Logical English

A position paper prepared for Logic and Practice of Programming (LPOP) 2020

Robert Kowalski
Imperial College London
11 November 2020

Logical English (LE) is a controlled natural language, in which English sentences are translated into a program in a logic programming (LP) language, such as Prolog, Datalog or Answer Set Programming (ASP). Similar controlled English languages that are also executed by translation to LP include ACE [Fuchs and Schwitter, 1996; Fuchs et al, 2008; Fuchs, 2013], PENG [Schwitter, 2002] and PENG^{ASP} [Guy and Schwitter, 2017]. ACE and PENG are both intended for general-purpose knowledge representation and reasoning. In contrast, LE and PENG^{ASP} are syntactic sugar for logic programs. PENG^{ASP} provides syntactic sugar for ASP, and LE provides syntactic sugar for LPS [Kowalski and Sadri, 2015, 2016], which is an extension of pure Prolog, implemented in Prolog [Wielemaker et al, 2019].

The ultimate goal of LE is to serve as a general-purpose computer language, which can be understood by a reader without any training in computing, logic or mathematics. It is inspired in part by the language of law, which can be viewed as a programming language that is executed by humans rather than by computers [Kowalski, 1992]. LE can also be viewed as a domain-specific language for legal applications, similar to the English-like languages Blaux [Morris, 2020] and Lexon [Diedrich, 2020], both of which also have LP roots. Blaux is a combination of the LP language Flora-2 and the visual coding environment Blockly. Lexon on the other hand combines syntactic sugar for logic programs with higher-order logic, and compiles into Solidity, the programming language for the Ethereum blockchain.

LE is a declarative language, like the language of law. But LE also has an imperative character, inherited from LPS. Computation in both LE and LPS is similar to computation in a conventional imperative language, starting from an initial state, observing a potentially infinite stream of external events as input, and generating a potentially infinite stream of actions as output, while destructively updating a current state, with the goal of solving a problem or of simulating a real or imaginary world.

But unlike computation in conventional imperative languages, computation in LE also has a logical interpretation as generating a sequence of states and events, whose time-stamped history underlies a model that makes the program's goals true.

LE is a work in progress [Kowalski, 2019]. There have been three experimental implementations of variants of LE based on LPS or Prolog, focussed primarily on legal applications [Davila, 2017; Karadotchev, 2019; Fu, 2020]. The website http://demo.logicalcontracts.com/ contains an example of the rock-paper-scissors game in a form of Logical English in the Fintech submenu of the examples menu. It

also contains a number of LPS examples in the examples menu, which can be modified and executed online. Here are some examples based on this work:

(1) If a player P1 plays a choice C1 and another player P2 plays a choice C2
and C1 beats C2 and it is not the case that the game is over
then P1 receives the prize and it becomes the case that the game is over.

(2) A transaction is governed by IsdaAgreement
if a confirmation of the transaction states
that the transaction is governed by IsdaAgreement
and the transaction commences on a first day
and IsdaAgreement is dated as of a second day
and the first day is on or after the second day.

(3) It becomes the case that a requirement is defaulted on a day
when it is the end of the day
and the requirement is potentially defaulted
and the lender delivers a notice to the borrower on another day
and the notice is that the requirement is potentially defaulted
and the other day is 3 days before the day
and it is not the case that the requirement is cured.

The first example translates into a reactive rule in LPS, written in the form *if antecedent then consequent*. Reactive rules in LPS represent goals that are made true by making their *consequents* true whenever their *antecedents* become true. *Consequents* can be made true either deliberately by performing actions or fortuitously by observing external events. The symbols P1, P2, C1 and C2 name variables. Their optional use in LE is similar to their use in legal texts. They provide names for variables, which look mathematical, but can be understood without mathematical training.

The second and third examples translate into ordinary LP clauses, which have the form *conclusion if conditions*. LP clauses define the models that can make goals true. In (2) and (3), as more generally in ACE, PENG and LE, variables can be represented by common nouns (such as *transaction* or *notice*) preceded by an article (*a*, *an* or *the),* as in *a transaction* and *the transaction*. The articles *a* and *an* are used for the first occurrence of the variable, and *the* is used for later occurrences of the same variable. Other variables of the same type in the same sentence can be introduced by preceding them with such adjectives as *first*, *second*, or *another* and *the other*, etc.

In (2) the condition *a confirmation of the transaction states that the transaction is governed by IsdaAgreement* illustrates the embedding of an object-level sentence inside a higher-order or meta-level sentence. This embedding is represented in Prolog by translating the phrase *is governed by* both into a predicate symbol, which is "used" at the object-level in the conclusion of the sentence, and into a function symbol, which is "mentioned" at the meta-level in a condition of the sentence.

Sentence (3) translates into an LP clause representing a causal relationship between an event (*it is the end of a day*) and a fluent (*a requirement is defaulted on a day*), which is initiated by the event. In LPS, this relationship is implemented by adding the fluent to the current state if it is the end of a day and if the other conditions of the clause hold at the end of the day. The representation uses the ontology of the event calculus [Kowalski and Sergot,1986], but an implementation involving destructive updates of a single current state. The frame axiom of the event calculus is an emergent property, which is true in any model that satisfies the goals, but it is not used for reasoning. The example could be expressed equally well in PENG^{ASP}, which supports the writing of temporal specifications using an ASP-based adaptation of the event calculus.

To reduce ambiguity, LE has no pronouns, such as *he*, *she*, or *it*. To reduce the need for a dictionary, all nouns and verbs are expressed in the singular. The restriction to singular nouns means that LE does not use English quantifiers that require the use of plural nouns, such as *all* and *some*.

The use of articles in LE avoids the need for explicit quantification. In the case of a range-restricted LP clause (containing no variable in the conclusion that is not in the conditions), the natural reading of the articles in English conforms to the LP convention that all variables in the clause are universally quantified. But in the case of a non-range-restricted clause, the natural reading is that any variable in the conclusion that is not in the conditions is existentially quantified, For example:

(4) An event of a person acquiring citizenship of the land of oz occurs on a day
if the person is born in a place on the day and the place is in the land of oz.

Here the natural reading is that all variables are universally quantified except for the variable *an event*, which is existentially quantified. Moreover, although the scope of the universally quantified variables is limited to the clause, the existentially quantified variable has wider scope, as in the added clause:

(5) A person celebrates the event if the person lives in the land of oz.

These readings of the English article are compatible with the interpretation of implicit quantifiers in existential (or ∀∃) rules, and with the elimination of existential quantifiers by skolemization [Baget et al, 2011].

All of the examples above are written in a basic form of LE, which is syntactic sugar for LPS. The plan is to develop LE as a series of extensions, starting from this basic form, introducing increasingly more natural syntaxes, while avoiding the introduction of ambiguity. For example, the LE sentence (2) above could be written in an extended form of LE as:

(6) A transaction is governed by IsdaAgreement
if the confirmation of the transaction states
that the transaction is governed by IsdaAgreement
and the transaction commences on a day that is on or after the day as of
which IsdaAgreement is dated.

Here *a confirmation* is replaced by *the confirmation* to indicate that the relation between the confirmation and the transaction is "functional", in the sense that there is only one confirmation for each transaction. The relative pronoun *that*, as well as the preposition *as of* followed by *which*, introduces a restrictive relative clause, which inserts a logical condition into the text of another logical expression.

LE and its relationship with other logics and other computer languages

LE and its logical underpinning LPS are based on a more general logic for abductive logic programming (ALP) [Kakas et al, 1992; Kowalski, 2011] in which logic programs are extended with abducible predicates (generalising actions in LPS) and with goals in first-order logic (generalising reactive rules and constraints). As in LPS, goals in this ALP logic are made true by a model determined by the logic program extended by facts expressed in the vocabulary of the abducible predicates.

In this ALP logic, goals of the form *if antecedent then consequent* are material implications, which can be satisfied *preventively* by making the *antecedent* false, or proactively by making the *consequent* true whether or not the *antecedent* ever becomes true. They can also be satisfied while performing unnecessary and irrelevant actions. However, in LPS and LE, goals of this form are reactive rules, which can be solved only *reactively* and *relevantly*, by generating actions to make *consequents* true whenever *antecedents* become true.

LPS is scaled down from ALP, losing some of the power of a problem-solving language, to compete more effectively with conventional computer languages for efficiency. However, the LE syntax for LPS introduces language features that are absent from both ALP and LPS, but which have been found to be useful in other computer languages. For example, even in the basic form of LE, the use of common nouns provides some of the features of a typed, object-oriented language. Other proposed extensions of LE provide some of the features of a functional language, as in *Monday is the day before the day before Wednesday*, which compiles into the LP relational form *Monday is the day before another day and the other day is before Wednesday*. The inclusion in LE of these and other features suggests that LE has the potential to compete with conventional computer languages not only for efficiency, but also for expressive power.

But no matter how LE compares with other computer languages today, there is no need for the computer languages of the future to employ such machine-oriented features as the use of variables to name computer memory locations and the use of assignment statements to manipulate the contents of computer memory. Nor is there any need for them to employ complex symbolic syntax in situations where natural language syntax can be used just as effectively instead.

We need to follow the lead of legal scholars who campaigning against legalese that can be understood only by legal professionals, and who are advocating plain language that can be understood by ordinary people [Williams, 2004]. In the world of computing, we need to move away from languages that make people think like

machines, and employ languages that make computers think more like people. Arguably, Logical English and its controlled natural language cousins provide a path, which is both logical and natural, for helping to reach this goal.

References

Baget, J.F., Leclère, M., Mugnier, M.L. and Salvat, E., 2011. On rules with existential variables: Walking the decidability line. *Artificial Intelligence*, *175*(9-10), pp.1620-1654.

Davila, J. 2017. Rock, Paper, Scissors. http://demo.logicalcontracts.com/example/RockPaperScissorsBaseEN.pl

Diedrich, H. 2020. Lexon Bible: Hitchhiker's Guide to Digital Contracts. Wildfire Publishing.

Fu, Z, 2020. Logical English (LE) for representing legal documents, MSc thesis. Imperial College London.

Fuchs, N.E. 2013, Attempto Project. http://attempto.ifi.uzh.ch/site/

Fuchs, N.E. and Schwitter, R., 1996. Attempto controlled english (ace). arXiv preprint cmp-lg/9603003.

Fuchs, N.E., Kaljurand, K. and Kuhn, T., 2008. Attempto controlled english for knowledge representation. In *Reasoning Web* (pp. 104-124). Springer, Berlin, Heidelberg.

Guy, S.C. and Schwitter, R., 2017. The PENG^{ASP} system: architecture, language and authoring tool. Language Resources and Evaluation, 51(1), pp.67-92.

Kakas, A.C., Kowalski, R.A. and Toni, F., 1992. Abductive logic programming. *Journal of logic and computation*, *2*(6), pp.719-770.

Karadotchev, V., 2019. First Steps Towards Logical English. MSc thesis. Imperial College London.

Kowalski, R., 2011. *Computational logic and human thinking: how to be artificially intelligent*. Cambridge University Press.

Kowalski, R., 1992. Legislation as Logic Programs In: Logic Programming in Action (eds. G. Comyn , N. E. Fuchs, M. J. Ratcliffe), Springer Verlag, pages 203-230.

Kowalski, R. 2019. Logical English slides. http://www.doc.ic.ac.uk/~rak/papers/Logical%20English.pdf

Kowalski, R. and Sadri, F., 2015. "Reactive Computing as Model Generation." New Generation Computing,, Volume 33, Issue 1, pp 33-67

Kowalski, R. and Sadri, F., 2016. Programming in logic without logic programming. Theory and Practice of Logic Programming, 16(03), pp.269-295.

Kowalski, R. and Sergot, M., 1986, "A Logic-based Calculus of Events", in New Generation Computing, Vol. 4, No.1, February pp. 67-95.

Morris, J., 2020. Blawx Alpha: User Friendly Rules as Code on the Web. https://www.blawx.com/

Schwitter, R., 2002, English as a formal specification language. In Proceedings. 13th International Workshop on Database and Expert Systems Applications (pp. 228-232). IEEE.

Schwitter, R., 2019. Augmenting an answer set based controlled natural language with temporal expressions. In PRICAI 2019: Trends in Artificial Intelligence, vol. 11670 of LNAI, 500–513. Springer.

Wielemaker, J., Riguzzi, F., Kowalski, R. A., Lager, T., Sadri, F., & Calejo, M. 2019. "Using SWISH to realize interactive web-based tutorials for logic-based languages", Theory and Practice of Logic Programming, 19(2), 229-261.

Williams, C., 2004. Legal English and plain language: An introduction. *ESP across Cultures*, *1*(1), pp.111-124.