Undergraduate Logic Teaching in Computing: Why, What, How?

Roger Villemaire

May 27, 2022

Abstract

Logic lies at the heart of computing, but its presence is somewhat dimming. This is particularly striking at the undergraduate level, where far too little logic training is done. This paper examines this regrettable state of affairs, considering what is actually taught, what should be taught, and how logic must be taught at computing undergraduates.

1 Introduction

Logic is with Mathematics and Engineering a founding discipline of Computing. While many methods and concepts from logic are to be found in computing fields, such as, compiler design, artificial intelligence, programming languages, computer organization and architecture, and theoretical computer science, the significance of logic in computing is somewhat fading away. However, our field has continued to progress, consistently producing new results and methods. Furthermore, while the theoretical computer science logic community is still thriving, the computing logic community has grown tremendously, shifting the center of gravity toward applications.

Unfortunately, there is too little awareness of these striking logic-based applications in the general computing community. Furthermore, the distinct lack of logic teaching at the undergraduate level in computing programs moreover strongly limits opportunities to make these advances more broadly known.

The objective of this paper is to summarize the current state of logic teaching at the undergraduate level, reflect on the central contributions of logic, and propose a tentative undergraduate course in logic for computing programs.

Accordingly, Section 2 will recap current undergraduate logic teaching, taking the program at the author's institution as a representative example, and Section 3 will highlight the mostly professional orientation of current undergraduate computing degrees. Section 4 will then summarize what current logic teaching is missing, and Section 5 will present a tentative course content. Finally, Section 6 concludes the paper.

2 Logic at the undergraduate level

There is quite a wide range of undergraduate degrees in computing. It hence comes as no surprise that the Association for Computing Machinery (ACM), a major association in the field, publishes curricula recommendations for Computer Engineering, Computer Science, Information Systems, Information Technology, and Software Engineering[1]. However, apart from some computer science programs, there is generally little logic teaching. In order to set the stage for the following discussion on teaching of logic in undergraduate computing degrees, this section presents the logic content in a representative computing undergraduate degree that of the bachelor's in computer science and software engineering (BIGL, by its French acronym) at my own institution [2].

The BIGL leads to a bachelor's degree in applied sciences (B.Sc.A.) and is not an engineering degree. Graduates are therefore not automatically eligible to join the Quebec Association of Professional Engineers [3] and are hence forbidden by law to claim to be software engineers. They can however present themselves professionally as computer scientists working in software engineering. One should note that in the industry, engineers and computer scientists work side by side and can fulfill exactly the same tasks. For computing, there is no provision in the law as to activities reserved to professional engineer as for, for instance, in civil engineering.

Nevertheless, the BIGL has a strong emphasis on software engineering, and differs mostly from similar degrees in engineering schools by the lack of science courses such as multivariable calculus, differential equations, physics, and chemistry. Prospective students must however master univariable calculus and linear algebra as taught in pre-university colleges, a usual requirement for computer science degrees in Quebec.

With such an emphasis on software engineering and professional skills, it will come as no surprise that logic content in our degree is scarce. Moreover, the BIGL went through its last major revision in 2018 following the *Curriculum Guidelines for Undergraduate Degree Programs in Computer Science* from the ACM-IEEE Joint Task Force on Computing Curricula 2013 [5]. While these guidelines offer quite some flexibility, the program should nevertheless also be similar to many other computer science degrees in North America.

As to the teaching of logic in the BIGL, this is mostly done in the *Mathematics for Computer Science*[4] course. In that class students learn propositional calculus and first-order logic. They use truth-table to show equivalence of propositional formulas and devise and verifying first-order properties on specific structures. First-order semantics is however informal. Proof methods, such as induction, are also informally introduced and applied to specific examples. This course also introduces basic notions about binary relations, which play a significant role in computer science logic and, accordingly, will be an entire part

[1] `https://www.acm.org/education/curricula-recommendations`
[2] `https://etudier.uqam.ca/programme?code=7617`
[3] `https://www2.oiq.qc.ca`
[4] INF1132 Mathématiques pour l'informatique

of the course proposed in Section 5.

Logic-based concepts are also present in some other compulsory courses. One can mention here entity-association and relational models, or relational algebra and SQL, in the *Data Bases*[5] class. As we will later see in Section 5, I also consider in this category the teaching of *Unified Modeling Language (UML)*[6] and *Object Constraint Language (OCL)*[7] in the *Software Engineering: analysis and modeling*[8] class.

Rules, Horn clauses, resolution and some constraint processing appear in the *Functional and Logic Programming*[9] class. Finally, one can also mention the compulsory *Computer Architecture*[10] class that presents logic circuit and the *Algorithmic*[11] class which very briefly introduces Turing machines, the halting problem, P=NP and NP-completeness.

A single major optional course introduces some logic that of *Artificial Intelligence (AI)*[12]. This course is worth mentioning since AI shares a close relationship with logic. Furthermore, this class being immensely popular, it also largely shapes our students' views on logic. In this class students learn how to represent a problem in first-order logic, transform a formula into *Conjunctive Normal Form (CNF)*, and proofs by unification and resolution. While not necessary presented as logic-based, this course also presents constraint processing, with constraint propagation and backtracking search for a solution.

The Curriculum Guidelines for Undergraduate Degree Programs in Computer Science of the Joint Task Force on Computing Curricula 2013 (CS2013) [5] divides topics into Core-Tier1 that should be covered by all Computer Science programs, Core-Tier-2 that should be at least mostly, if not completely, covered, and Elective. It also clearly explains that Core-Tier1 and Core-Tier2 are not sufficient, and that any Computer Science program should cover many of the Elective topics in significant depth.

It is significant that CS2013 only mentions Propositional logic and Predicate logic as Core-Tier1 topics. As Core-Tier2 topics, it further mentions digital Logic and propositional and predicate logic with resolution and theorem proving in the setting of Knowledge Representation and Reasoning, with resolution limited to propositional logic. Finally, Elective topics contain Description logics, Logic-based knowledge representations for Natural Language Processing, Inductive logic programming (ILP), Logic Programming, formal specification (Z, first-order logic) in requirements Engineering, and assertion and analysis languages (OCL, *Java Modeling Language (JML)*, model-checking) in formal software modeling and analysis.

[5] INF3080 Bases de données
[6] `https://www.omg.org/spec/UML`
[7] `https://www.omg.org/spec/OCL`
[8] INF5151 Génie logiciel: analyse et modélisation
[9] INF6120 Programmation fonctionnelle et logique
[10] INF4170 Architecture des ordinateurs
[11] INF5130 Algorithmique
[12] INF4230 Intelligence artificielle

3 Professional purpose and career objectives at the undergraduate level

Undergraduate degrees in computing, particularly that at my own university, tend to be very professionally oriented. This is quite natural since computing shapes modern society and computing professionals are in high demand. Accordingly, essentially all graduates go to industry where many interesting career opportunities await them. Graduate students are also mostly international, a fact clearly not limited to my own university [8]. In this setting, it is therefore neither possible, nor appropriate, to teach at the undergraduate level in computing mainly in preparation to graduate studies. This sets computing apart from most scientific fields and should impacts on how logic is presented to a computing audience.

Teaching of logic must therefore adapt to this state of things and allows undergraduates to gain skills that are meaningful in a professional setting. Moreover, even if very few undergraduates directly continue to the graduate level, there is also the development of less conventional paths with former graduates coming back later in their career to graduate school. In all cases, students could better leverage logic-based methods both in industry and academia if they had appropriate initial training at the undergraduate level.

4 What is then missing in the actual covering of logic in undergraduate computing degrees?

As we saw in Section 2, little logic is actually covered in a typical undergraduate computing degree. I claim that more logic teaching is essential, and this section will present the fundamental aspects that should be considered in devising an undergraduate logic class in computing.

4.1 What is logic?

In its most fundamental aspect, logic is the formal representation of thoughts allowing their formal processing. It is a formal representation in the sense that it introduces a language constructed in a systematic way that clearly distinguishes what is, and what is not, a formula. It represents thoughts in the sense that one can convey a point of view, a conception of some situation, by some formula. Formal processing first allows a formula to be processed to yield an unequivocal, unambiguous meaning that allows to settle conflicting interpretations. Secondly, formal processing also allows to infer new, implicit, knowledge and compare viewpoints in terms of their consequences.

As I already argued in the context of graduate courses in logic [9], logic is all about modeling. This is obviously also the case at the undergraduate level. However, at that level, it is particularly important to step back, thoroughly presenting what modeling is all about.

4.2 Modeling

A model is a representation of *some aspect* of reality. A model is useful in that it allows to reason about that specific aspect. Modeling is used everywhere in computing, since computational systems tend to be complex, and expensive to design, build, and debug. A model is furthermore useful in many ways. First as a communication tool to convey some aspect of a design, and this even before any code artifact is developed. But also, to help to analyze, and reason about this aspect.

It is then of paramount importance to stress the fact that, in order to be useful, a model will represent some aspect of interest, put clearly not everything of interest! Indeed, in order to reason about an aspect of a system, one does not need a description of the complete system. Not only will too many details clutter the model and slow down its development, but it will also make reasoning more difficult.

There is also no such thing as the "right" model. Different models can offer complementary views. The right question is to ask whether a model is useful, for some objective. But here also, there can be many different useful models, and this should always be emphasized when students develop their own models.

What distinguished logic from other modeling methods is that it offers a clearly defined modeling formalism, with a well-defined semantics and inference methods. A clear semantics allows to settle disagreements on the meaning of an expression. Well defined inference methods allow computational processing in order to help reasoning.

But logic does not exist in a vacuum. It is therefore of paramount importance to establish links with other concerns and subfields of computing.

4.3 Logic is about applications

It is important to stress the interactions of logic with other computing fields, such as, for instance, networks/communications, software engineering, databases, hardware/computer systems organization, data science, and artificial intelligence.

An undergraduate course in logic should present meaningful applications that shows that logic is not an isolated or limited subject but an integral part of computing with impact throughout the field. Plainly, this offers a takeaway for the entire undergraduate course in logic. Indeed, what will students remember after finishing such a course? It is of paramount importance that students acquire some skills that they can show and use. Applications also makes the material more relevant and easier to relate to.

Within a curriculum, it is also central to establish links with other computer subjects and courses. For instance, among the first applications of logic in computer science is the formal verification of communications protocols. In a typical computing undergraduate degree, students are usually introduced to communication protocols in a dedicated network course. Communications protocols allow numerous behavior that are difficult to encompass, and students

will usually realize that assuring that a protocol is correct, can be somewhat tricky. A logic class can bring these questions to the fore and show some simple, but representative, examples of protocol verification.

Similar verification questions arise with hardware and computer systems. In a more hardware-oriented degree one could look, for instance, into equivalence of propositional formulas or present some cases of finite state machines verification.

Software engineering, where effective communication and analysis of software systems is a central concern, also offers many opportunities. For instance, the class diagrams of the *Unified Modeling Language (UML)*, a fundamental standard of the *Object Management Group (OMG)*[13] is an instance of (binary) relational models that should be covered in an undergraduate logic class as suggested in Section 5.

In relation to Databases, Codd's relational model offers a twofold opportunity. First, that of emphasizing that relations, i.e., the DB tables, are not simply a collection of tuples, but rather regroup under a meaningful name related attributes. Secondly, first-order logic with its logical operators is the foundation of relational algebra, and *Structured Query Language (SQL)* simply because these operators are basic building blocks that one can combine in order to express more complex notions, from simpler.

In Data Science one should not miss the opportunity to speak of the *World Wide Web Consortium (W3C)'s*[14] *Resource Description Framework (RDF)* where the linking structure of subject-predicate-object triples represents information in terms of binary relations. The *Web Ontology Language (OWL)*, or more precisely *Description Logic (DL)*, being a fragment of first-order logic could appear naturally in a logic course, as done in Section 5.

As for *Artificial Intelligence (AI)*, already many of the issues touched upon in this section are of a knowledge representation nature, so one can simply extend toward some more AI typical problems, such as games or planning. One should however also mention *Constraint Satisfaction Problems (CSP)*, with its relation to SAT, as will be proposed in Section 5.

Section 5 will develop a tentative course along these lines. But before, let us look at some common objections that hinder logic teaching in undergraduate computing degrees.

4.4 Objections to logic

The logic community cannot do without taking note of an important skepticism toward logic in computing, when not outright hostility. Logic education at the undergraduate computing level is therefore not only a question of educating our student but also of correcting many misconceptions among our colleagues!

In many minds logic is still strongly associate to theory. As teaching of theory has declined, since modern computing degrees tend to emphasize professional skills, in this mindset logic is considered of minor interest to computing which is

[13] `https://www.omg.org/`
[14] `https://www.w3.org/`

all about building systems. Indeed, logic plays a key role in theoretical computer science, with important results and a vibrant community. The objective of logic teaching at the undergraduate level is obviously not to downplay this role, nor to object to theory-related logic courses that exists in some institution. However, there is more to logic than simply theoretical results, and many of the applied advancements of the field should find their place in undergraduate teaching. So, it must be clearly stated that while logic plays a key role in theory, logic is not only about theory and that it is relevant for the computing professions.

Another objection that arises often, particularly when working with colleagues in more applied fields, such as networking or robotics, is that logic is far too complex, and that it is way too difficult for 'normal people' to understand some formal statement let alone to write such a statement.

While this objection is natural, and that one must recognize that writing, reading, and understanding formal logic statements can be challenging, there are two important pitfalls at play. First, improper, or crude tools makes the task even more difficult. Computer scientists expect, and deserve, a nicer experience, more in line with modern software development environments. One must be attentive to this aspect. Logic-based tools did indeed improve in the last decades, even if there is still some progress to be made. One should therefore carefully choose the tools used in a undergraduate logic class. Secondly, practitioners tend not to realize the similarity between writing code and logic expressions. Yes, logic expressions are declarative and therefore not usually a sequence of instructions. However, they represent conditions, and as such can be decomposed in simpler expressions that will allow to manage devising complex statements. In fact, the two aspects are strongly related. If logic-based tools commonly presented a development environment allowing parameter passing and decomposition into simpler expressions, it would be easier to leverage usual software engineering practice and allow a much smoother transition from writing code to writing specifications. This barrier should not be underestimated, as a robotician already told me – "everything is always simple for a logician" – making him suspicious of the usability of any tool that could emerge from our community.

Another often heard objection is that the fundamental problems of logic are simply too complex. For instance, SAT is NP-complete, we therefore do not expect a polynomial algorithm. Propositional logic being at the foundation of most logics used in applications, this pretty much knockout the whole field! However, this also come from a misunderstanding of what is at play here. At the very basic computational level, while SAT is NP-complete, there has been tremendous improvements in algorithmic design with the engineering of impressively efficient SAT-solvers. While this feat has been acknowledged by the computing community, it has not completely undermined this line of argument. Another not sufficiently recognized aspect is that NP-completeness can also be interpreted as a sign that propositional logic is concise: a short formula can express a complex constraint. A last aspect is that one refers here to worst-case complexity. While average-case analysis is often heard of, efficient SAT-solving bring the analysis of typically encountered instances to the fore. SAT-solving

indeed yields quite unpredictable run time, which can be reasonable in many typical applications.

However, there is also a more fundamental confusion at play here. The SAT problem is a synthesis question: find values of variables that satisfies the formula. This is a constraint satisfaction problem and cannot be expected to generally be easy to solve. But one does not necessary start from scratch, and one can indeed check a formula on a finite structure in polynomial time simply using the usual first-order semantics. One therefore need to clearly distinguish the question of verifying the validity of a formula on some structure from that of finding a structure on which the formula is satisfied.

Sadly, there is a misunderstanding of the nature of logic even in Artificial Intelligence (AI), a field that has strong ties with logic. There is of course a vibrant community of logicians in AI that have a very clear understanding of what logic is about and what it can bring to AI. However, this is just a small part of the AI community. Most AI researchers, while aware of logic as usually presented in AI textbooks, tend to miss the point.

For instance, with the striking advances and successes of machine learning, a whole generation of AI researchers often consider that logic, and more generally symbolic AI, is something of the past. In the mind of many, logic can be reduced to Prolog, an influential symbolic AI programming language now mostly replaced by conventional programming languages. Logic is hence seen as somewhat outdated.

On a more fundamental level, particularly in AI textbooks, logic is often reduced to IF/THEN rules and forward/backward chaining. Also, logic is often reduced to first-order logic, since most other two-valued logics can be reduced to it. However, this totally overlooks the question of devising the appropriate logical formalism for a setting, a line of research that is indeed highly active among AI logicians. Finally, the criticism that logic gives rise to inefficient technologies, is also often heard here also, in particular in relation to ontologies and semantic web technologies.

The next section takes these objections in consideration and proposes a logic course at the undergraduate level.

5 A tentative undergraduate first course in logic

This is all well and fine, but what should then be taught in an undergraduate logic course in computing? This section will propose such a course. The proposed content is obviously influenced by the author's experience and interests. Arguably, there are many conceivable alternative contents. The objective of this section is simply to put forward a possible content built around logic applications connecting and creating links to other computing subjects.

In accordance with current undergraduate education delivery in computing, this course proposes to develop skills in order to build logic-based models and apply logical methods to computing problems. The objective is therefore not to give a introduction to a broad selection of logic-based methods that could only

be further developed toward interesting applications in later courses. Indeed, the vast majority of undergraduate students strive for a professional career and this course ought to allow them to develop skills that they can use. Nevertheless this course is also a starting point in applying logic-based method for those who aim at more advanced studies in the applications of logics.

5.1 Propositional logic

This course is intended to come after a class similar to *Mathematics for Computer Science*[15] presented in Section 2. Students should therefore already have some familiarity with propositional logic, truth-tables, and have written some properties. This course would rapidly summarize this knowledge and then move on to show how propositional logic can be effectively used as a modeling tool, emphasizing links to other computing fields, and outlining the fundamental principles that makes these applications possible.

5.1.1 Modeling

From day one, it must be made clear that there are efficient tools for processing propositional formulas. The course should therefore start by introducing such a tool so that students can readily install and run it on some simple examples.

While a direct use of a SAT-solver is conceivable, it would surely be more convenient to rather use a *Satisfiability Modulo Theories (SMT)*[2] solver such as *Z3*[16]. This has many advantages. First, there are bindings for many programming languages, for instance C, C++, Java, Haskell, OCaml, Python,[17] allowing an approach in line with other computing courses. Secondly, this makes it possible to seamlessly go beyond pure propositional solving, using the reductions incorporated in the SMT solver. One can then process more complex data, such as integer arithmetic, without having to delve into the details of an explicit reduction to SAT. Both advantages come in handy particularly when devising assignments. As this course is not conceived for an advance audience, a typical assignment would nevertheless be to devise, implement, and use a SAT encoding for some graph or game problem or the encoding of some finite domain CSP.

In more ambitious settings, it is however entirely conceivable to devise an assignment that requires a SMT reduction. One could then briefly present the key points of replacement of terms by Boolean variables and the use of a domain solver, on some representative examples.

Alternatively, one could devise a more challenging assignment around the propositional modeling of time, as is done in *Bounded Model Checking (BMC)* [3]. Without introducing temporal logic and delve into the specific of BMC, it is possible to devise an assignment around a simple communication protocol, in the line of those presented in communications textbooks, or do a simple reachability analysis for a simple program.

[15] INF1132 Mathématiques pour l'informatique

[16] `https://github.com/Z3Prover`

[17] `https://en.wikipedia.org/wiki/Z3_Theorem_Prover`

5.1.2 Reachout and links to other computing fields

In order to show that logic is not an isolated subject and emphasize links to other computing subjects, it should be mentioned that SAT-solving is a special case of CSP, a whole subfield of AI. Furthermore, one could also mention that propositional logic also plays a key role in knowledge representation, games, and planning, in AI and also in formal verification of hardware and software systems.

When considering the time required to solve explicit SAT-instances, the relation to NP-completeness, that students could see maybe only later in their cursus, should be mentioned. Without getting into the formal definition, it can at least be mentioned that we do not know whether or not there is a polynomial time algorithm for SAT, and that most experts expect that there is none. Accordingly, all known algorithms are exponential in the worst case. An informal discussion on worst, average, and typical case (for some application setting) in relation to the run times encountered in class could prepare students for the more advanced classes on these topics but also allow students to reflect on their own run time experiments in this class.

Connection to computer architecture could also be touched upon.

5.1.3 Principles

The course should stress the fact that SAT is a fundamental reasoning question to which essentially all specific questions of a finite nature can be reduced. It should be also emphasized that this includes using SAT to check/validate the model and infer new knowledge. This will surely arise naturally when modeling.

Indeed, one can check and validate a model with additional constraints that check that some specific expected behavior is actually allowed by the model. Otherwise, one will debug the model by removing constraints to locate which parts prevent the expected behavior.

Furthermore, it is important to stress that logic is a general setting for answering a broad class of questions. For instance, while one can solve puzzles, such as Sudoku, one can also study any question expressible in this setting. One can therefore ask whether an element of some set of digits is present in some zone and analyze the game beyond finding a solution. On a more general, but related level, satisfiability solves most logical questions, such as equivalence of formulas.

After some modeling, where errors, multiple formalizations, and ambiguities will arise, it could be the right moment to show that while there can be multiple non-equivalent but relevant models, the interpretation of a specific propositional formula is unequivocal and unambiguous. It is important to give the formal semantic, which is recursive, and simply implementable. One should also emphasize the fact that the semantics effectively gives a single meaning to a formula, no matter how complex.

As to the algorithms behind SAT solving, one should at least mention the general inference and search principles applicable to any CSP. This yields, for

Conjunctive Normal Forms (CNF) the *unit propagation* and *chronological backtracking* DPLL algorithm [4].

Showing how to translate a propositional formula into CNF should also be attainable. For the general principle, one can note that a formula is satisfied when the variables do not agree with any false truth-table line. This directly yields a CNF, possibly of exponential size. This should generate interest in the fact that there are however translations into an *equi-satisfiable* CNF of linear size. Tseitin's translation could be explained with new variables for the internal nodes of the syntax tree and adding clauses ensuring a value compliant with the connector's definition. However, this could also be left to more advanced courses.

I do not expect that one could do much better than mention that modern SAT solvers use more advanced methods to prune the search space. However, in some places, if time permits, discussing some key principles of the *Conflict-Driven Clause Learning (CDCL)* algorithm [6] could be considered.

5.2 Description logic

One could expect the course to move from propositional to first-order logic. Yet, keeping with the emphasis on applications, I rather propose to present description logic. Description logic, or logics, is a family of logics devised for knowledge representation, in particular in relation with the semantic web. Technically, these logics can be seen as extensions of (multi-) modal logics or as decidable fragments of first-order logic. As we will see, this choice will not preclude the presentation of fundamental facts about first-order logic.

5.2.1 Modeling

Keeping up with the principle of starting with a tool that students can use from day one, I propose Protégé[18] with the Hermit[19] reasoner.

Protégé is a free and open-source editor with a sizable number of contributed plug-ins for various reasoning and data processing tasks. This tool offers a nice GUI allowing to define concepts (unary relations) and roles (binary relation) in a way reminiscent of UML class (and object) diagrams. Description logic reasoning can be done by various plug-ins, such as Hermit. This setting works pretty well for small to medium size descriptions or as a starting point in larger cases that can then be better processed with the stand-alone version of the reasoner.

Basically, on the technical level, description logics allows unary (concept) and binary (role) relations, and guarded quantifications of the form $\forall y(R(x,y) \to \varphi)$ and $\exists y R(x,y) \wedge \varphi$. This part of the course should hence start with a recap on binary relations: reflexivity, symmetry, transitivity, but above all composition.

Knowledge representation through binary relations with constraint on compositions of these binary relations occurs repeatedly in computing. However, it

[18] `https://protege.stanford.edu/`
[19] `http://www.hermit-reasoner.com/`

is rarely explicitly addressed, emphasizing the meaning of specific compositions to determine required inclusions between their images (ranges).

This can be addressed from the outset as this already occurs in such simple textbook examples of students enrolled into classes. In that case, one can enforce the constraint that a student is enrolled only in courses of its registered program. On the diagram, this reduces to following two different paths, to the same destination. It is always surprising that many students tend to read the diagram simply as access paths, overlooking that different paths usually yield different relations. It is hence of paramount importance to allow students to work with many representative diagrams, explicitly writing down the meaning of compositions in natural language. This is the require first step in devising image inclusion constraints.

Description logic also allows cardinality constraints similar to UML ranges ($k..l$, $k..*$). Incrementally introducing description logic constructs, one can develop skills in writing, reading, and understanding these formal expressions.

From the starting point one must emphasizes that modeling is about eliciting the crucial facts in the considered setting. This can be done in different, possibly non-equivalent ways, but clearly one should not overload a model with irrelevant details. It should be made clear that modeling is about devising proper concepts and relations in order to describe the system in a natural way. Students should hence be invited to discuss and compare their models.

Now, as different models are devises and compared, it will be natural to introduce computational inference. The reasoner plays an integral part of the tool, since it not only allows to infer implicit knowledge, but also to experiment in order to understand limitations and shortcomings of the model. A profitable standpoint is to make the student aware that a model is always incomplete. Furthermore, when some aspect of the system is missing in the model, the inference engine clearly cannot conclude on this aspect. The pleasant thing about the Protégé/Hermit environment is that one can add/remove description logic constraints and experiment with what the reasoner can conclude. This helps a lot, raising awareness of the knowledge actually encompassed by the model. In practice it can be challenging to figure out exactly what is encompassed by our model. The tools help to find this out, enhancing our reasoning abilities by turning our attention to relevant parts of our model.

5.2.2 Reachout and links to other computing fields

The link to Software Engineering (SE) modeling, UML class and object diagram, is pretty direct and natural to establish. In both cases, the underlying framework is based on unary and binary relations. Furthermore, SE has a long tradition establishing that writing down a description already helps to communicate and analyze a design, independently from any tool or processing framework. Logic adds inference of implicit knowledge.

Ontologies in the setting of the semantic web and W3C standards should obviously be mentioned. From triple stores one can touch on relational Databases, emphasizing that they, more generally, use n-ary relations. A brief discussion

of close world assumption and verifying within a structure, vs., open world and reasoning on all structures, could be of interest. This would nicely contrast data processing in databases vs. in the semantic web.

5.2.3 Principles

The course should address at least some elements on how inference engines work. A detailed algorithmic description of inference in description logic is wide beyond what should be expected here. It is nevertheless possible to give the key elements and develop an understanding of the fundamental aspects at stake.

Tableaux methods[7] are quite effective with Description Logic [1]. Furthermore, the basic tableau method can be presented quite concisely both for propositional and first-order logic, by reducing to *negative normal form (NNF)*.

The course should hence present the tableau method by first transforming a formula into NNF, using de Morgan laws. Tableaux should be presented first in the context of propositional logic, introducing all basic concepts: open branch, closed tableau, and the fact that a tableau proof is a proof by refutation. This allows to emphasize that the tableau steps correspond to the intended meaning of the connectives. The proof of the completeness theorem is also attainable by a case analysis. This could even be shown by an analysis of the different cases on representative examples. Termination is easily justified since a formula is decomposed at each step.

Tableaux for first-order logic and completeness can be done among the same lines. Branches are somewhat more involved since an open branch defines a model. The method can hence be presented on some fix language not to overload the key points. Justification of completeness can be done along the lines used in the propositional case, obviously emphasizing that the construction won't terminate and that the constructed structure will be infinite, in general. One can however have student work on examples where it does indeed terminate, yielding a finite counter-example. This will show that there are indeed cases where this construction terminates.

Obviously, this is not the full story and tableaux methods for description logic are indeed much more involved. However, I would not suggest more in that direction in this course. Nevertheless, if time permits, one could show how description logic is embeddable in first-order logic, by formula translation. But, in any case, one should mention that tableaux methods are indeed extended to description logic, and that in most cases (there are many description logics variants) the algorithm terminates and yields a finite counter-example. One can then conclude the class by mentioning that for first-order logic in general, infinite structure are essential for completeness, i.e., that there is no completeness when restricted to finite structures.

6 Conclusion

In computing, research and practice areas are primarily defined by the intended applications rather than by the methods used. While logic contributed and is still contributing strongly to computing, it cannot ignore this fact. It is therefore of paramount importance to strongly put forward the applications that have emerged from our field.

However, as we have seen, there is truly little logic teaching in current computing degrees. But, the development of computing, and especially the maturity of many logic-based methods offer a major opportunity to greatly extend logic teaching in modern professionally-oriented computing degrees.

Logic is relevant and it is of paramount importance to convey that formalization of thoughts and inference lead to many fruitful computing applications. This brings modeling, a central activity in computing, to the fore, but in a principled and sound way. This is indeed challenging and requires some efforts. However, this goal is totally attainable, and this, with an approach following current teaching practices in undergraduate computing degrees. This is surely also an ideal opportunity to counter unfortunate stereotypes around logic and make important advanced in our field much better known, among our students and colleagues.

Section 5 has proposed an undergraduate logic course around SAT and description logic. Arguably, this is not the only relevant possibility. However, this proposal allows to build a course that put modeling to the fore and develop hands-on skills with tools appropriate beyond the classroom. The presented topics are also representative of advances of the last decades and are readily applicable. As shown in that section, this choice of topics also allows to present fundamental notions and principles and establish connections with other computing fields.

Establishing such a course is possibly an ambitious endeavor. This is however a challenge that can, and should, be met by the logic community to further disseminate logic knowledge and make the case of the contribution of our field to computing.

References

[1] F. Baader, I. Horrocks, C. Lutz, and U. Sattler. *An Introduction to Description Logic*. Cambridge University Press, 2017.

[2] C. W. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Satisfiability modulo theories. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 825–885. IOS Press, 2009.

[3] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in Computers*, 58:118–149, 2003.

[4] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[5] Joint Task Force on Computing Curricula, Association for Computing Machinery (ACM) and IEEE Computer Society. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. Association for Computing Machinery, New York, NY, USA, 2013.

[6] J. P. M. Silva, I. Lynce, and S. Malik. Conflict-driven clause learning SAT solvers. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 131–153. IOS Press, 2009.

[7] R. M. Smullyan. *First-order logic*, volume 43 of *Ergebnisse der Mathematik und ihrer Grenzgebiete*. Springer-Verlag, Berlin, Heidelberg, and New York, 1968.

[8] M. Y. Vardi. Where have all the domestic graduate students gone? *Communications of the ACM*, 63(9):5, Aug 2020.

[9] R. Villemaire. Logic modelling. In *4th International Conference on Tools for Teaching Logic TTL 2015*, volume abs/1507.03686 of *CoRR*. arXiv, 2015.