Logical English for Law and Education

Robert Kowalski[1], Jacinto Dávila[2], Galileo Sator[3] and Miguel Calejo[4]

[1] Imperial College London, [2] Universidad de Los Andes, Venezuela.
[3] University of Turin, [4] logicalcontracts.com, Lisbon

Abstract. In this paper we present 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.

Keywords: Logical English, Prolog, Law, Education

1 Introduction

Logical English (LE) [6-11, 15] 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 wide-spectrum computer 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. In an educational setting, it can be used to introduce both computational and logical thinking across the whole range of subjects taught in school, bridging STEM and non-STEM subjects alike.

Basic syntax. LE differs from pure Prolog primarily in the syntax for atomic predicates. In LE, predicates and their argument are declared by means of templates, as in:

a person likes *a thing*

where asterisks delimit the argument places of the predicate. In the simplest case, an argument place can be filled by a constant or a variable. For example:

Ordinary English: Alice likes anyone who likes logic.

Logical English: *Alice likes a person if the person likes logic.*

Prolog: likes(alice, A) :- likes(A, logic).

A *variable* is a noun phrase ending with a common noun, such as “person” or “thing” and starting with a determiner such as “a”, “an” or “the”. The indefinite determiner, “a” or “an”, introduces the first occurrence of a variable in a sentence. The same noun

phrase with the indefinite determiner replaced the definite determiner, “the”, represents all later occurrences of the same variable in the same sentence. Any other string of words in the position of an argument place is a constant. Unlike in Prolog, upper and lower case letters have no significance. Here is another example:

Templates:	*a person* is a parent of *a person*, *a person* is the mother of *a person*.
Logical English:	A person is a parent of an other person if the person is the mother of the other person.
Prolog:	is_a_parent_of(A, B) :- is_the_mother_of(A, B).

These examples illustrate some of the following characteristics of the basic syntax of LE, which are inherited from LP:

- Sentences in LE have the form of facts or rules. *Facts* are atomic sentences, whereas *rules* are sentences of the form *conclusion if conditions*, where the *conclusion* is an atomic sentence and the *conditions* are a combination of atomic sentences, typically connected by *and*.
- All variables in a sentence 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, even if they have the same name.
- The basic version of LE is untyped, like Prolog, and variable names are purely mnemonic. So, the first example sentence above has the same translation into Prolog as the meaningless sentence *Alice likes a hat if the hat likes logic.* We are developing an extended version of LE in which types are represented by common nouns, and the arguments of predicates are checked for compatibility with types that are declared in the templates.
- LE is designed so that sentences in LE have a unique translation into pure Prolog. But LE is also designed to be as unambiguous as possible, when understood by a human reader. For this purpose, LE deliberately eliminates the use of pronouns, which are a major source of ambiguity, as in the sentence *A person is a parent of an other person if she is the mother of her*.
- The current, basic syntax of LE does not include relative clauses, as in *Alice likes anyone who likes logic*. This is another deliberate choice, because relative clauses are another source of ambiguity. For example, the relative clause *which breathe fire* is ambiguous in the sentence *All dragons which breathe fire are scary*. The relative clause can be understood restrictively as meaning that *a dragon is scary if the dragon breathes fire*. Or it can be understood non-restrictively, as meaning that, not only are all dragons scary, but they also breathe fire.

  Logically, restrictive relative clauses add extra conditions to a sentence, whereas non-restrictive relative clauses add extra conclusions to the sentence. There are syntactic conventions for distinguishing between restrictive and

non-restrictive relative clauses (such as the use of commas), but not everyone uses them correctly and consistently, and they differ between American and British English.

```
18  A meeting is prohibited
19      if a person attends the meeting
20      and the person is unvaccinated
21      and it is not the case that
22          the meeting is excused.
23
24  A person has an obligation that the person pays £100
25      if the person attends a meeting
26      and the meeting is prohibited
27       and the person is notified that the meeting is prohibited.
28
29  An arrest warrant is issued for a person
30      if the person has an obligation that the person pays an amount
31      and it is not the case that
32          the person pays the amount.
33
34  scenario one is:
35      Boris attends christmas party.
36      Novak attends christmas party.
37      Novak is unvaccinated.
38      Novak is notified that christmas party is prohibited.
39      Novak pays £100.
40
41  scenario two is:
42      Boris attends christmas party.
43      Novak attends christmas party.
44      Novak is unvaccinated.
45      Boris is notified that christmas party is prohibited.
46      Novak pays £100.
47      Boris pays £1000.
48
49  query one is:
50      which person has an obligation that which eventuality.
51
52  query two is:
53      An arrest warrant is issued for which person.
54
```

Fig. 1. An LE program together with alternative scenarios and queries, displayed in a VS Code editor. The editor provides syntax highlighting, predictive text, and a simple form of static type checking. https://le.logicalcontracts.com/p/unvaccinated.pl.

2 The SWISH implementation of LE

The current implementation of LE in SWISH [20] translates LE programs and queries into Prolog or s(CASP) [15]. The implementation uses Prolog or s(CASP) to answer queries, and it translates answers and explanations into LE syntax. Figure 1 displays an example in a VS Code editor. The example illustrates some important additional features, which are not in the basic syntax of LE.

Negation as failure for rules and exceptions. LE uses negation as failure to cater for exceptions, as in the negative condition on lines 21 and 22 of figure 1. In contrast, ordinary natural language often omits such explicit negative conditions for exceptions, and relies instead on stating separately that the conclusion of a rule does not apply, as in Proleg [16, 17]. For example, instead of stating that *a meeting is excused if the meeting is a meeting of the cabinet ministers*, it may be more natural to state that *it is not the case that a meeting is prohibited if the meeting is a meeting of the cabinet ministers*. We are exploring the possibility of extending LE, to include such a treatment of exceptions for legal applications.

Metapredicates for propositional attitudes. LE inherits the feature of Prolog that sentences can occur as arguments of meta-predicates. LE uses this to represent deontic modalities (obligation, prohibition, permission) and other propositional attitudes (notification, belief, desire, dislike), introduced by the keyword *that*. For example, in line 24 of figure 1 the keyword *that* introduces the proposition *the person pays £100* as an argument of the meta-predicate *A person has an obligation.* Similarly, the keyword *that* in line 27 introduces the proposition *the meeting is prohibited* as an argument of the meta-predicate *the person is notified*.

The implementation of LE translates the sentence on lines 24-27 into the Prolog rule:

```
has_an_obligation_that(A, pays(A, '£_100')) :-
        attends(A, B), is prohibited(B),  is_notified_that(A, is_prohibited(B)).
```

It translates the sentence on lines 29-32 into the Prolog rule:

```
'An_arrest_warrant_is_issued_for'(A) :-
        has_an_obligation_that(A, pays(A, B)),  not pays(A, B).
```

Notice that the sentence expresses the deontic character of an obligation by representing the less-than-ideal consequence of violating the obligation.

Scenarios and queries. Figure 1 also includes a number of scenarios and queries, which can be combined and posed to the system, as shown in figure 2.

In the combination of query one and scenario one, Novak is obligated to pay £100, but Boris is not, because, although both have attended a prohibited party (thanks to Novak), only Novak has been notified of the prohibition.

In the combination of query one and scenario two, Boris is obligated to pay £100, but Novak is not, because this time it is Boris, rather than Novak, who is notified of the prohibition.

In the combination of query two with scenario one, no arrest warrant is issued, because Novak, the only person obligated to pay £100, pays the required amount.

In the combination of query two with scenario two, Boris is issued an arrest warrant, because he pays an incorrect amount. An explanation for issuing the arrest warrant to Boris is displayed in figure 3.

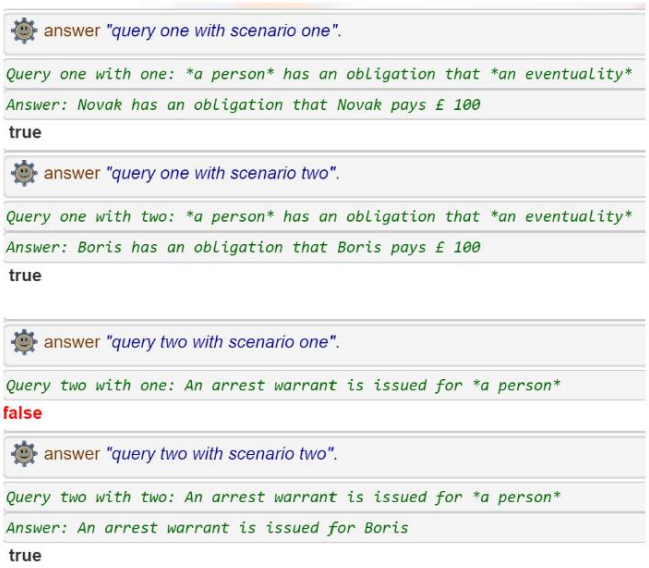

Fig. 2. A log of combined queries and scenarios together with their answers.

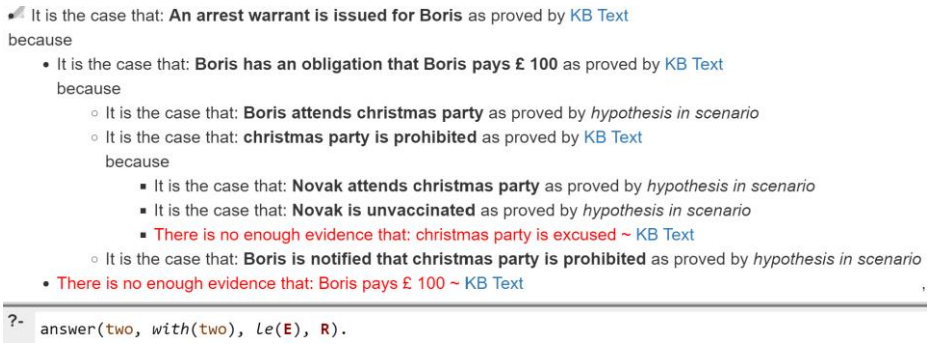

Fig. 3. An explanation for the answer to query two with scenario two.

3 Logical English for Legal Applications

We have been using LE to explore the representation of a wide range of legal texts, helping to identify ambiguities, explore alternative representations of the same text, and compare the logical consequences of the alternatives. The texts include portions of loan agreements, accountancy law, Italian citizenship, EU law on criminal rights, International Swaps and Derivative contracts, and insurance contracts.

The Italian Citizenship Example. We are also developing analogues of LE for other natural languages, such as Italian and Spanish. Figure 4 shows both an LE and a Logical Italian (LI) representation of Article 1 of Act No. 91 of 5 February 1992:

> E’ cittadino per nascita: a) il figlio di padre o di madre cittadini; b) chi e’ nato nel territorio della Repubblica se entrambi i genitori sono ignoti o apolidi, ovvero se il figlio non segue la cittadinanza dei genitori secondo la legge dello Stato al quale questi appartengono.

Both representations in figure 4 were generated manually. In contrast with the manually generated LE representation in figure 4, google translate gives the following translation of the original Italian text into English:

> Citizen by birth: a) the child of a citizen father or mother; b) who was born in the territory of the Republic if both parents are unknown or stateless, or if the child does not follow the citizenship of the parents according to the law of the state to which these belong.

Both the Italian text and its English translation are ambiguous: In particular, both the English condition "the child does not follow the citizenship of the parents according to the law of the state to which these belong" and its Italian counterpart, taken literally, seem to cover only the case where both parents have the same citizenship. Moreover, both the Italian "ovvero se" and the corresponding English "or if" seem to relate to a separate alternative from the alternatives that precede it. These readings of the natural language texts leave uncovered such deserving cases as the child having one parent who is stateless or unknown, and another parent who cannot pass on its citizenship to its child. It seems doubtful that these omissions would have been intended by the law.

The LE and LI representations in figure 4 incorporate only one interpretation of Article 1.1. Of course, other interpretations are possible, and they could also be represented in LE and LI. For comparison, see the similar case of children found abandoned in the UK, covered by the 1981 British Nationality Act, as formulated both in the original English and in an earlier, unimplemented variant of LE [7].

```
the knowledge base italian_citizen_new includes:

a person A is an italian citizen
if the person A is an italian citizen by birth.

a person A is an italian citizen by birth
if a person B is the parent of A
and B is an italian citizen.

a person A is the parent of a person B
if A is the father of B
    or A is the mother of B.

a person A is an italian citizen by birth
if A is born in italy
and for all cases in which
    a person B is the parent of A
    it is the case that
    B is stateless
        or B is unknown
        or A does not inherit the citizenship of B.
```

```
la base di conoscenza cittadinanza_italiana include:

una persona A ha la cittadinanza italiana
se A ha la cittadinanza italiana per nascita.

una persona A ha la cittadinanza italiana per nascita
se una persona B è genitore di A
e B ha la cittadinanza italiana.

una persona A è genitore di una persona B
se A è madre di B
    o A è padre di B.

una persona A ha la cittadinanza italiana
se A è nato in italia
e per tutti i casi in cui
    una persona B è genitore di A
    è provato che
    B è sconosciuto/a
        o B è apolide
        o A non segue la cittadinanza di B.
```

Fig. 4. LE and LI representations of Article 1 of Act No. 91 of 5 February 1992.
https://le.logicalcontracts.com/p/italian_citizen_new.pl
https://le.logicalcontracts.com/p/cittadinanza_italiana.pl

Figure 4 illustrates several features of LE that were not demonstrated earlier:

- LE uses indentation, rather than brackets, to represent the relative strength of binding of the logical connectives *and* and *or*.
- Variables can be given symbolic names, such as A and B in this example.
- Conditions can have the form *for all cases in which conditions it is the case that conclusion*, which are translated into forall(conditions, conclusion).

In figure 4, the possibility that a parent is unknown is expressed positively (as a kind of "strong" negation), to reflect the wording of the original legal text. Alternatively, the same possibility could be expressed using negation as failure, to conclude that a parent of a person is unknown if there is no information about the parent. In fact, with the representation in figure 4, it is possible to know that a person is born in Italy, but not to know who the parents are. In such a case, the *for-all* condition would be satisfied vacuously, and the person would be an Italian citizen by default.

4 Logical English for Education

By eliminating ambiguity from natural language, LE forces a writer to think more clearly about the relationship between sentences and their meanings. Thinking about meaning is unavoidable when writing sentences for translation into computer-executable code. But it also helps to avoid misunderstandings in communication among humans. Moreover, it helps to bridge the gap between the sciences and the humanities, by showing that clarity of language and thought is important in all academic disciplines.

The Italian citizenship example shows in a simple case how the use of symbolic names, which is associated with STEM disciplines, can be used to improve the clarity of communication in a non-STEM area. But the logical use of natural language, associated with LE and with some non-STEM disciplines, is also an important skill for use in STEM subjects, to make technical information more accessible to a wider audience.

The definition of subset. Figure 5 shows both an LE and an LS (Logical Spanish) representation of the definition of subset. Arguably, the definition can be understood by a reader without any training in mathematics or logic, but with only a reading knowledge of English or Spanish. Figure 6 shows all answers to the LE query *which set is a subset of which other set*, first with the scenario named *facts*, and then with the scenario named *lists*.

The subset example illustrates several features that have not been seen earlier:

- Because in the current version of LE variable names are purely mnemonic, the conditions that A and B are sets, on lines 13 and 14, need to be stated explicitly. These conditions would not be necessary if common nouns were treated as types. We plan to extend LE to include such types in the near future.
- The notion of set in lines 12-18 is an abstract notion, which is neutral with respect to how sets are represented concretely. Scenarios one and two employ different concrete representations. Scenario *sets* represents sets by facts that

define the *belongs to* relation explicitly. Scenario *lists* represents sets by Prolog-style lists, and the *belongs to* relation is defined in terms of the *is in* relation, which is LE syntax for the Prolog *member* predicate. In both scenarios, there are only two sets. In both scenarios, there is no empty set.

```
a set A is a subset of a set B
    if A is a set
    and B is a set
    and  for all cases in which
    a thing belongs to A
    it is the case that
    the thing belongs to B.

scenario facts is:
    family one is a set.
    family two is a set.

    Bob belongs to family one.
    Alice belongs to family one.

    Alice belongs to family two.

query subset is:
    which set is a subset
    of which other set.

scenario lists is:
    [Alice, Bob] is a set.
    [Alice] is a set.

    a thing belongs to a set
        if the thing is in the set.
```

```
un conjunto A es un subconjunto de un conjunto B
    si el conjunto A es un conjunto
    y el conjunto B es un conjunto
    y en todos los casos en los que
    una cosa pertenece a el conjunto A
    es el caso que
    la cosa pertenece a el conjunto B.

escenario hechos es:
    la familia uno es un conjunto.
    la familia dos es un conjunto.

    Roberto pertenece a la familia uno.
    Alicia pertenece a la familia uno.

    Alicia pertenece a la familia dos.

la pregunta subconjunto es:
    cuál conjunto es subconjunto
    de cuál otro conjunto.

escenario listas es:
    [Alicia, Roberto] es un conjunto.
    [Alicia] es un conjunto.

    una cosa pertenece a un conjunto
        si la cosa is in el conjunto.
```

Fig. 5. A definition of the subset relation in LE and LS. https://le.logicalcontracts.com/p/sets%20with%20lists.pl
https://le.logicalcontracts.com/p/conjunto.pl

```
Answer: family one is a subset of family one
true
Answer: family two is a subset of family one
true
Answer: family two is a subset of family two
true
?- answer subset with facts.
```

```
Answer: [Alice,Bob] is a subset of [Alice,Bob]
true
Answer: [Alice] is a subset of [Alice,Bob]
true
Answer: [Alice] is a subset of [Alice]
true
?- answer subset with lists.
```

Fig. 6. All subsets with sets represented by facts or by lists.

Reading versus writing. It is natural to associate teaching computer science with teaching students how to write computer programs. But this overlooks the fact that most people will never need to write computer programs in their adult life. Some people may want to read programs, to convince themselves that the programs meet their requirements. Some may want to understand explanations for answers to queries, and they may want to modify assumptions to obtain better answers. But hardly anyone will need to write programs themselves from scratch.

Focussing on teaching students how to write computer programs also overlooks the fact that learning to write well in any language, whether it be a natural language or a computer language, is much harder than learning to read. In this respect, LE has an advantage over other computer languages, because it can exploit a much wider range of examples requiring only a reading knowledge of natural language.

How to be a happy dragon. By focusing on reading rather than writing, examples of programming language constructs that would ordinarily be considered too difficult to teach at an introductory level can be included from the very beginning. Figure 7 illustrates such an example. Here the first sentence uses recursion, the second uses negation as failure, and the third uses universal quantification, achieving the same effect as iteration, while-loops or recursion in conventional programming languages.

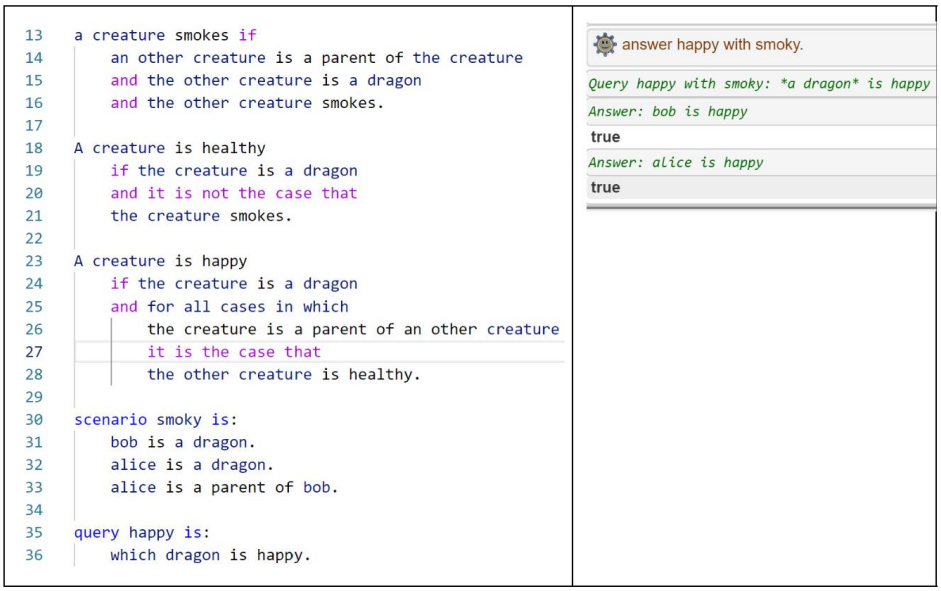

Fig. 7. An LE program for introducing young children to logic and computing. https://le.logicalcontracts.com/p/happy_dragon.pl

Although this style of English may seem artificial, it can be made more natural, while remaining unambiguous, by treating common nouns as types. For example, the sentence on lines 13-16 could be written more naturally and more simply as:

A dragon smokes if an other dragon is a parent of the dragon
and the other dragon smokes.

All the examples we have seen until now can be understood without any knowledge about how LE is executed. Moreover, that understanding can be enhanced by experimenting with different scenarios and queries, and by exploring the logical consequences. In this example, a student can learn that alice is happy, because her only child,

bob, is healthy; bob is healthy because he does not smoke; and bob does not smoke, because his parent alice does not smoke. It might be harder to convince a student that bob is a happy dragon too. But at least it shows that Logic and 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.

The Euclidean Algorithm. As a computer language, LE combines in one language the features of a programming language, database language, and knowledge representation and problem-solving language. All the examples we have seen so far are examples of its use for knowledge representation and problem solving. The representation in figure 8 of the Euclidean algorithm for computing the greatest common divisor (gcd) of two numbers illustrates its use for programming. It uses the built-in Prolog predicates for subtraction and for testing inequalities.

Notice that a query such as *which number is the gcd of 1946 and which other number* cannot be answered, because the Prolog predicate for inequality can be used only when the two numbers are both given as input. On the other hand, the same program can be used both to test that a given number is the gcd of two other given numbers, as well as to generate the gcd. This capability would need two separate programs in a conventional imperative programming language.

```
10  a number N is the gcd of  N and N.
11
12  a number D is the gcd of a number N and a number M
13      if N > M
14      and a number smallerN is N-M
15      and D is the gcd of smallerN and M.
16
17  a number D is the gcd of a number N and a number M
18      if M > N
19      and a number smallerM is M-N
20      and D is the gcd of N and smallerM.
```

Fig. 8. The Euclidean algorithm. https://le.logicalcontracts.com/p/Euclid.pl

On the other hand, the LE representation is not an algorithm. The Euclidean algorithm is the behaviour obtained by using the LE representation to reason top-down (or backward), as in Prolog. This behaviour can be described imperatively:

To find the gcd D of two given numbers N and M:
If N = M, then D = N.
If N > M, replace N by N-M, find the gcd D' of N-M and M, then D = D'.
If M > N, replace M by M-N, find the gcd D' of N and M-N, then D = D'.

One of the advantages of the declarative representation is that it is written in the same logical style as the natural definition (or specification) of gcd, illustrated in figure 9.

Compared with the imperative representation, the LE representation in figure 8 makes it much easier to reason that the Euclidean algorithm correctly computes the gcd. As David Warren points out [19], this can be done by using mathematical induction, exploiting the fact that the bottom-up (inductive) interpretation of the program in figure 8 computes the same gcd relation as the top-down (algorithmic) interpretation.

Notice that the specification of gcd, illustrated in figure 9, is also executable, although it is much less efficient than the Euclidean algorithm.

```
12  a number D is the gcd of a number N and a number M
13      if D divides N
14      and D divides M
15      and for all cases in which
16          an other number divides N
17          and the other number divides M
18          it is the case that
19          the other number =< D.
20
21  a number D divides a number N
22      if  D is between 1 & N
23      and 0 is N mod D.
```

Fig. 9. The definition of gcd. https://le.logicalcontracts.com/p/gcd.pl

5 Related and future work

LE can be regarded as a controlled natural language, which is similar in spirit to ACE [3] and PENG [18], which are also implemented in Prolog. But, whereas LE is syntactic sugar for pure Prolog, ACE and PENG are syntactic sugar for first-order logic. PENG^{ASP} [4], on the other hand, which is syntactic sugar for ASP, is closer to LE, but also closer to natural English.

LE inherits the wide spectrum use of LP as a computer language for programming, program specification, databases and knowledge representation and reasoning. However, in its current form, it is not entirely general-purpose. It lacks the ability of imperative languages to represent an agent's goals and the ability of an agent to satisfy goals by executing actions in reaction to external events.

To remedy this disability, we developed the language LPS (Logic Production System) [12-14] as an extension of LP. In fact, the earliest implementation of LE was for a smart contract using the rock-paper-scissors game [2] written in LPS. We plan to extend LE to include the reactive rules and causal laws of LPS. Other proposed extensions include a more natural representation of rules and exceptions, following the approach of [16-17], as well as natural language analogues of object-oriented types and embedded functions and relations as in Ciao [1,5].

In the meanwhile, the current version of LE and its natural language cousins, such as LI and LS, indicate the future potential of logic-based computer languages with a natural language syntax. In this paper, we have highlighted legal applications and education as two major areas in which the benefits of such languages can be exploited already today.

References

1. Casas, A., Cabeza, D., Hermenegildo, M.V.: A syntactic approach to combining functional notation, lazy evaluation, and higher-order in LP systems. In: Functional and Logic Programming: 8th International Symposium, FLOPS 2006, pp. 146-162. Springer (2006).
2. Davila, J.: Rock-Paper-Scissors https://demo.logicalcontracts.com/p/rps-gets.pl (2017).
3. Fuchs, N.E., Schwitter, R.: Attempto controlled English (ACE). arXiv preprint cmp-lg/9603003 (1996).
4. Guy, S.C., Schwitter, R. The PENG^{ASP} system: architecture, language and authoring tool. *Lang Resources & Evaluation* 51, 67–92 (2017).
5. Hermenegildo, M., Morales, J., Lopez-Garcia P., Carro, M.: Types, modes and so much more – the Prolog way. In: Warren, D., Dahl, V., Eiter, T., Hermenegildo, M., Kowalski, R. and Rossi, F. (eds.) Prolog - The Next 50 Years. LNCS, vol. 13900. Springer (2023).
6. Kowalski, R.: English as a logic programming language. New Generation Computing 8(2). 91–93 (1990),
7. Kowalski, R.: Legislation as Logic Programs. In: G. Comyn , N. E. Fuchs, M. J. Ratcliffe (eds.), Logic Programming in Action, pp. 203-230, Springer (1992).
8. Kowalski, R.: Logical English, In Proceedings of Logic and Practice of Programming (LPOP) (2020).
9. Kowalski, R., Datoo, A.: Logical English meets legal English for swaps and derivatives. Artificial Intelligence and Law, 30(2), 163-197 (2022).
10. Kowalski, R., Dávila, J., Calejo, M., Logical English for legal applications. XAIF, Virtual Workshop on Explainable AI in Finance (2021).
11. Kowalski, R., Dávila, J., Sartor, G., Calejo, M.: Logical English for Law. In Proceedings of the Workshop on Methodologies for Translating Legal Norms into Formal Representations (LN2FR), JURIX, (2022).
12. Kowalski, R., Sadri, F.: Reactive Computing as Model Generation. New Generation Computing, 33(1), 33-67 (2015).
13. Kowalski, R., Sadri, F.: Programming in logic without logic programming. Theory and Practice of Logic Programming, 16(03), pp.269-295 (2016)
14. Kowalski, R., Sadri, F., Calejo, M., Dávila, J.: Combining Logic Programming and Imperative Programming in LPS. In: Warren, D., Dahl, V., Eiter, T., Hermenegildo, M., Kowalski, R., Rossi, F. (eds.) Prolog - The Next 50 Years. LNCS, vol. 13900. Springer, Heidelberg (2023).
15. Sartor, G., Dávila, J., Billi, M., Contissa, G., Pisano, G., Kowalski, R.: Integration of Logical English and s(CASP), 2nd Workshop on Goal-directed Execution of Answer Set Programs (GDE'22) (2022).
16. Satoh, K., Asai, K., Kogawa, T., Kubota, M., Nakamura, M., Nishigai, Y., Shirakawa, K., Takano, C., PROLEG: An implementation of the presupposed ultimate fact theory of Japanese civil code by PROLOG technology", LNCS, 6797, pp 153-164. Springer (2010).
17. Satoh, K.: PROLEG: Practical Legal Reasoning System, In: Warren, D., Dahl, V., Eiter, T., Hermenegildo, M., Kowalski, R., Rossi, F. (eds.) Prolog - The Next 50 Years. LNCS, vol. 13900. Springer, Heidelberg (2023).
18. Schwitter, R.: English as a formal specification language. In: Proceedings. 13th International Workshop on Database and Expert Systems Applications pp. 228-232, IEEE (2002).
19. Warren, D.S.: Writing Correct Prolog Programs. In: Warren, D., Dahl, V., Eiter, T., Hermenegildo, M., Kowalski, R., Rossi, F. (eds.) Prolog - The Next 50 Years. LNCS, vol. 13900. Springer, Heidelberg (2023).

20. Wielemaker, J., Schrijvers, T., Triska, M., Lager, T.: SWI-Prolog. Theory and Practice of Logic Programming 12(1-2), 67-96 (2012).