Logical English as an Executable Computer Language

Robert Kowalski
Imperial College London
Together with

Fariba Sadri
Miguel Calejo
Jacinto Davila
Vesko Karadotchev
Conclusion

Who would we expect to be the best person to write a computer program? A trained programmer, most of us would reply. Perhaps an unusually inventive end-user, given access to tools such as code generators, and fourth generation languages. But a lawyer ...

Well, according to Bob Kowalski, a professor of computational logic at Imperial College London, lawyers are just the people. Kowalski, you may recall, devised the principles of logic programming, on which the PROLOG language. No matter that it may be a rather specialised and stilted form of English, it is still much more accessible to the average human than any mathematical formalism.

An example, according to Kowalski, is the emergency notice found in London Underground trains:

"Press the alarm signal to alert the driver. The driver will stop immediately if any part of the train is in a station. If not, the train will proceed"
Logical English (LE)

• modelled on the language of law.

• understandable by English speakers without special training.

• suitable for general computation, i.e. programming, databases and knowledge representation.

• prototypes in Prolog
The Language of Law

- declarative
- describes the legal consequences of actions and events
- logical conditionals and logical all/every/exists
- avoids ambiguity, but embraces vagueness (open texture)
- passively judges the behaviour of external agents

The majority of computer languages

- imperative
- assignment statements
- if-then-else
- iteration and loops
- unambiguous by design
- actively support the goals of external agents

Smart contracts
activey support the goals of legal agreements.
Examples from

- The British Nationality Act
  logic programs.

- A simplified loan agreement
  deterministic finite automata.

- Rock, Paper, Scissors
  (Lessons and insights from a cryptocurrency lab)
  imperatives.

- ISDA Master Agreement
  (International Swaps and Derivatives Association)
  obligations.
ARTICLES

THE BRITISH NATIONALITY ACT AS A LOGIC PROGRAM

The formalization of legislation and the development of computer systems to assist with legal problem solving provide a rich domain for developing and testing artificial-intelligence technology.

M. J. SERGOT, F. SADRI, R. A. KOWALSKI, F. KRIWACZEK, P. HAMMOND, and H. T. CORY
1.-(1) A person born in the United Kingdom after commencement shall be a British citizen if at the time of the birth his father or mother is (a) a British citizen; or (b) settled in the United Kingdom.
1.(1) A person born in the United Kingdom after commencement shall be a British citizen if at the time of the birth his father or mother is (a) a British citizen; or (b) settled in the United Kingdom.

A person acquires British citizenship by subsection 1.1 at a time if the person is born in the UK at the time and the time is after commencement and another person is the father of the person or another person is the mother of the person and the other person is a British citizen at the time or the other person is settled in the UK at the time.

“a”, “an” or “another” indicates the first occurrence of a logical variable “the” indicates a later occurrence of the same logical variable in the same sentence. No “all”, “every” or “some”.
All payments that are specified in some transaction are obligatory.

needs to be rewritten.

For example:

A payment is obligatory
if the payment is specified in a transaction.

Or:

It is an obligation that a payment is made
if the payment is specified in a transaction.
Examples from

- The British Nationality Act logic programs.

- A simplified loan agreement deterministic finite automata.

- Rock, Paper, Scissors (Lessons and insights from a cryptocurrency lab) imperatives.

- ISDA Master Agreement (International Swaps and Derivatives Association) obligations.
Contract as Automaton: The Computational Representation of Financial Agreements

Mark D. Flood
Office of Financial Research
mark.flood@ofr.treasury.gov

Oliver R. Goodenough
Office of Financial Research and Vermont Law School
oliver.goodenough@ofr.treasury.gov
ogoodenough@vermontlaw.edu
Agreement

This loan agreement dated June 1, 2014, by and between Lender Bank Co. ("Lender") and Borrower Corp. (Borrower), will set out the terms under which Lender will extend credit in the principal amount of $1,000 to Borrower with an un-compounded interest rate of 5% per annum, included in the specified payment structure.

1. The Loan
At the request of Borrower, to be given on June 1, 2014, Lender will advance $1,000 to Borrower no later than June 2, 2014. If Borrower does not make such a request, this agreement will terminate.

2. Repayment
Subject to the other terms of this agreement, Borrower will repay the loan in the following payments:
   (a) Payment 1, due June 1, 2015, in the amount of $550, representing a payment of $500 as half of the principal and interest in the amount of $50.
   (b) Payment 2, due June 1, 2016, in the amount of $525, representing a payment of $500 as the remaining half of the principal and interest in the amount of $25.

3. Representations and Warranties
The Borrower represents and warrants, at the execution of this agreement, at the request for the advance of funds and at all times any repayment amount shall be outstanding, the Borrower’s assets shall exceed its liabilities as determined under an application of the FASB rules of accounting.

4. Covenants:
The Borrower covenants that at the execution of this agreement, at the request for the advance of funds and at all times any repayment amount shall be outstanding it will make timely payment of all state and federal taxes as and when due.

5. Events of Default
The Borrower will be in default under this agreement upon the occurrence of any of the following events or conditions, provided they shall remain uncured within a period of two days after notice is given to Borrower by Lender of their occurrence (such an uncured event an "Event of Default"):
   (a) Borrower shall fail to make timely payment of any amount due to Lender hereunder;
   (b) Any of the representation or warranties of Borrower under this agreement shall prove untrue;
   (c) Borrower shall fail to perform any of its covenants under this agreement;
   (d) Borrower shall file for bankruptcy or insolvency under any applicable federal or state law.

A default will be cured by the Borrower (i) remedying the potential event of default and (ii) giving effective notice of such remedy to the Lender. In the event of multiple events of default,
Agreement

This loan agreement dated June 1, 2014, by and between Lender Bank Co. ("Lender") and Borrower Corp. (Borrower), will set out the terms under which Lender will extend credit in the principal amount of $1,000 to Borrower with an un-compounded interest rate of 5% per annum, included in the specified payment structure.

1. The Loan:

At the request of Borrower, to be given on June 1, 2014, Lender will advance $1,000 to Borrower no later than June 2, 2014. If Borrower does not make such a request, this agreement will terminate.
It is an obligation that the lender advances $1000 to the borrower at a time and the time is before the end of 2014/6/2 if the borrower has requested $1000 on 2014/6/1.

Or

The lender becomes liable to litigation at a time if it is the end of 2014/6/2 at the time and the borrower has requested $1000 on 2014/6/1 and it is not the case that the lender has advanced $1000 at another time and the other time is before the end of 2014/6/2.
1. The Loan:

At the request of Borrower, to be given on June 1, 2014, Lender will advance $1000 to Borrower no later than June 2, 2014. If Borrower does not make such a request, this agreement will terminate.

The contract becomes terminated when it is the end of 2014/6/1 and it is not the case that the borrower has requested the loan.

The contract becomes terminated at a time if it is the end of 2014/6/1 at the time and it is not the case that the borrower has requested the loan at the time.

“conclusion when conditions” means “conclusion at a time if conditions at the time”

translated into Prolog/LPS (Logical Production System)

\[ \text{end}_{-}\text{of}_{-}\text{day}(2014/6/1) \Rightarrow \text{initiates terminated if not requested(borrower, 1000, 2014/6/1).} \]

```
1 end_of_day(2014/6/2)
2 initiates liable_to_litigation(lender)
3 if requested(borrower, 1000, 2014/6/1),
4 not advanced(lender, 1000).
5
6 end_of_day(2014/6/1)
7 initiates terminated
8 if not requested(borrower, 1000, 2014/6/1).
```
The history of states and events can be visualised as a state transition diagram (or deterministic finite automaton)

1. `observe request(borrower, 1000) at '2014-06-01T15:00'. % at 15:00.
2. `observe advance(lender, 1000) at '2014-06-02T18:00'.
3. `observe pay(borrower, lender, 550) at '2015-06-01T12:00'.
4. `observe pay(borrower, lender, 525) at '2016-06-01T06:00'.
Examples from

- The British Nationality Act logic programs.
- A simplified loan agreement deterministic finite automata.
- Rock, Paper, Scissors (Lessons and insights from a cryptocurrency lab) imperatives.
- ISDA Master Agreement (International Swaps and Derivatives Association) obligations.
Each player simultaneously forms one of three shapes with an outstretched hand:
Rock-paper-scissors in an “early” variant of Logical English

When a player pays a prize
then the reward that is a number becomes the number minus the prize.

If a first player has played a first choice at a first time
and a second player has played a second choice at the first time
and the first player is different from the second player
and the first choice beats the second choice
and it is not the case that the game is over at the first time
then initiate the game is over from the first time to a fourth time
and the reward is a prize at the first time
and the first player pays the prize from the first time to a second time.

If a first player has played a first choice at a first time
and a second player has played a second choice at the first time
and the first player is different from the second player
and it is not the case that the first choice beats the second choice
and it is not the case that the second choice beats the first choice
and it is not the case that the game is over at the first time
then initiate the game is over from the first time to a fourth time
and the reward is a prize at the first time
and a number is half the prize
and the first player pays the number from the first time to a second time
and the second player pays the number from the first time to the second time.
Rock-paper-scissors in LPS on the Ethereum blockchain

```prolog
beats(scissors, paper).
beats(paper, rock).
beats(rock, scissors).

prolog_events e_transaction(latest, _From, _Input, _Wei, _To). % Generate events from the blockchain

e_transaction(latest, From, Input, Wei, To) initiates played(From, Input, Wei) if
  lps_my_account(To), Wei>0, not played(From, _, _).

fluents played(_Player, _Choice, _Value), gameOver.

reward(R) at T if
  balance(V) at T,
  R is round(V*0.9). % keep 10% :-)

balance(B) at T if
  findall(V, played(_, _, V) at T, L), sum_list(L, B).

num_players(N) at T if
  findall(P, played(P, _) at T, L), length(L, N).

false num_players(N), N>2.

pay(Player, Prize) from T1 to T3 if % plan / macro action on the blockchain
  lps_my_account(Us),
  e_sendTransaction(Us, Player, Prize, PaymentTx) from T1 to T2,
  e_existsTransactionReceipt(PaymentTx) at T3.

if played(P0, Choice0, _) at T1, played(P1, Choice1, _) at T1, P0 \=\= P1, beats(Choice0, Choice1), not gameOver at T1 then
  initiate gameOver from T1, reward(Prize) at T1, pay(P0, Prize) from T1 to T2.

if played(P0, Choice, _) at T1, played(P1, Choice, _) at T1, P0 \$> P1, not gameOver at T1 then
  initiate gameOver from T1, reward(Prize) at T1, Half is Prize/2, pay(P0, Half) from T1, pay(P1, Half) from T1.
```

June 26, 2018

http://logicalcontracts.com
Reactive rules in LPS behave like imperative programs

```plaintext
if played(P0,Choice0), played(P1,Choice1), P0 <= P1,
beats(Choice0,Choice1), not gameOver, reward(Prize)
then initiate gameOver, pay(P0,Prize).
```

Imperative English

If a player has played a choice and another player has played another choice, and the choice beats the other choice, and the game is not over, and the reward is a prize then end the game and pay the player the prize.

Declarative English

If a player has played a choice and another player has played another choice, and the choice beats the other choice, and the game is not over, and the reward is a prize then the game becomes over and the player is paid the prize.
IMPERATIVE AND DEONTIC LOGIC

By P. T. Geach

The logic of proper imperatives is, I think, fairly trivial. For every proper imperative, there is a future-tense statement whose 'coming true' is identical with the fulfilment of the imperative. This is the source of everything that can be said about the inferability, incompatibility, etc. of imperatives; their being imperatives does not affect these logical interrelations.
Examples from

- The British Nationality Act logic programs.
- A simplified loan agreement deterministic finite automata.
- Rock, Paper, Scissors (Lessons and insights from a cryptocurrency lab) imperatives.
- ISDA Master Agreement (International Swaps and Derivatives Association) obligations.
ISDA
International Swaps and Derivatives Association, Inc.

2002 MASTER AGREEMENT
dated as of March 22, 2011

Bank of America, N.A. and LKQ Corporation

have entered and/or anticipate entering into one or more transactions (each a “Transaction”) that are or will be governed by this 2002 Master Agreement, which includes the schedule (the “Schedule”), and the documents and other confirming evidence (each a “Confirmation”) exchanged between the parties or otherwise effective for the purpose of confirming or evidencing those Transactions. This 2002 Master Agreement and the Schedule are together referred to as this “Master Agreement”.

Accordingly, the parties agree as follows:—

Possibly overriding some clauses

[Diagram of the Master Agreement structure]

ISDA Master Agreement

Schedule

Confirmation A

…

Confirmation Z
First Steps Towards Logical English

Author:
Vesko Karadotchev

Submitted in partial fulfillment of the requirements for the MSc degree in MSc Computing Science of Imperial College London

September 2019
2. Obligations

(a) General Conditions.

(i) Each party will make each payment or delivery specified in each Confirmation to be made by it, subject to the other provisions of this Agreement.

It is an obligation that a party performs an action if the action is specified in a confirmation and the action is a payment or the action is a delivery and it cannot be shown that it is not the case that it is an obligation that the party performs the action.

An obligation is satisfied if the obligation is that the party performs an action and the party performs the action.

An obligation is violated if it is not the case that the obligation is satisfied.
(c) Netting of Payments. If on any date amounts would otherwise be payable:—

(i) in the same currency; and

(ii) in respect of the same Transaction,

by each party to the other, then, on such date, each party’s obligation to make payment of any such amount will be automatically satisfied and discharged and, if the aggregate amount that would otherwise have been payable by one party exceeds the aggregate amount that would otherwise have been payable by the other party, replaced by an obligation upon the party by which the larger aggregate amount would have been payable to pay to the other party the excess of the larger aggregate amount over the smaller aggregate amount.

It is not the case that

it is an obligation that a party pays to a counterparty an amount in a currency for a transaction on a date

if

it is an obligation that the party pays to the counterparty a net amount in the currency for the transaction on the date.

It is an obligation that a party pays to a counterparty a net amount in a currency for a transaction on a date

if

the net amount is a larger aggregate amount minus a smaller aggregate amount

and

the larger aggregate amount is the sum of each amount of each payment by the party to the counterparty in the currency for the transaction on the date

and

the smaller aggregate amount is the sum of each amount of each payment by the counterparty to the party in the currency for the transaction on the date.
Legislation as Logic Programs*

Robert A. Kowalski

Department of Computing
Imperial College of Science, Technology and Medicine
London SW7 2BZ, UK

January 1991
Revised June 1992

Abstract. The linguistic style in which legislation is normally written has many similarities with the language of logic programming. However, examples of legal language taken from the British Nationality Act 1981, the University of Michigan lease termination clause, and the London Underground emergency notice suggest several ways in which the basic model of logic programming could usefully be extended. These extensions include the introduction of types, relative clauses, both ordinary negation and negation by failure, integrity constraints, metalevel reasoning and procedural notation.
Conclusion

Who would we expect to be the best person to write a computer program? A trained programmer, most of us would reply. Perhaps an unusually inventive end-user, given access to tools such as code generators, and fourth generation languages. But a lawyer...

Well, according to Bob Kowalski, a professor of computational logic at Imperial College London, lawyers are just the people. Kowalski, you may recall, devised the principles of logic programming, on which the PROLOG language. No matter that it may be a rather specialised and stilted form of English, it is still much more accessible to the average human than any mathematical formalism.

An example, according to Kowalski, is the emergency notice found in London Underground trains:

Press the alarm signal to alert the driver. The driver will stop immediately if any part of the train is in a station. If not, the train will proceed
Computational Logic and Human Thinking: How to be Artificially Intelligent

This earlier draft of a book of the same title, published in July 2011 by Cambridge University Press, presents the principles of Computational Logic, so that they can be applied in everyday life. I have written the main part of the book informally, both to reach a wider audience and to argue more convincingly that Computational Logic is useful for human thinking. However, I have also included a number of additional, more formal chapters for the more advanced reader.

Robert Kowalski

LPS
Logic Production Systems

Imperial College London
Department of Computing

First Steps Towards Logical English

Author:
Vesko Karadotchev

LPS aims to close the gap
Acknowledgements

Fariba Sadri for BNA + LPS
Miguel Calejo for LPS on SWISH + Logical Contracts
Jacinto Davila for preliminary version of Logical English
Vesko Karadotchev for most recent development of Logical English

Thank you
Additional background slides
A Survey and Classification of Controlled Natural Languages

Tobias Kuhn*
ETH Zurich and University of Zurich

What is here called controlled natural language (CNL) has traditionally been given many different names. Especially during the last four decades, a wide variety of such languages have been designed. They are applied to improve communication among humans, to improve translation, or to provide natural and intuitive representations for formal notations. Despite the apparent differences, it seems sensible to put all these languages under the same umbrella. To bring order to the variety of languages, a general classification scheme is presented here. A comprehensive survey of existing English-based CNLs is given, listing and describing 100 languages from 1930 until today. Classification of these languages reveals that they form a

© 2014 Association for Computational Linguistics
British Nationality
Act 1981

1981 CHAPTER 61

An Act to make fresh provision about citizenship and nationality, and to amend the Immigration Act 1971 as regards the right of abode in the United Kingdom.

[30th October 1981]

BE IT ENACTED by the Queen's most Excellent Majesty, by and with the advice and consent of the Lords Spiritual and Temporal, and Commons, in this present Parliament assembled, and by the authority of the same, as follows:—

PART I

BRITISH CITIZENSHIP

Acquisition after commencement

1.—(1) A person born in the United Kingdom after commencement shall be a British citizen if at the time of the birth by birth or adoption,

(a) a British citizen; or

(b) settled in the United Kingdom.

(2) A new-born infant who, after commencement, is found abandoned in the United Kingdom shall, unless the contrary is shown, be deemed for the purposes of subsection (1)—

(a) to have been born in the United Kingdom after commencement; and

(b) to have been born to a parent who at the time of the birth was a British citizen or settled in the United Kingdom.
Robert Kowalski on criticism of a project to put the British Nationality Act into Prolog

How the logic of the law is put on trial

Logic programming arouses strong reactions. Supporters, like Philip Leith, reported by Brian Bloomfield (Computer Guardian, March 23, 1987), find it hard to control their enthusiasm. Others, like Philip Leith, reported by Brian Bloomfield, find it hard to control their hostility.

At first, from its beginning in 1972, logic programming led a quiet and relatively unnoticed existence, until the Japanese identified it as the key to their fifth-generation computer systems. Meanwhile, professional computer scientists were taken by surprise and judged the Japanese proposals as being too optimistic. Software engineers discovered the lack of attention to orthodox software engineering practices. Artificial intelligence researchers criticised the attention given to the logic programming language Prolog and determined to make a more popular language Lisp.

The major responses to the Japanese challenge, such as Mira, the University of Strathclyde, and Enet, did their best to avoid imitating the Japanese.

In the past two years attitudes have begun to change without any special promotion outside Japan, logic programming has become accepted into the mainstream of computing.

But logic programming is still vulnerable to attack, because unlike approaches to computing it also aims to give insights into the way humans work.

In his article Bloomfield reports Philip Leith's attack on our use of logic programming at Imperial College to formalise the 1981 British Nationality Act. These attacks are based upon a misrepresentation of our work.

Our use of logic to formalise legislation is based on the thesis that much human knowledge is substantially complex and can usefully be formulated as

Robert Kowalski is Professor of Computational Logic at Imperial College of Science and Technology, University of London.
My learned box of tricks

Lawyers are about the last people most of us would think of putting in charge of our computers because they are often thought verbose and given to obscurantism.

Robert Kowalski, professor of computational logic at London's Imperial College of Science, Technology and Medicine, disagrees. In fact he thinks that the next generation of computer scientists should be recruited from the law, not mathematics.

At their best, says Professor Kowalski, who has just published a paper drawing parallels between legislation and logic programming, lawyers are capable of a clarity and precision of expression that matches that of mathematicians. But lawyers have an advantage over those same mathematicians, he says, in that they communicate in English, a language accessible to us all rather than only to a closed circle of the initiated.

Best of all, he says, are legal draftsmen. What they are really doing when they draft laws is writing programs expressed in human language to be executed by the train is in a station — is, in effect, a wonderful piece of programming.

Most computer scientists automatically fall back on mathematics to specify what a program should do, says the professor, but it may also be possible to achieve the same results in plain English.

Take the specification for sorting a sequence. Rendered in the kind of English which a legal draftsman might be at home with rather than in mathematical symbols, one might say, suggests professor Kowalski, that "the result of sorting a sequence should be an ordered permutation of the input sequence".

Concepts such as "order" and "permutation" in turn can also be defined in natural language rather than maths. Thus a sequence might be regarded as ordered "if for every pair of elements in the sequence the earlier one is smaller than or equal to the later element in the sequence".

Lawyers would make ideal program writers'

Professor Kowalski says: "If you gave the
15 April 1993

Professor Robert Kowalski
Department of Computing
Imperial College of Science, Technology and Medicine
180 Queen's Gate
London SW7 2BZ
ENGLAND

Dear Professor Kowalski

LEGISLATION AS LOGIC PROGRAMS

...for drafting of delegated legislation (mostly)... approaches between them vary considerably. I was interested to see that your redrafted versions of subsections 1 (1) and 1 (2) of the British Nationality Act 1981 were more like the way this Office would draft provisions of that kind than were the original provisions.
Contract as Automaton

Deterministic Finite Automaton (DFA) as a chain of event and consequence:

- Start state \( q_0 \) at the top
- Terminal states (3) at bottom
- “Happy” or intended path traced in green
- More “interesting” ramifications traced in black
The history of states and events can be visualised as a timeline (or Gantt chart)

```sql
1. observe request(borrower, 1000) at '2014-06-01T15:00'. % at 15:00.
2. observe advance(lender, 1000) at '2014-06-02T18:00'.
3. observe pay(borrower, lender, 550) at '2015-06-01T12:00'.
4. observe pay(borrower, lender, 525) at '2016-06-01T06:00'.
```
A smart contract in an imperative programming language

def player_input(choice):
    if num_players < 2 and msg.value == 1000:
        reward += msg.value
        player[num_players].address = msg.sender
        player[num_players].choice = choice
        num_players = num_players + 1
        return(0)
    else:
        return(-1)
def finalize():
    p0 = player[0].choice
    p1 = player[1].choice
    # If player 0 wins
    if check_winner[p0][p1] == 0:
        send(0,player[0].address, reward)
        return(0)
    # If player 1 wins
    elif check_winner[p0][p1] == 1:
        send(0,player[1].address, reward)
        return(1)
    # If no one wins
    else:
        send(0,player[0].address, reward/2)
        send(0,player[1].address, reward/2)
        return(2)
LPS aims to close the gap between logical and imperative computer languages, by performing actions to generate models to make goals of the logical form *if antecedent then consequence*. Model generation serves as a global imperative, which generates commands to make *consequence* true whenever *antecedents* become true.
A logical representation of a legal document that is close to natural human language, but executable by computer.

It can be used to:

- monitor compliance of the parties to a contract.
- enforce compliance, by issuing warnings and remedial actions.
- explore logical consequences of hypothetical scenarios.
- query and update the Ethereum blockchain