E-Learning of Foundation of Computer Science

Yukiyoshi Kameyama[1] and Masahiko Sato[2]

[1] Department of Computer Science, University of Tsukuba
kam@cs.tsukuba.ac.jp
[2] Graduate School of Informatics, Kyoto University
masahiko@kuis.kyoto-u.ac.jp

Abstract. We report our teaching experience of undergraduate courses on logic and computation. The topics covered are the syntax and the semantics of basic logical and computational calculi. We put our emphasis on the treatment of formal systems and formal reasoning which is notoriously difficult for most students. To overcome this difficulty, we developed an e-learning system called CAL and have been using it for the last several of years. The CAL system turned out not only useful, but also essential to achieve the goals for our courses.
In this paper we give an overview of the CAL system, describe the design principle of our courses, and discuss our teaching experiences.
Keywords: E-Learning, Computational Logic, Formal System, Proof-Checker, Game, Syntax and Formal Semantics, Programming Languages, Propositional Logic, Simply Typed Lambda Calculus.

1 Introduction

For the last several years, we have been teaching undergraduate courses which cover the foundational topics for logic and computation. The course title is "Computational Logic" for the first author at University of Tsukuba, and "Computation and Logic" for the second author at Kyoto University.[1] The specific topics covered are the syntax, the semantics and formal reasoning. The main goal of these courses is to help students think about the computer programs as formal objects so that they can represent and reason about programs rigorously. We believe that understanding formal systems and formal reasoning is a necessary first step for all students in Computer Science Department, since computers are only able to treat formal objects (apart from the meaning), and computer programming is therefore a way of formal reasoning. If a student does not understand formal systems, then he can never write good programs.

Our goal involves a challenging issue in education, since, most students are to some extent familiar with *informal* mathematical reasoning, while *formal* systems and *formal* reasoning are for them a completely new language as if it

[1] The two courses have essentially the same concept and share much materials, but differ in concrete subjects. This is natural because each course is in a different curriculum at each department and also because the course hours are different. In this paper, we mainly focus on the course taught at University of Tsukuba.

were an unknown foreign language. In order to get accustomed with a foreign language, students must try as many exercises as possible (reading, writing, speaking, and listening). Similarly, in order to understand formal reasoning for the first time, students must work out a sufficient number of exercises until they understand how to reason formally. Answering one question about a formal system takes only a few minutes for some students while other students need more than several days or even weeks to solve a single question. Even worse, checking the correctness of answers takes a long time even for experienced teachers. Due to lack of human resources, traditional courses on formal logic tend to be difficult, uninteresting, or even inaccessible for most students.

A solution for this problem is to use a proof-checker. Given a question about formal reasoning, a student inputs his answer to the proof-checker, which checks if the input is correct or wrong. In the latter case, the proof-checker points out an error in the proof, and then the student can improve his wrong proof based on the error message.

However, things are not so simple. This scenario works only if the software is well designed and developed to be used for this purpose. Existing softwares for proof-checking (and even our initial version of the CAL system) do not suffice all the needs for our purpose. Let us point out some of the problems.

Firstly, the software must have a good user interface. Since students must sometimes input a large symbolic expression as a proof object, a good editorial facility is essential for the success of this approach, while many existing proof-checkers have very poor editors, or too specialized editors which students do not know.

Secondly, the software should not be bound to any specific logic or calculus. Rather, it should allow the defining facility of various formal system, and the proof checker must be generic so that teachers can define (or modify) logical systems and computational calculi from time to time. Thirdly, the software must be designed so that it gives good advices to the student when he inputs a wrong answer. Error messages from the software are advices to students by which they can improve wrong answers, and thus important for the purpose of education. The advices should be not too kind (for instance, automatic theorem proving facility should not be provided), and not too unkind (at the minimum, the system should pinpoint which part of the user's input is wrong).

We have solved these issues by developing and using a software called CAL (computation and logic). The CAL system acts as a proof-checker which checks if a given proof is correct against a formal system. It is fully automatic, and does not need teachers' help so that students can try to solve questions whenever they want, even in the midnight from their home. The CAL system has been carefully designed to fulfill the requirements we mentioned above. The initial version of the CAL system was developed at Kyoto University in 1998 by the second author's leadership with the first author and Izumi Takeuti. In 2001, the second author has totally rewritten the software to obtain the current version, which has been used by the authors as courseware since then. It should be noted that the current version of the CAL system is surprisingly stable, and thus gives

a solid foundation for e-learning. Our teaching experience in the past years has shown several positive signs to use the CAL system in our courses.

The paper is organized as follows. In Section 2, we give a short overview of the CAL system and also explain the underlying concepts. In Section 3, we explain the design principles of our course, and give some concrete examples. In Section 4, we report our teaching experience in the last several years. In Section 5, we make concluding remarks.

2 Overview of the CAL system

In this section, we briefly overview the CAL system without getting into technical details. More information on the CAL system can be found in another paper [8].

2.1 Logical Perspective

Abstractly, the CAL system implements two functions: to *define* a formal system in a generic logical framework (called Natural Framework), and to *check* if a given proof is correct with respect to the definition of the formal system.

The key concept of the Natural Framework is a *derivation game*, which we think is parallel to an ordinary game such as Shogi (Japanese Chess), Chess and Card Games. The important properties of ordinary games are that they have rigid rules to which players must follow, and that there is a rigid criterion as to who wins/looses the game. Formal reasoning with logical systems and computational calculi can be thought as a game too, although it is played by a single person rather than multiple players.

A derivation game in CAL corresponds to a formal system. The salient aspect of CAL's derivation games is that one can define data structures under binding structures as derivation games. The rules of a derivation games are formal derivation rules such as the introduction rule of implication (in the propositional logic game) and the lambda-introduction rule (in the simply typed lambda calculus game). The CAL proof-checker checks if a user's input strictly follows the rules or not. In this sense, the derivation game enjoys the same two properties as above: each derivation game has its own rules players must follow, and also there is rigid criteria as to whether the player wins the game or not.

The formal systems that have been so far defined as derivation games in CAL range from various logical systems and computational calculi to inductively defined datatypes and operational semantics. We could also define an ordinary game as an instance of derivation games. In Section 3, we show a few concrete derivation games we use in our courses.

2.2 Inside the CAL System

As a concrete software, the CAL system consists of three components: User interface, parser, and checker.

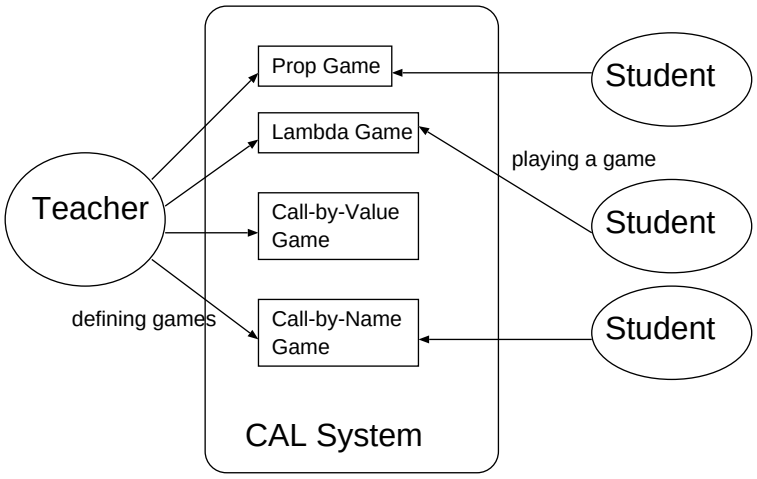

Fig. 1. Defining and Playing with Derivation Games

The parser and the checker do the obvious things: the former parses an input string and converts it to a CAL expression, then the latter checks it. The key feature is that it is generic in the sense that, given a specification of a formal system as a derivation game, it behaves as the proof checker of that specified formal system. This is important since teachers often want to switch from one formal system (logic or programming language) to another in a single course. Technically, we have developed a new theory of expressions, and a new meta-logic called Natural Framework by which one can define an arbitrary derivation game. It should be noted that not only the checker, but the parser is generic in the syntax, since the surface syntax varies depending on a derivation game.

The user interface of the CAL system is implemented on top of the Emacs editor, and a user can communicate with the CAL system through a buffer of the emacs editor. In fact the CAL system is simply a collection of Emacs functions. A user (student) invokes the CAL system by simply calling the top-level function which is similar to the read-eval-print loop of Lisp, then he can interact with the CAL system until he quits the loop. During this interaction, the user can get questions in each game, and can input answers for any questions he likes to answer then. The server checks whether the user's answer is a correct one for the corresponding question in the current game, and shows the results. When the user inputs a wrong answer, the system not only says it is wrong, but indicates the wrong part of the user's answer. The important design principle of the CAL system is that, the error message should not be too kind, nor indicates multiple errors at a time. Our goal is to help students understand the syntax and the semantics of formal systems, and to do so, they must think about what was wrong at the rejected answer. This is in contrast with most compilers' approaches in which they always try to show verbose error messages, by which programmers can often correct their errors without thinking about the reason.

```
CAL version 5.22 of Sun Dec  8 16:44:54 2002

CAL:Prop <1> play[Prop]
Let's play Prop!
CAL:Prop <2> Q[50]
Derive the following judgment in the Prop game.

A, B, C ⊢ ((A∨B)⊃C) ⊃ ((A⊃C)∧(B⊃C))

CAL:Prop <3>
```

Fig. 2. Playing the Prop game with CAL

2.3 Monitoring

The CAL system is not implemented as a centralized server in the server-client model. Each user has a copy of the executable image in his (virtual) memory space, and interacts with the copy independently with others. This implies that failure in one user's CAL system (possibly due to a bug in the system) does not affect other users.

A problem in this model is that how we collect information (each student's achievement). We chose a simple solution for this, namely, when each user quits the CAL system, the system writes the results (such as the student's name and the question numbers he correctly answered) into a file in the CAL system's directory with some simple cryptographic method. This directory is periodically scanned and statistics is calculated, which is sent to a teacher by e-mails so that he can grasp the students' progress.

3 Course Design

After having decided to use the CAL system, we have set several design principles for our course. In this section, we explain these principles one by one.

3.1 Informal first, formal second, meta properties last

That our major goal is to cultivate students' ability of formal reasoning does not imply we totally ignore informal practice of mathematical reasoning. In fact, we take an opposite way; for all formal systems we teach in the courses, we spend several hours for in-class lectures before the students start playing the corresponding games with CAL. In other words, students do not have to play an adventure game.[2]

[2] In an adventure game, the player must find rules and even objectives of the game by themselves.

Although the meta properties of formal systems such as normalization and confluence are important to understand the formal systems deeply, we do not teach them until (most) students finish to solve the questions in the corresponding game. This is mainly because, some meta properties are so useful in reducing the search space in the proof search, and therefore even if they do not understand the formal system, they can make use of the meta properties to construct proofs quickly, which might prohibit them to think about the formal reasoning deeply. We believe that students can find the real meaning of meta-properties only after they do trial-and-errors with the formal system for sufficiently many times.

3.2 Not too many formal systems

A natural consequence of our observation that learning formal reasoning is similar to learning a foreign language is that, we should not offer too many different languages (formal systems) to students. Rather, they should concentrate on one formal until their understanding reach at a certain level. Hence, we define a limited number of derivation games, but for each game, we give a great number of questions which students must solve (play with). In the course at University of Tsukuba, we have deliberately chosen only four games (see below), which are we think the minimum core when teaching basic logical and computational systems.

3.3 Formalize as much as possible (not only syntax, but also semantics)

The central slogan of the course is that we can formalize informal practice of reasoning in terms of derivation game (aka inductive definition with binding structures). To reinforce this message, we try to formalize as many concepts which appear in each formal system as possible. For instance, we not only formalize the syntax of a computational system, but only we give the semantics as a derivation game (formal semantics). By seeing that both the static side (syntax) and the dynamic side (operational semantics) can be equally formalized, some students might predict the expressive power of derivation games, or induction definitions. This principle need to be compromised since we do not want to define too many formal systems (the second principle), though.

3.4 Concrete Derivation Games

Based on these principles, we have deliberately chosen only four games as follows:[3]

- Prop game for propositional logic.
- Lambda game for simply typed lambda calculus.

[3] The second author defines six games which include Heyting arithmetic and dependently typed lambda calculus.

– CBV game for call-by-value evaluation of simply typed lambda calculus.
– CBN game for call-by-name evaluation of simply typed lambda calculus.

The first one is a logical system, and the rest are about a computational system, among which the Lambda game is about the syntax, and the last two are about the semantics.

The Prop game corresponds to the natural-deduction style propositional logic. We have chosen this game as the first game, since it is the most fundamental logic and students have sufficient knowledge about it. We take a variant of the standard (classical) logic in that the double-negation elimination rule (or the law of excluded middle) is missing, hence it is intuitionistic. This choice is motivated by the Curry-Howard isomorphism, which connects intuitionistic logic to typed lambda calculus, and is one of the central principles in the modern study on functional programming languages. The difference in fact does not matter since the questions we give to students are all provable in both logic, i.e. without using the double-negation elimination rule, and we never ask them to show unprovability of a formula.[4] Readers can find a sample session with the Prop game (a question about proving some formula, and a student's answer) in Figure 5 in the appendix.

The Lambda game corresponds to the syntax of the terms in the simply typed lambda calculus augmented with the product, coproduct (sum), and empty types. The introduction of these types are again motivated by the Curry-Howard isomorphism.

The CBV and CBN games give two operational semantics of the simply typed lambda calculus. We think that teaching formal semantics to students is essentially important; in daily programming, they are not conscious with the evaluation order, and do not care about the meaning of programs. In these games we give so called big-step operational semantics, namely, we define a binary relation $a \downarrow v$ with the meaning that the term a evaluates to the value v. A few key rules of the CBV game is illustrated in Figure 3. To read the rules, we note that the CAL expression $\lambda(x : A)b[x]$ means $\lambda x^A.b[x]$ in the standard notation (where A is the type of the bound variable x), and $b[c]$ is an application of a higher-order expression b to an expression c. Note that the rule is expressed using a higher-order encoding, which is similar to higher-order abstract syntax (such as the one used in the Twelf system), but is (we believe) simpler and more fundamental. By virtue of this encoding, the binding structures of the (informal) rules are precisely represented by the CAL rules.

$$\frac{}{x \downarrow x}\ \text{var} \qquad \frac{}{\lambda(x : A)b[x] \downarrow \lambda(x : A)b[x]}\ \lambda \qquad \frac{f \downarrow \lambda(x : A)b[x] \quad a \downarrow v \quad b[v] \downarrow w}{f(a) \downarrow w}\ \text{apply}$$

Fig. 3. Big step call-by-value semantics

[4] In fact, it is impossible to show unprovability inside the CAL system.

The call-by-name semantics is defined in a similar way. Since the difference

$$\frac{}{x \downarrow x}\ \text{var} \qquad \frac{}{\lambda(x:A)b[x] \downarrow \lambda(x:A)b[x]}\ \lambda \qquad \frac{f \downarrow \lambda(x:A)b[x] \quad b[a] \downarrow c}{f(a) \downarrow c}\ \text{apply}$$

Fig. 4. Big step call-by-name semantics

of the two semantics is small, students, at a first sight, may not be able to understand the difference of the two systems, or even recognize the difference. Only after having solved a sufficient number of questions about these two games, students are able to recognize the difference of the two semantics in a precise way.

4 Teaching Experience

We have been teaching the undergraduate course using the CAL system for five years at University of Tsukuba, and we believe that it has been successful. Let us briefly state some of our experiences.

- To our surprise, students very much like solving questions given by the CAL system. Before we started to use the CAL system, we have never encountered this situation, and most students seemed to dislike formal reasoning. In fact, students seemed to have spent much longer time than expected in solving questions of the CAL system. By collecting students' impression, we have found that they were enjoying the derivation games in CAL as if it were a new kind of computer games, even if the derivation games are played by a single person.
- Many students preferred to play the derivation games at the computer room in the university, rather than to play from their homes or other places. While they could enjoy solving the questions from anywhere in the world (provided they can connect to internet), they liked to solve questions with other students. One obvious reason for this is that they wanted to exchange information among students. However, a more important, and surprising reason is that, they are competing with each other. The messaging system of the CAL system shows the list of top ten players who have solved most questions then, and the list is updated daily. The list stimulated many students, and they tried to solve questions as fast as possible, earlier than other students. In short, modest competition stimulates students.
- Students sometimes found useful meta theorems by themselves through the interaction with the CAL system. While they could not prove the meta theorems (no one have found the meta-theoretic induction which works on the proof objects.), they found (or conjectured) a few meta theorems.
  In the Lambda game, most students found that the derivation that gives a typing to a given term are trivially constructed from the given term. In the

Prop game, a few clever students found a more interesting meta theorem about the shape of the formal proof, which is known as the subformula property.[5] This is in fact a substantial theorem in logic and typed lambda calculus.

We think that these are positive signs for the use of the CAL system especially when teaching the foundational area of computer science.

5 Conclusion

In this paper we have reported our experience on teaching the courses on foundation of computer science. Formal reasoning is no doubt a suitable subject to teach with an e-learning system, but the results seem to indicate more than that. We have found that many positive signs to use the CAL system, which strongly encourage further efforts.

Yet, there remains room for improvement. One of the most important problem is the error messages. Since the CAL system is generic, i.e. is not bound to any specific system, its error messages are uniformly generated and sometimes not very useful for users. Moreover, it signals only one error at a time so that students must improve their derivations one by one, typing similar derivations (which might be several ten lines) many times. Also we should widen the scope of the CAL system so that one can formalize more different systems as derivation games.

The second author has already started to refine the underlying meta logic and plans to revise the system in near future.

Acknowledgments:

The first author would like to thank Tetsuo Ida at University of Tsukuba for encouragements.

[5] Natural deduction style logic usually enjoys the subformula property: "all formulas appearing in a normal proof are subformulae of the conclusion or one of the assumptions in the proof".

References

1. Barendregt, H. P., *The Lambda Calculus, Its Syntax and Semantics*, North-Holland, 1981.
2. Barwise, J. and J. Etchemendy, *Tarski's World*, CSLI Lecture Notes, No. 25, CSLI Publications, Cambridge University Press, 1994.
3. de Bruijn, D. G., Lambda Calculus Notation with Nameless Dummies, a Tool for Automatic Formula Manipulation, with Application to the Church-Rosser Theorem, *Indag. Math.* 34, pp. 381–392, 1972.
4. Harper, R., F. Honsell, and G. Plotkin, A Framework for Defining Logics, *Journal of the Association for Computing Machinery*, Vol. 40, No. 1, pp. 143–184, 1993.
5. Sato, M., and M. Hagiya, Hyperlisp, in de Bakker, van Vliet eds., *Algorithmic Languages*, North-Holland, pp. 251–269, 1981.
6. Sato, M., Theory of Symbolic Expressions, II, *Publ. of Res. Inst. for Math. Sci.*, Kyoto Univ., **21**, pp. 455–540, 1985.
7. Sato, M., An Abstraction Mechanism for Symbolic Expressions, in V. Lifschitz ed., *Artificial Intelligence and Mathematical Theory of Computation (Papers in Honor of John McCarthy)*, Academic Press, pp. 381–391, 1991.
8. Sato, M., Y. Kameyama, I. Takeuti, Masahiko Sato, Yukiyoshi Kameyama, Takeuti Izumi, "CAL: A Computer Assisted Learning system for Computation and Logic", EUROCAST2001, *Lecture Notes in Computer Science* 2178 (eds. R. Moreno-Diaz, B. Buchberger, J.-L. Freire), pp. 509–524, 2001.
9. Sato, M., T. Sakurai and Y. Kameyama, A Simply Typed Context Calculus with First-Class Environments, Proc. Fifth International Symposium on Functional and Logic Programming (FLOPS), *Lecture Notes in Computer Science* 2024, pp. 359–374, 2001.

Appendix.

```
Q[50]
Derive the following judgment in the Prop game.

A, B, C ⊢ ((A∨B)⊃C) ⊃ ((A⊃C)∧(B⊃C))

CAL:Prop <4>
A[50][
A,B,C ⊢ ((A∨B)⊃C) ⊃ ((A⊃C)∧(B⊃C)) in Prop since
((A∨B)⊃C) ⊃ ((A⊃C)∧(B⊃C)) by ⊃I {
  (X1:: (A∨B)C) [
     (A⊃C)∧(B⊃C) by ∧I {
        A⊃C by ⊃I {
          (X2::A)[
            C by E {
              (A∨B)C by assume {X1};
              A∨B by ∨IL {
                A by assume {X2}
              }
            }
          ]
        };
        B⊃C by ⊃I {
          (X3::B)[
            C by E {
              (A∨B)C by assume {X1};
              A∨B by ∨IR {
                B by assume {X3}
              }
            }
          ]
        }
     }
  ]
}
]
Correct Answer!
CAL:Prop <5>
```

Fig. 5. A sample session in the Prop game.