FOLL-E: Teaching First-Order Logic to Children

Simon Vandevelde, Joost Vennekens

KU Leuven, De Nayer Campus, Dept. of Computer Science, Belgium
Leuven.AI – KU Leuven Institute for AI, B-3000 Leuven, Belgium
Flanders Make – DTAI-FET
{s.vandevelde, joost.vennekens}@kuleuven.be

Abstract

First-order logic (FO) is an important foundation of many domains, including computer science and artificial intelligence. In recent efforts to teach basic CS and AI concepts to children, FO has so far remained absent. In this paper, we examine whether it is possible to design a learning environment that both motivates and enables children to learn the basics of FO. The key components of the learning environment are a syntax-free blocks-based notation for FO, graphics-based puzzles to solve, and a tactile environment which uses computer vision to allow the children to work with wooden blocks. The resulting FOLL-E system is intended to sharpen childrens' reasoning skills, encourage critical thinking and make them aware of the ambiguities of natural language. During preliminary testing with children, they reported that they found the notation intuitive and inviting, and that they enjoyed interacting with the application.

Introduction

First-order logic (FO) is an important foundation of mathematics, philosophy and computer science. Zooming in on computer science, many curricula offer students an introductory logic course, which helps students to, e.g., reason about the correctness of programs, prove properties such as termination of an algorithm, analyse requirements, etc. Also in the domain of AI, the importance of logic can hardly be exaggerated. Indeed, many of the first AI systems were based on FO logic, and to this day there is a vibrant community that uses logic for AI, in fields such as Knowledge Representation, Logic Programming, Theorem Proving and SAT solving.

Many key insights from computer science and computational thinking have been successfully taught to children of various ages (Bell and Vahrenhold 2018). Probably the most spectacular example are block-based programming environments, such as Scratch (Junior) (Resnick et al. 2009; Maloney et al. 2010) and Blockly (Fraser 2015), which are used around the world to introduce even very young children to key concepts of algorithmic thinking. Learning computational thinking at an early age may help children in making more informed decisions about their future field of study, and may provide them with useful skills, either for a career in CS or some other domain.

While Scratch is a very successful example, there exist also numerous other efforts to make important ideas from computer science accessible to children. For instance, in the field of AI, research has been done into helping children understand concept of neural networks and deep learning (Rauber and Gresse von Wangenheim 2022; Lane 2021). One topic which appears to be curiously understudied, however, is the teaching of FO to children. On the face of it, logic would seem to be a prime candidate: it is not only important in computer science and AI, but it can also help children in a variety of other disciplines. Moreover, it nicely complements the Scratch-like tools: these tools teach children to think about how algorithms work and how computers perform tasks, while logic could help children to think about requirements and about properties of problem domains. In other words, Scratch can teach children the "how", while logic could help them to figure out the "what".

The main research question of this paper is therefore:

> Is it possible to use a "Scratch-like approach" to teach first-order logic to children?

At first sight, there seems to be a number of obvious objections, such as:

- Logic has a steep learning curve. Maybe its syntax and semantics are too complex for children to understand.
- Logic does not "do" anything. When children learn to program, especially in a graphical environment, they can immediately observe what a program does, thereby getting valuable direct feedback. Such feedback may be impossible to obtain when learning logic.
- Logic is not fun. Animations, sounds and graphics can all help to get children engaged in learning how to program. By contrast, logic may be too boring to learn.

In this paper, we will investigate whether these objections can be overcome by designing an appropriate learning environment. First, we will clarify the concrete goals of this environment. Then, we describe our approach to achieving these goals, both conceptually and technically. Afterwards, we present FOLL-E (First-Order Logic Learning Environment), a concrete implementation of a blocks-based tool for teaching FO to children. We also discuss our experiences

and observations of using FOLL-E at two events centered around introducing CS to children. Finally, we finish by concluding, and lay out potential future work.

Concrete Goals

When designing the tool, we will attempt to achieve a number of concrete goals. The tool will have children write logic formulas in order to perform certain tasks, with the goal of teaching them certain skills. Some of our goals have to do with the format in which the logic is written, while others have to do with the task that is to be performed, and still others with the skills that the children should learn.

Skills

So far, we having been talking about "teaching children first-order logic". Since FO can be used in different ways and for different purposes, we should start by specifying what it is precisely that we want children to learn. In this paper, our goal is to make children understand which piece of knowledge is expressed by an FO formula, and conversely, to be able to formalise a given piece of knowledge by means of an FO formula. In other words, we want children to understand that a formula such as

$$\forall x : Human(x) \Rightarrow Mortal(x) \tag{1}$$

means that "all humans are mortal", and we want them to be able to come up with this formula as a representation of this knowledge.

Our tool will therefore have a model-theoretic rather that proof-theoretic view on FO: we will focus on teaching the relation between a formula and its models, rather than on constructing proofs or deducing formulas. We make this choice because we view this as the more foundational skill: if the children do not understand what a formula means or how they can correctly formalise a certain statement, then deduction will not be of much use to them anyway, because they will not understand what it is that they have deduced.

Representation of FO

The traditional way of writing logic in mathematics—that is, the use of formulas such as (1)—is not well-suited for our purposes. Our goal is therefore to design a way of writing FO formulas that is more suited for getting children to understand the essence of FO, while avoiding needless struggles with syntax. In particular, we have the following goals for our notation.

- *Avoid discouraging the children with finicky syntax.* The traditional notation requires learning several new symbols, and how to combine them correctly. Forgetting the meaning of a symbol or hitting syntax errors may discourage children, so we want to avoid this as much as possible.
- *Make the structure of formulas clear.* To know what a formula means, it is necessary to understand its structure. In traditional notation, this requires knowing the rules of operator precedence, which introduces an additional layer of complexity.
- *Encourage experimentation.* Building different formulas and figuring out their meaning is an excellent way of learning. We want a notation that encourages children to try things out.
- *Encourage collaboration.* Talking about formulas is also an excellent way of learning: trying to explain the meaning of a formula to someone else is typically a great way of increasing not only the other person's but also your own understanding.

Task to Solve

Our tool will ask children to construct formulas in order to solve certain tasks. In designing the tasks to be solved, we have the following goals.

- *The task should be focused on the $\models$-relation between formulas and their models.* In other words, the tool should actually be teaching the skills that we want to teach.
- *The task should allow clear and immediate feedback.* A tight feedback loop is crucial for the learning process, so if the user makes a mistake, we should be able to tell them as quickly as possible what they actually did wrong.
- *The task should allow a gradual increase in difficulty.* To create a succesful learning trajectory, we should be able to start from very simple exercises and gradually build up to more complex ones.
- *The tasks should be engaging and fun.* A fun task should help to keep children motivated throughout the learning process.

Conceptual Approach

To achieve the goals identified in the previous section, we opt for the following approach. The task that we ask the user to perform is to construct a formula that distinguishes a given set of "good" examples from a given set of "bad" examples. For instance, we could present the user with a red square, a blue square and a green square as good examples and a red circle, a blue circle and a green circle as bad example, and the solution could then be the formula $Shape = Square$. Here, the vocabulary in which the formula is to be written (e.g., the symbols $Shape$ and $Square$), is part of the assignment. Such a task has the following benefits:

- It is focused on the model-theoretic semantics of formulas, corresponding to our goals.
- It avoids the use of natural language: if we were to ask the user to formalise a given natural language statement ("the shape is square"), the user would first have to correctly understand the statement, which may introduce problems that are related to reading comprehension.
- It introduces a puzzle element. By asking the user to figure out by themselves what the piece of knowledge is that has to be formalised, they are presented with an additional challenge, which they hopefully perceive as fun. Moreover, the puzzle element is not simply "tacked on", but it focuses on the difference between the models and

non-models of a formula, which goes to the essence of what the user should learn.

- It allows gradually increasing difficulty. This general framework has enough flexibility to allow a wide range of difficulty.
- We can present the different examples as pictures to the user. By choosing a topic that is more whimsical than just coloured shapes, we can introduce a visually appealing and fun setting.
- The setting allows for an intuitive way of providing feedback to the user: for an incorrect solution, we can highlight precisely those examples that are misclassified (i.e., those that should be models of the formula but are not, and those that should not be models of the formula but are). This tells the user what they did wrong and may immediately point them in the right direction for fixing their solution.

The notation that we use for FO-formulas is a visual, block-based notation. The success of Scratch-like tools suggests that this is a good approach, which achieves the goal of being syntax free and engages the users' visual intuitions. By carefully designing the blocks, we ensure that it is impossible for the user to create syntax or typing errors. To visualise the structure of the formulas, we lay-out the blocks in the style of a parse-tree, placing subformulas below the formulas in which they appear. Finally, the blocks can be shaped like pieces of a puzzle, which encourages users to try to fit different pieces together, thereby encouraging experimentation.

The last design choice that we make is to allow the user to build formulas with actual wooden blocks, rather than by just using a GUI on a computer. By providing the user with a tactile, hands-on experience, we hope to further encourage both experimentation and collaboration in a playful way. Manipulating the building blocks together with classmates seems more easy and natural when they are real-world objects, instead of pictures on a computer screen.

Concrete Implementation

This section introduces FOLL-E (First-Order Logic Learning Environment), the concrete implementation of our proposed approach. For this tool, we want a specific problem domain which is (a) enticing and captivating for children and (b) simple enough for them to experiment with. While this is a fine line to tread, it may have a substantial effect on the attractiveness of the tool, and the children's engagement with it.

For the application domain, we have chosen the design of simple robots, as shown in Fig. 1. The robots have been hand-drawn in a cartoonish style with vibrant colours to create an inviting visual. A robot consists of six body parts in total, called *components*, which can be individually coloured in one of three colours. Specifically, a robot consists of two arms, two legs, a head and a body, which can each be coloured either red, blue or green. Additionally, a robot may also wear a hard hat.

To express constraints on an application domain, we devised a generic blocks-based formalism for FO. It was de-

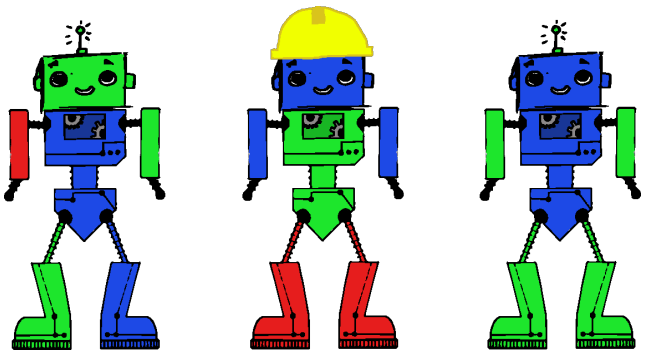


Figure 1: Example robot designs.

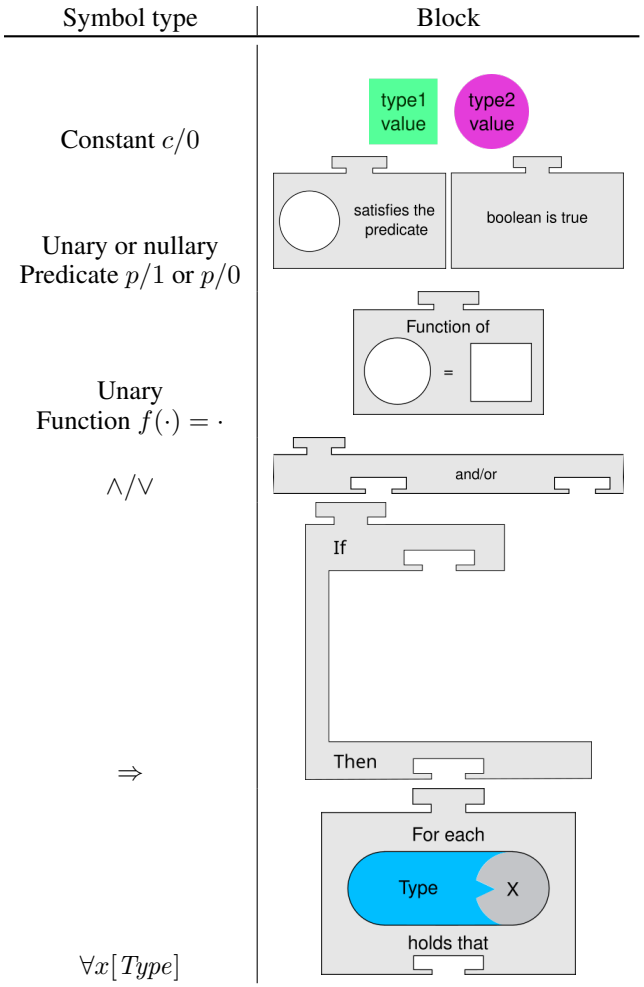


Figure 2: Overview of all generic blocks

Block type	Specific Block Meaning
Constant	Blue, Red, Green, Right Arm, Left Arm, Right Leg, Left Leg
Proposition	Robot wears a helmet
Predicate	. . . is an Arm, . . . is a Leg
Function	Color of . . . = . . .
Implication	If . . . Then . . .
Con-/Disjunction	. . . And/Or . . .
Type	Component
Variable	c
Quantifier	For each . . . holds . . .

Table 1: Overview of the blocks specific to the robot application, and their general block type. ". . ." signifies an open slot.

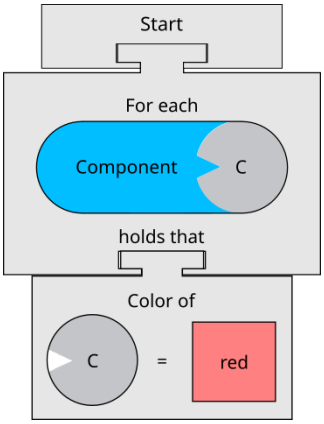


Figure 3: Blocks expressing "$\forall c : Color(c) = red$"

signed with a *pegs-and-slots* approach in mind, in which blocks physically fit together if the connection is syntactically correct and the formulas are correctly typed. Fig. 2 shows an overview of the specific blocks. Note that our application uses only predicates and functions with arity ≤ 1, but the notation can of course be extended to higher arities if needed.

These generic blocks can be instantiated to specific blocks for a chosen vocabulary. For our tool, we prepared a library of blocks specific to the robot domain. In total, 16 different expression blocks are available in the tool. An overview of the specific blocks is shown in Table 1, together with the generic block type that they belong to. We represent all terms of type $Component$ by squares and terms of type $Color$ by circles. Future applications could use more types mapped to more such basic shapes. Variables are represented by the shape of their type with an additional cut-out triangle in their left side. This allows a variable to be used wherever another term of that type could be used, but restricts the terms that can be quantified over to variables. Fig. 3 shows a simple example.

The blocks are laser cut out of 3mm plywood and engraved with their intended meaning in text and, when applicable, also with an icon such as an arm or a helmet. Blocks

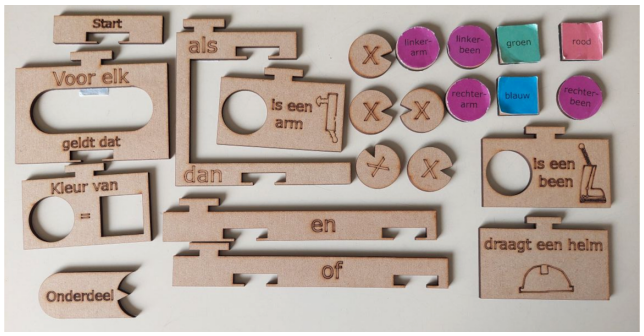


Figure 4: The laser-cut blocks included in each FOLL-E box. (Text engraving in Dutch)

representing a colour were also painted in that colour. In this way, the notation is low on text, with much visual stimulation. As an added bonus, this also leads to better visibility from a distance, making it easier for the teachers to assist. Fig. 4 shows the end result.

In the application, the children are shown three "good" robots and three "bad" robots, and must devise a rule that distinguishes between them. By puzzling together blocks, the children can express sentences of increasing difficulty such as "The left leg is blue", "If the robot wears a helmet, it has a green body" and "Every component that is a limb must be coloured green.". Similar to Scratch, the tool is "always live". A Raspberry Pi mini-computer continually scans the playing area and, as soon as it detects a complete sentence (i.e., all the slots of the connected blocks have been filled), it converts this into FO. The IDP-Z3[1] reasoning engine (Carbonnelle et al. 2022) then checks whether the "good" robots indeed satisfy the formula, in which case a green check mark pops up next to them. Similarly, the "bad robots" are checked for unsatisfiability, in which case a green cross is shown. Finally, IDP-Z3 performs propagation, i.e., computes those properties that hold in all models of the formula. These are then shown as a partially coloured robot, where for example only the left leg is coloured blue. This robot is displayed in the centre of the screen, to help the children understand which property they actually expressed.

In this way, FOLL-E gives immediate feedback which results in high interactivity, allowing the children to quickly explore the effects of specific blocks. Moreover, the feedback is purely visual in the form of the check/cross marks and the centre robot, which we feel makes the application more intuitive and enticing. In total, we hand-crafted ten levels of increasing complexity for the children to play with.

The complete tool consists of the laser cut blocks, a box with a clear panel on top and a Raspberry Pi 3 B+ with a Raspicam inside, a computer monitor and a keyboard. The Pi is able to detect the individual blocks by the fiducial markers glued to them. Specifically, we use ArUco markers (Garrido-Jurado et al. 2014). To detect complete sentences, we chain blocks together by starting at the *start* block and recursively looking for attached blocks. This approach has multiple ad-

[1]https://www.IDP-Z3.be

vantages: it is invariant to the blocks' rotation, the designs can be changed easily by replacing the markers and occlusions are not possible due to the camera placement.

The application is written entirely in Python 3, with the GUI created using the Pygame (Pygame Project 2021) library. All source code is available online[2]. To ensure smooth communication to and from the IDP-Z3 system, the application makes use of *IDP-Z3Py*, an as of yet unreleased Python API which aims at facilitating the embedding of IDP-Z3 in Python applications. A preliminary version of this API is contained in the code repository. The total cost of one FOLL-E box, including the Raspberry Pi and the laser cutting but excluding the monitor and keyboard, comes down to about €60.

Figure 5 shows a few photos of children playing with FOLL-E.

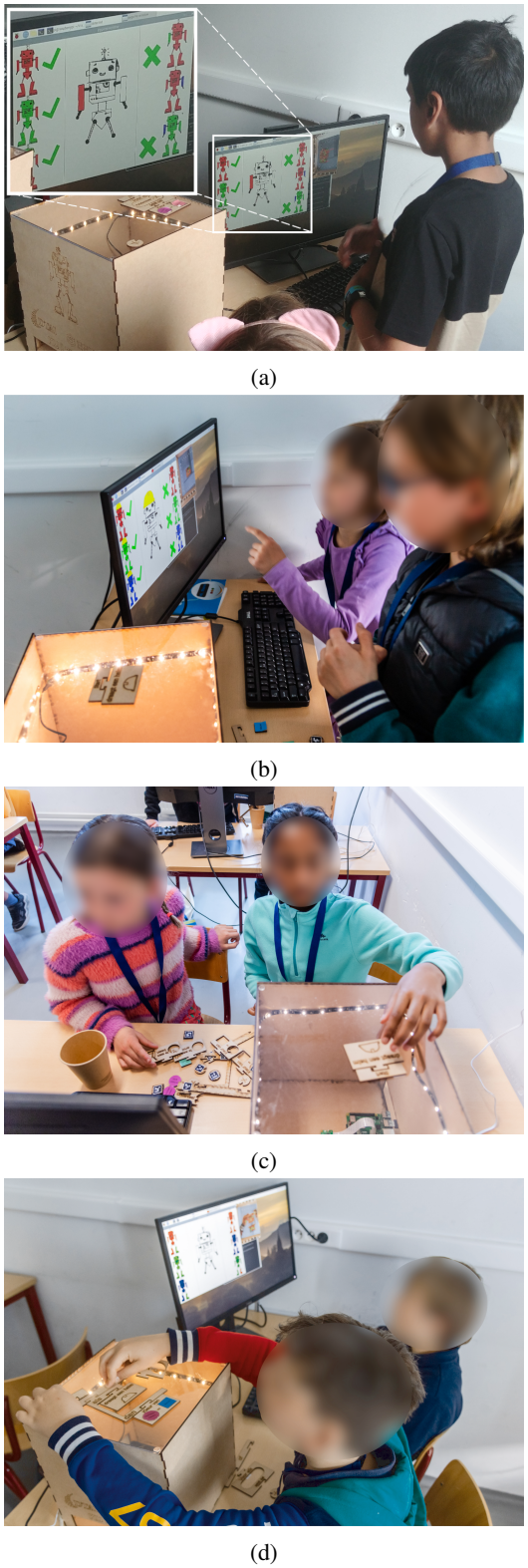
Results

As a preliminary evaluation, we tested FOLL-E at two separate events aimed at different age groups. The first event was a "STEM-workshop" organised for high school students aged 16-18, typically with a technical background. The students were split into two groups of 15 and 9 students, which both partook in our workshop on Logics, but at different times of the day. During this four hour workshop, we aim to make the students aware of human reasoning by analysing our thinking behaviour when solving simple puzzles such as Sudoku. By gradually increasing the level of abstraction used to solve the puzzles, the students implicitly start solving SAT-encodings of the logic puzzles manually. During the last 30 minutes of the workshop, we briefly introduce the role of reasoning engines such as IDP-Z3 in solving such problems automatically, after which the students work through the FOLL-E puzzles in pairs.

The second event, the "Children's University", is an annual event for children between the age of 8 and 13. Both the morning and afternoon consist of a lecture of two hours and a workshop of 1.5 hours. In total, 28 children (13 morning + 15 afternoon) registered for our workshop "Teach the computer". During the workshop, we first spend 30 minutes on an interactive exercise in which we illustrate that computers are by themselves incapable of solving problems, and should be "taught" instead. Afterwards, they are given a 5 minute introduction to FOLL-E after which they are free to play through the levels in pairs.

We observed the children throughout both events, and will now summarize our observations. Firstly, the blocks-based notation successfully eliminates both syntax errors and type errors. The errors that the children did make fall in one of two categories: *reasoning errors* and *formalization errors*. This first category contains the errors made when trying to deduce the rule that distinguishes good from bad robots. The second category of errors happen during formalization, where children incorrectly translate their own understanding of the rule into the blocks-based notation.

Regardless of their age, all student pairs typically follow the same approach. First, they examine the robots and dis-



(a) (b) (c) (d)

Figure 5: Photos taken during the Children's University workshop showing children playing with FOLL-E.

[2]https://gitlab.com/Vadevesi/foll-e

cuss what they think the distinguishing rule is. Once they agree on a rule, they try to represent it using the blocks. If their solution is not correct, they try to see if the error is in their formalization or their reasoning, and correct it accordingly. If they really can not figure it out, they ask for help.

All students in all groups were able to complete all ten levels, but the time required varied with age. The 16-18 year olds completed all levels in 20 minutes or less, whereas the 8-13 year olds generally needed 40-50 minutes. A significant time difference was also noticeable within the younger groups, where the children near the upper age limit (13) finished about 15 minutes ahead of the others.

One of the co-authors of this work has extensive experience with teaching Scratch to children aged 8-13, and reports the following differences:

- The use of physical blocks makes the tool more inviting than Scratch, and makes collaboration easier.
- Students are more focused. In Scratch, they are easily distracted by all of its bells and whistles.

Of course, our scope differs from Scratch's, not just in the skills we teach, but also in the fact that we fix the vocabulary that is to be used up front, while Scratch allows children to define their own functions.

In both age groups, children who had previous experience with Scratch initially had some difficulty adjusting to the declarative paradigm. Instead of the intended declarative solution such as "robot wears a hat", they initially tried to express an imperative statement such as "if the robot wears a hat, then it is correct".

The children reported that in general (a) they enjoyed their time with FOLL-E, (b) found the puzzle-like nature inviting and (c) liked the assignment. One of the older Children's University students said: "Being able to see how it works is very impressive", with regards to the visualization of the aruco marker detection. Another mentioned afterwards that "It's satisfying to puzzle the pieces together and see the result, especially if it's correct."

Many of the younger children reported some difficulty spikes between the levels. Among others, the transition to levels using quantification was difficult. A possible explanation for this is that some children solved the levels that were meant to initially introduce quantifiers in an untended way, without actually using quantification, which lead to a difficulty spike later on. The tests also revealed a few minor bugs. Syntax errors are technically still possible by placing an incorrect block *on top* of another block's slot instead of fitting it inside. In these situations, the application simply crashes. Also, sentences without models (such as "left arm is a leg"), crash the application, which should not happen.

To measure the effect of the tool on the logical reasoning of the children, we surveyed both Children's University groups (aged 8-13) using a questionnaire that gauges logical thinking. This questionnaire consists of four questions on spatial reasoning, and four questions on propositional reasoning. Examples of these questions are as follows:

> "The pen lies to the left of the book, and the book lies to the right of the pencil. Is the pencil definitely to the left of the pen?" (spatial)

and

> "If there is an L on this page, there is also a 7. There is no 7 on this page. Is there definitely an L? " (propositional)

The students in the morning session took the this test before using the tool, as control group, and the afternoon students took the test after using the tool. This resulted in an average score of 5.46/8 for the control group and 4.6/8 for the intervention group, which therefore performed worse than the control group. A possible explanation might be that the control group took the test at 11.30h, while the intervention group took it at 16.15h, at the end of a busy day. In future work, a test with larger groups in more controlled circumstances will be done.

Related Work

Interest in teaching logic to children has been around for a long time. For example, Kowalski (1982) reports on how they teach logic programming to children using Prolog, with the aim of learning how to apply logic to other subjects, and to spark interest in programming. While this work has the same target audience as ours, its goal is different, since they aim to teach (logic) programming whereas we are focused on the declarative meaning of FO formulas. Most tools for teaching logic are geared towards university-level students (Lodder, Jeuring, and Passier 2006; Mauco, Ferrante et al. 2009; Hatzilygeroudis and Perikos 2009). We briefly discuss some of the tools that could also be used to teach to K-12.

An overview of early work is provided by (Goldson, Reeves, and Bornat 1993). Typically, these tools were focused on creating proofs and verifying them. The most well known of these is *Hyperproof* (Barwise and Etchemendy 1992), a descendant of *Tarski's World* (Barwise and Etchemendy 1991), which implements a graphical environment to teach First Order Logic syntax and semantics. The user is tasked with expressing formulae on labelled geographical shapes in a 3D-world. For instance, if cube *A* is in front of tetrahedron *B*, they could write "*Cube(A)*", "*Tet(B)*" and "*FrontOf(A,B)*" If all cubes are in front of a tetrahedron, they could write "$\forall x : Cube(x) \Rightarrow \exists y : Tet(y) \wedge FrontOf(x, y)$". A survey by Fung et al. (1996) on students in a logics CS course reported the benefits of Tarski's World (and by extension, Hyperproof) as encouraging exploration, removing difficulty spikes and being able to visualise quantification.

Reeves (1996) notes that expressing logic specifically on geometrical shapes is not sufficiently *captivating* for students, as there seems to be no connection between the "subject matter" and CS. Therefore, the author proposes an extension of Tarski's World in which students reason over graphical user interfaces instead, stating that this is more relevant to them and thus more motivational.

Mauco, Ferrante, and Felice (2014) present two tools, *FOST* and *LogicChess*, which like Hyperproof allows students to express FO statements about objects in a graphical environment. The tools can check the statements for syntax errors, and evaluate their truth values. In the paper, the authors also remark that both these tools are domain-specific

with pre-coded relations and functions. They regard this as a major downside, as "FO's expressiveness lies in the possibility of working with arbitrary domains", and not just the predetermined domains. To overcome this, the authors propose a framework for didactical FO tools focusing on extendability.

Our approach differentiates itself from these previously mentioned works in multiple ways. Firstly, it has a model-theoretic view on FO rather than a proof-theoretic view, which we regard as a more foundational skill for children. Secondly, we use a novel blocks-based notation for FO, in which children do not need to pay attention to FO's syntax and can thus focus more on creating sentences with a correct meaning. They also do not interact with the notation through a GUI, but instead write formulas with wooden blocks, resulting in a type of CS Unplugged (Bell and Vahrenhold 2018). For these reasons, we feel that our approach results in a more captivating experience, in which children remain engaged longer in their tasks.

Conclusion and Future Work

First-order logic has always been a cornerstone of computer science and AI. Yet, it is difficult to teach in general due to its steep learning curve. In this paper, we present a novel blocks-based notation for FO aimed at children, which is designed to be syntaxless, structured and playful. Using physical blocks further enforces the playful nature while stimulating experimentation and team work.

Based on these blocks, we created FOLL-E, a First-Order Logic Learning Environment with a focus on high interactivity, visual feedback, and fun. Children are asked to express knowledge that distinguishes "good" examples from "bad" ones. These blocks are then scanned and converted to FO, after which a logical reasoning engine verifies the (in)correctness of the examples. This result is then shown, together with a visualisation of the consequences of their expression.

We tested the tool with around 50 children aged between 8-13 and 16-18. They were all enthusiastic, stating that the notation is intuitive and inviting, and that they overall enjoyed their time.

In future work, we will be teaming up with cognitive psychology researchers to correctly measure the benefit of the tool. Furthermore, we will be looking at what changes can be made to make the tool more effective, such as introducing more levels, experimenting with different applications besides robot design, and adding automated error explanation.

Acknowledgments

This research received funding from the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme. The authors would also like to thank Kaat De Bock (https://www.kaatdebock.be) for their robot illustrations, Davy of Makerspace De Nayer for assisting us with lasercutting, Walter Schaeken for his help with preparing the questionnaire, and Maarten Vandersteegen and Kristof Van Beeck for their technical suggestions.

References

Barwise, J.; and Etchemendy, J. 1991. The Language of First-Order Logic. In *Center for the Study of Language and Information*.

Barwise, J.; and Etchemendy, J. 1992. Hyperproof: Logical Reasoning with Diagrams. In *Working Notes of the AAAI Spring Symposium on Reasoning with Diagrammatic Representations*, 80–84.

Bell, T.; and Vahrenhold, J. 2018. CS Unplugged—How Is It Used, and Does It Work? In Böckenhauer, H.-J.; Komm, D.; and Unger, W., eds., *Adventures Between Lower Bounds and Higher Altitudes: Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday*, 497–521. Cham: Springer International Publishing. ISBN 978-3-319-98355-4.

Carbonnelle, P.; Vandevelde, S.; Vennekens, J.; and Denecker, M. 2022. IDP-Z3: A Reasoning Engine for FO (.). *arXiv preprint arXiv:2202.00343*.

Fraser, N. 2015. Ten Things We've Learned from Blockly. In *2015 IEEE Blocks and beyond Workshop (Blocks and Beyond)*, 49–50.

Fung, P.; O'Shea, T.; Goldson, D.; Reeves, S.; and Bornat, R. 1996. Computer Tools to Teach Formal Reasoning. *Computers & Education*, 27(1): 59–69.

Garrido-Jurado, S.; Muñoz-Salinas, R.; Madrid-Cuevas, F.; and Marín-Jiménez, M. 2014. Automatic Generation and Detection of Highly Reliable Fiducial Markers under Occlusion. *Pattern Recognition*, 47(6): 2280–2292.

Goldson, D.; Reeves, S.; and Bornat, R. 1993. A Review of Several Programs for the Teaching of Logic. *The Computer Journal*, 36(4): 373–386.

Hatzilygeroudis, I.; and Perikos, I. 2009. A Web-Based Interactive System for Learning NL to FOL Conversion. volume 226, 297–307. ISBN 978-3-642-02936-3.

Kowalski, R. A. 1982. Logic as a Computer Language for Children. In *ECAI*, 2–10.

Lane, D. 2021. *Machine Learning for Kids: A Project-Based Introduction to Artificial Intelligence*. No Starch Press. ISBN 1-71850-056-4.

Lodder, J.; Jeuring, J.; and Passier, H. 2006. An Interactive Tool for Manipulating Logical Formulae. *Technical Report UU-CS*, 2006.

Maloney, J.; Resnick, M.; Rusk, N.; Silverman, B.; and Eastmond, E. 2010. The Scratch Programming Language and Environment. *ACM Transactions on Computing Education*, 10(4).

Mauco, M. V.; Ferrante, E.; et al. 2009. Clausula: A Didactic Tool to Teach First Order Logic. *Proceedings de ISECON*.

Mauco, V.; Ferrante, E.; and Felice, L. 2014. Educational Software for First Order Logic Semantics in Introductory Logic Courses. *Information Systems Education Journal*, 12(6): 15.

Pygame Project. 2021. Pygame 2.1.2.

Rauber, M. F.; and Gresse von Wangenheim, C. 2022. Assessing the Learning of Machine Learning in K-12: A Ten-Year Systematic Mapping. *Informatics in Education*.

Reeves, S. 1996. A Teaching and Support Tool for Building Formal Models of Graphical User-Interfaces. In *Proceedings 1996 International Conference Software Engineering: Education and Practice*, 98–105. IEEE.

Resnick, M.; Maloney, J.; Monroy-Hernández, A.; Rusk, N.; Eastmond, E.; Brennan, K.; Millner, A.; Rosenbaum, E.; Silver, J.; Silverman, B.; and Kafai, Y. 2009. Scratch: Programming for All. *Communications of The Acm*, 52(11): 60–67.