Logical English for Law and Education

Robert Kowalski[1], Jacinto Davila[2], Galileo Sartor[3] and Miguel Calejo[4]

[1] Imperial College London, [2]Contratos Lógicos. C.A. and Universidad de Los Andes,
[3]University of Turin, [4]logicalcontracts.com, Lisbon

In this paper we summarize the key features of Logical English as syntactic sugar for logic programming languages such as pure Prolog, ASP and s(CASP); and we highlight two application areas, coding legal rules, and teaching logic as a computer language for children.

Logical English (LE) [4, 6, 7, 8] exploits the unique feature of Prolog-like logic programming (LP), that LP is the only programming paradigm based on the use of logic for human thinking and communication. By exploiting this feature, LE becomes a general-purpose programming language, which can be understood with only a reading knowledge of English and without any technical training in computing, mathematics, or logic.

LE is not only a Turing-complete computer programming language. It has the potential to represent and reason with a broad range of human knowledge, as shown by its ability to codify the language of law [7, 8, 13]. In an educational setting, it can be used to introduce both computational and logical thinking to children across the whole range of subjects taught in school, bridging STEM and non-STEM subjects alike.

Consider the following example:

Ordinary English:	All meetings with unvaccinated people are prohibited unless they are excused.
Logical English:	*A meeting is prohibited if a person attends the meeting* *and the person is unvaccinated* *and it is not the case that the meeting is excused.*
Prolog:	is_prohibited(A) :- attends(B, A), is_unvaccinated(B), not is_excused(A).

The example [9] illustrates some of the following characteristics of LE:

- LE avoids pronouns, which are a major source of ambiguity, as in the case of “they”, which in this example could refer either to meetings or to people.
- LE represents variables by common nouns prefixed by a determiner such as “a”, “an” or “the”. The indefinite determiner, “a” or “an”, introduces the first occurrence of a variable in a rule. The definite determiner, “the” prefixes all later occurrences of the same variable in the same rule. As in Prolog (with some exceptions noted in the full paper), all variables are implicitly universally quantified with scope being the rule in which they occur. This means that variables in different rules have no relationship with one another.
- Sentences in LE are either facts, or rules, as in Prolog. Rules have the Prolog-like conditional form *conclusion if conditions*, where the *conclusion* is an atomic sentence and the *conditions* are a combination of atomic sentences, typically connected by *and*. But *conditions* can also be connected by *or* and negation written in the form *it is not the case*

that. The relative precedence of the logical connectives is indicated by indentation (not illustrated in this example).

- As a matter of style and in the interests of greater precision, common nouns are preferably expressed in the singular, and verbs are expressed in the present tense. The temporal relationship between events and time-varying facts can be expressed, if necessary, by referring to time explicitly.

LE inherits the feature of Prolog that propositions can occur as arguments of higher-order or meta-level predicates. LE uses this to represent deontic modalities (obligation, prohibition, permission) and other propositional attitudes (notification, belief, desire, dislike). For example, here the keyword *that* introduces the proposition *a meeting is prohibited at a time T1* as an argument of the meta-predicate *the person is notified*:

a person violates the rules at a time T2
if the person is notified that a meeting is prohibited at a time T1
and the person attends the meeting at T2
and T1 is before or at the same time as T2.

As this rule also shows, a variable can be given a symbolic name.

Atomic sentences, which are facts, the conclusions of rules, or constituents of the conditions of rules, are instances of predicates declared by means of templates, such as:

a person* violates the rules at *a time,
a person* is notified that * message,
an eventuality* is prohibited at *a time.

where the asterisks identify the arguments of the predicates.

We have used an implementation of LE in SWISH [15] to represent a wide range of legal texts, helping to identify ambiguities, to explore modifications and alternative representations of the same text, and to compare the logical consequences of the alternatives. The SWISH implementation can compile LE into both Prolog and s(CASP) [13] and can provide explanations in LE for queries applied to alternative scenarios. We are also developing analogues of LE for other natural languages, such as Spanish and Italian.

Our experience with using LE for many practical, proof-of-concept applications suggests that LE has many valuable uses that do not require users to write programs. It also suggests that LE can be used to teach logic and computing to children of all ages, by focusing on teaching them to read, understand, modify, and use examples written in LE, without necessarily teaching them to write programs themselves.

Writing documents in any language, no matter whether in a computer language or in a natural language, is far harder than reading and understanding documents that are already written in that language. Moreover, most people will never need to write programs in their adult life at all. Some people may need to read programs, to convince themselves that the programs meet their requirements. But many people may want to understand explanations for answers to problems posed as queries, and they may want to modify assumptions to obtain better outcomes for those problems.

LE can be used to teach logic and computing by means of engaging examples, ranging across the sciences, humanities, practical life and imaginary worlds. By focusing on reading rather than

on writing, features of Prolog that would ordinarily be too difficult could be included at the earliest stage of an introduction to LE/Prolog, using such imaginary examples as:

A dragon smokes if another dragon is a parent of the dragon
and the other dragon smokes.
A dragon is healthy if it is not the case that
the dragon smokes.
A dragon is happy if for all cases in which
the dragon is a parent of another dragon
it is the case that
the other dragon is healthy.

Here the first sentence uses recursion, the second uses negation as failure, and the third uses universal quantification, to achieve the same effect as iteration, while-loops or recursion in conventional programming languages

Of course, this style of English may not be very natural, but it is understandable with only a reading knowledge of English, and without any understanding of how a computer would use these sentences to solve such problems as:

Alice is a dragon. Bob is a dragon. Alice is a parent of Bob. Alice is green.
Which dragon is happy?

You might need to explain that the computer believes that a sentence is not true if it does not believe that the sentence is true. But once you have explained that, it should not be too hard to understand that Alice is a happy dragon,

You might find it harder to convince a reader that Bob is a happy dragon too. But at least it shows that Logic and Computing can be fun. It also shows that Computing can be introduced to children at an early age without having to use examples, such as controlling a robot or manipulating images on a screen, which can be implemented just as well, or maybe even better, in an imperative programming language.

But maybe not all education can be reduced to having fun, and some lessons may even require hard work. Perhaps the LE definition of subset [10] is such an example. Here is the definition in Logical Spanish [11]:

Un conjunto es un subconjunto de otro conjunto
si en cualquier caso en el que
una cosa pertenezca a el conjunto
es también el caso que
la cosa pertenece a el otro conjunto.

Arguably, these examples and many others, such as those developed by Richard Ennals [2] and Tom Conlin [1] in the 1980s and more recently by Yuanlin Zhang and his colleagues [12, 14], show that Prolog can be used to integrate Logic and Computing in all areas of the educational curriculum. We believe that the use of such examples with a natural language syntax and with an initial focus on reading rather than on writing will significantly lower the barrier for realising this potential.

References

1. Conlon, T., 1985. Learning micro-prolog. Addison-Wesley.
2. Ennals, R., 1983. *Beginning micro-PROLOG*. Ellis Horwood.
3. Kowalski, R. (1982) Logic as a Computer Language for Children, In Proceedings of European Conference on Artificial Intelligence, Orsay, France.
4. Kowalski R (1990) English as a logic programming language. New Generation Comput 8(2):91–9.
5. Kowalski R (1992) Legislation as Logic Programs In: Logic Programming in Action (eds. G. Comyn , N. E. Fuchs, M. J. Ratcliffe), Springer Verlag, 203-230.
6. Kowalski R (2020) Logical English, In Proceedings of Logic and Practice of Programming (LPOP).
7. Kowalski, R. and Datoo, A., 2022. Logical English meets legal English for swaps and derivatives. *Artificial Intelligence and Law*, *30*(2), pp.163-197.
8. Kowalski, B., Dávila, J., CA, C.L. and Calejo, M., Logical English for legal applications. XAIF, Virtual Workshop on Explainable AI in Finance (2021).
9. LE unvaccinated: https://le.logicalcontracts.com/p/unvaccinatedupdate.pl, last accessed 2022/09/06.
10. LE subset: https://le.logicalcontracts.com/p/subset%20with%20lists.pl, last accessed 2022/09/07.
11. LE conjunto: https://le.logicalcontracts.com/p/conjunto.pl, last accessed 2022/09/07.
12. Reyes, M., Perez, C., Upchurch, R., Yuen, T. and Zhang, Y., 2016, March. Using declarative programming in an introductory computer science course for high school students. In *Thirtieth AAAI Conference on Artificial Intelligence*.
13. Sartor, G., Dávila, J., Billi, M., Contissa, G., Pisano, G. and Kowalski, R., 2022. Integration of Logical English and s(CASP), *2nd Workshop on Goal-directed Execution of Answer Set Programs (GDE'22), August 1, 2022.*
14. Yuen, T., Reyes, M. and Zhang, Y., 2017. Logic Programming for an Introductory Computer Science Course for High School Students. *arXiv preprint arXiv:1706.09248*.
15. Jan Wielemaker, Torbjörn Lager , and Fabrizio Riguzzi, 2015. SWISH: SWI-Prolog for sharing. arXiv preprint arXiv:1511.00915.