system can be a good example of modeling exercises. In order to design a formal system, students need to define a set of axioms and inference rules by which some domain of interests can be modeled. Here, an axiom is any expression that describes a true fact in the domain under consideration. Figuring out candidate axioms requires a careful analysis on the domain. An inference rule should be selected in a way that it preserves truth. In other words, any transformation that converts a true fact into a true fact should be a candidate inference rule. Syntactically, an inference rule converts a wff into another wff and the truth remains the same semantically.

As an illustrative example of modeling an arithmetical property such as "2 plus 1 equals 3", we can use pq- system which is defined as follows (Hofstadter, 1999). A wff is any positive length string over {p, q, -}, and there is one axiom schema that generates an infinite number of axioms: xp-qx-, where x stands for a string of positive number of -'s. There is one inference rule: if xpyqz is a theorem, then xpy-qz- is a theorem, where x, y, and z are strings of positive number of -'s, respectively. Any theorem of this system can mean an arithmetical property if we interpret the theorem as follows: p, q, and the number of -'s in an arbitrary theorem is interpreted as "plus", "equal", and a numerical value that corresponds to the number of occurrences of -'s, respectively. For example, the theorem of this formal system, "--p-q---" can mean "2 plus 1 equals 3" with this interpretation.

2.3. *Completeness*

Once students get familiar with the notion of a formal system, we can explain a special type of formal system where one can show that there is a formal proof in the system for any true proposition expressible in the system (Mackenzie, 1995). One reason that we introduce a complete formal system to students lies in the fact that it can help students understand two different worlds defined in a mechanical way; i.e., the set of all true wffs in a formal system and the set of all theorems in a formal system. When designing a formal system for some domain of interests, presumably there can exist initial assumptions which can serve as axioms. From these, one can derive meaningful statements in the domain of interests by applying a set of inference rules. A complete formal system is one where all these true statements have proofs in the formal system.

One simple complete formal system (Gensler, 1984) can be defined as follows. A wff is defined as a positive length string over {/, +, (,), =} which takes on the form, $t_1 = t_2$, where t_1 and t_2 are terms that are defined as follows: (1) one or more /'s is a term (2) a string of the form, $(t_1 + t_2)$ is a term, where t_1 and t_2 are terms. There is one axiom schema that generates an infinite number of axioms: x = x, where x is a string of positive number of /'s. Let x and y be strings of positive number of /'s, respectively. Then, inference rules can be defined as follows: (1) (x+/) can be transformed into x/ (2) (x+/y) can be transformed into (x/+y). A wff is true if and only if it can be converted into a simple identity such as /=/, //=//, etc. using the following rules: (1) if a wff has a term of the form, (x+/y), then repeat changing the term to (x/+y) until it becomes a term of the form, (x+/). (2) if a wff has a term of the form, (x+/), then change the term to x/. A proof is a sequence of wffs, each of which is either a simple identity or else follows from previous wffs of the sequence by applying the inference rules.

It is easy to see that any true wff expressible in this system has a proof in the system. According to the definition of a true wff, any simple identity is a true wff and it can be directly derivable from the axiom schema. In addition, if we have a true wff which is not a simple identity, but can be eventually converted into a simple identity, then we can always find a proof for the wff. For example, we can show that a true wff, (//+/) = (/+//) has a proof as follows. First, according to the first inference rule, the lefthand side of the wff (i.e., (//+/)) becomes ///. Second, the righthand side of the wff (i.e., (/+//)) becomes (//+/) according to the second inference rule. Third, the righthand side of the wff (i.e., (//+/)) now becomes /// according to the first inference rule. Finally, we have a simple identity, /// = /// which can be directly derivable from one of the axioms.

When students understand the idea about a complete formal system, we can explain a formal system where some true statements cannot be proved; i.e., an incomplete formal system. We can mention the famous Gödel's incompleteness theorem as well, but explaining concepts necessary to describe and prove the theorem can be difficult for students who do not have taken advanced logic class. Specifically, one difficulty lies in the fact that it is not a simple matter to understand what is meant by carrying out a certain amount of arithmetic in a formal system (Franzén, 2005).

So, we can take a different approach; instead of explicitly defining an incomplete formal system where Gödel's incompleteness theorem applies, we use a formal system that has certain properties that are intuitive and easy to understand as follows (Smullyan, 1982):

(1) There are names of various sets of positive integers which can be arranged in an infinite sequence, $A_1, A_2, A_3, \ldots, A_n, \ldots$. A nameable set A has index n if $A = A_n$.
(2) For every pair of positive integers, x and y, there is a wff, x_A_y which is called a sentence that is true if x belongs to A_y and false otherwise.
(3) Every sentence is assigned its Gödel number and the Gödel number of x_A_y is denoted as $x \bullet y$.
(4) There are axioms and inference rules by which a sentence can be provable in the system. It is assumed that every sentence provable in the system is true.

Let P be the set of Gödel numbers of all the sentences provable in the system. For any set of positive numbers A, A^c denotes the set of numbers not in A and A^* denotes the set of all numbers x such that x • x belongs to A, respectively.

If this formal system satisfies the following additional three properties, then there is a true sentence not provable in this system.

(5) P is nameable in the system.
(6) For any nameable set A in the system, A^c is nameable in the system.
(7) For any nameable set A, A^* is nameable in the system.

Finding a true sentence that is not provable in this system can be a good exercise in deduction, but it may be difficult for students to know where to start. What we need is a sentence, x_A_y such that both the following two conditions are satisfied simultaneously: (1) x is indeed a member of the set A_y, so it is true (2) x • y is a member of P^c, so it is not provable. Let the set of numbers x such that x • x is a member of P^c be K. Then K is nameable because of the conditions (5) and (6). Therefore there is an index i such that K = A_i. Now, let's consider the sentence i_A_i. If this sentence is true, then i is a member of the set A_i and i • i is a member of P^c. In other words, if i_A_i is true, it is not provable. On the other hand, if i_A_i is false, then i is not a member of A_i and i • i is not a member of P^c, which means it is provable. Since a sentence that is provable is assumed to be true, it has to be the case that the sentence, i_ A_i is true and unprovable.

3. Feedbacks From Students

Our university has a course evaluation system where students can write comments about the class in general, and evaluate items such as contents of the leaning materials, whether students could acquire knowledge of the field, etc in an anonymous manner.

We have had positive feedbacks from students who took the class, Hypertext and Computability. For example, even though most students' majors were not in informatics-related disciplines, they did not have difficulty understanding the intuitive reason why a virus-detection problem is not solvable (i.e., there does not exist an algorithm that solves the problem). In other words, some important issues such as computability can be introduced to students without technical backgrounds on computer programming using a formal system and it can help them think further on structural differences between solvable problems and unsolvable problems. This is different from a situation where students merely understand the existence of unsolvable problems. Instead, they can become curious about why there cannot exist any algorithms for some problems, which can help stretch their imaginations.

There were students who showed intention to take a follow-up class of the class and one student took a discrete mathematics in our department in the following semester when he took the class. In addition, some students showed interests in working on additional exercise problems they could work on based on the materials that they learned in the class. As candidate exercise problems that students can work on, we believe the following types of problems are appropriate: (1) define a formal system for a board game such as tic-tact-toe, etc (2) find a model for a given logical theory that consists of wffs, where a logical theory is the set of true sentences in a model (Sipser, 2006). The first type of exercises are interesting because students can practice analyzing constraints that need to be satisfied and expressing them in terms of axioms and inference rules. The second type of exercises can help students understand possible consequences of adding axioms to a formal system. In other words, as more axioms are added, it becomes difficult to satisfy constraints specified in the form of an axiom and some models can be eliminated.

From the feedbacks that we have had for about 5 years since 2008, we strongly believe that important subjects in informatics can be introduced to students whose majors are not in informatics-related disciplines using the notion of a formal system. On top of this, our expectation is that almost the same materials that we have used in this class can be used to teach students in primary and secondary schools as well.

4. Conclusions

In this paper, we show how we can teach important, yet difficult issues in informatics to students whose majors are not in informatics-related disciplines using the notion of a formal system. The materials presented in this paper have been successfully used in a general elective class, Hypertext and Computability for about five years since the fall semester of 2008. Our experiences are that essential issues in informatics can be easily taught to non-major students using different kinds of formal systems. There are some advantages using a formal system to teach issues in informatics. First, it is a unified way by which various important issues in informatics can be taught. Second, it is not necessary to know how to write a computer program in order to understand why there are some computational problems that do not have algorithms (i.e., the existence of unsolvable computational problems).

Currently, we are working on ways by which different forms of complexity can be taught to students using the notion of a formal system. The motivation behind this research lies in the following two observations: (1) it becomes increasingly important to learn complexity science at school (Boy, 2013) (2) one important issue in complexity science is concerned with identifying rules that can result in complex phenomena in the world (Phelan, 2001), which can be explained using formal systems. In fact, there are various examples that can be used to teach complexity science which share a common denominator (i.e., a formal system in our context) although they look different on the surface. For example, it has been known that there exists a phase transition in the space of computational problem instances (Hayes, 1997) and this complex phenomenon is closely related to the hardness of computation for some problems which can be explained using a formal system. As a different example, the existence of computationally unsolvable problems such as the Halting problem can be easily explained using a formal system. It is also possible to teach Gödel's incompleteness theorem which is a special example of complex phenomena using a formal system.

References

Alagar, V.S., Periyasamy, K. (2011). *Specification of Software Systems*. (Texts in Computer Science), Springer-Verlag, London.

Bittner, T., Frank, A.U. (1997). An introduction to the application of formal theories to GIS. In: *Proc. Angewandte Geographische Information Sverarbeitung, IX (AGIT)*. Salzburg, Austria.

Boy, G., (2013). From STEM to STEAM: toward a human-centered education, creativity & learning thinking. In: *Proceedings of the European Conference on Cognitive Ergonomics*.

Dagienė, V., Jevsikova, T. (2012). Reasoning on the content of informatics education for beginners. *Socialiniai Mokslai*, 4(78), 84-90.

Day, T. (2012). Computability, Gödel's incompleteness theorem, and an inherent limit on the predictability of evolution. *Journal of the Royal Society Interface*.

Fitting, M. (2007). *Incompleteness in the Land of Sets*. College Publications.

Franzén, T., (2005). *Gödel's Theorem: An Incomplete Guide to Its Use and Abuse*. A K Peters/CRC Press.

Gensler, H.J., (1984). *Gödel's Theorem Simplified*. University Press of America, Inc.

Gibson, P., Méry, D. (1998). Teaching formal methods: lessons to learn. In: *IW-FM'98 Proceedings of the 2nd Irish conference on Formal Methods*.

Goldreich, O. (2012). Invitation to complexity theory. *XRDS: Crossroads: The ACM Magazine for Students*, 18(3), 18-22.

Hayes, B., (1997). Can't get no satisfaction. *American Scientist*, 85(2).

Hofstadter, D.R. (1999). *Gödel, Escher, Bach: An Eternal Golden Braid*, Basic Books.

Linz, P. (2006). *An Introduction to Formal Languages and Automata*, 4th edition. Jones and Bartlett Publishers.

Lobo, A.F., (2007). *Teaching Problem Reduction: NP-Completeness via Sudoku*, CIESC 2007.

Lowther, J., Shene, C. (2004). Toward an intuitive and interesting theory course: the first step of a road map. *Journal of Computing Sciences in Colleges*, 20(1).

Mackenzie, D., (1995). The automation of proof: a historical and sociological exploration. *IEEE Annals of the History of Computing*, 17(3).

Majherová, J. (2007). Virtual plants in high school informatics - L-systems. In: *Conference ICL 2007*.

Nelson, R.J. (1967). *Introduction to Automata*. John Wiley & Sons, Inc.

Phelan, S.E., (2001). What is complexity science, really? *Emergence*, 3(1).

Sipser, M., (1992). The history and status of the P versus NP question. *Proceedings of the 24th ACM Symposium on the Theory of Computing*. 603--618.

Sipser, M., (2006). *Introduction to the Theory of Computation*, 2nd edition. Thomson Course Technology.

Smullyan, R.M., (1982). *The Lady or the Tiger? & Other Logic Puzzles*. Dover Publications, Inc.

Tautu, P. (1976). Formal languages as models for biological growth. In: Berger, J. *et al.* (Eds.), *Mathematical Models in Medicine* (Lecture Notes in Biomathematics, vol. 11). Springer-Verlag, Berlin,127-134.

S. Yang received her bachelor's degree in computer science education from Korea University in Seoul, Korea. Her research interests include computer science education and the Semantic Web.

S. Park is a professor in the department of computer science education at Korea University in Seoul, Korea. He received both master's and doctoral degrees from the University of Southern California. His research interests include computer science education, the Semantic Web, and adaptive hypermedia.

Kelių informatikos sąvokų mokymas taikant formaliąsias sistemas

Sojung YANG, Seongbin PARK

Informatikos mokyme yra daug svarbių temų ir dauguma sutaria, kad algoritmai ir programavimas yra pagrindinės temos, kurios turi būti nagrinėjamos informatikos kurse. Straipsnyje pateikiami siūlymai, kaip šios temos galėtų būti lengviau mokomos pritaikius formaliąsias sistemas, sudarytas iš aksiomų ir išvedimo taisyklių, kuriomis remiantis įrodomos teoremos. Remiantis (Dagienė ir Jevsikova, 2012) argumentais, svarbias informatikos temas galima pateikti naudojant dėlionių pavyzdžius ir studentai gali neturėti jokių išankstinių žinių. Straipsnyje pristatyta medžiaga buvo dėstoma hiperteksto ir skaičiavimo kurso paskaitų metu nuo 2008 metų rudens semestro ir gali būti lengvai pritaikoma mokant pradedančiuosius studentus, kurie neturi techninių pagrindų.