First steps towards Logical English

Vesko Karadotchev

Imperial College London

vk2018@ic.ac.uk

September 16, 2019

Supervisor: Fariba Sadri
Special thanks to Bob Kowalski and Miguel Calejo

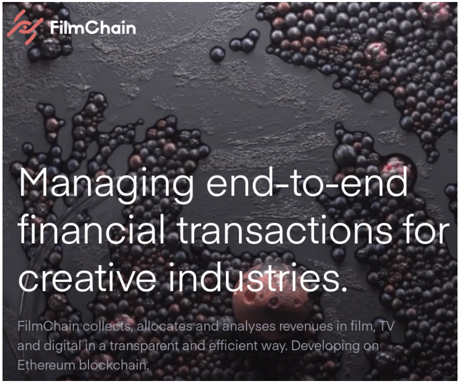
Overview

Motivation

```
it("can create a proposal on the fly", function() {
 var ballotInstance;
 const BILL_FOR_PRESIDENT_HEX = '42696c6c20666f7220707265736964656e74'
 return Ballot.deployed().then(function(instance) {
   ballotInstance = instance;
  return ballotInstance.createProposal("Bill for president");
}).then(function() {
  return ballotInstance.getProposalsCount.call();
}).then(function(index) {
  return ballotInstance.getProposalName.call(index.toNumber() - 1);
}).then(function(proposalName) {
  var str = proposalName.toString();
  var re = new RegExp(BILL_FOR_PRESIDENT_HEX, 'i');
  assert.match(str, re, "Bill for president was found!");
 })
});
```

Solidity code is difficult to understand

Motivation


FilmChain

Managing end-to-end financial transactions for creative industries.

FilmChain collects, allocates and analyses revenues in film, TV and digital in a transparent and efficient way. Developing on Ethereum blockchain.

FilmChain insight: movie industry does not trust smart contracts

Three high-level objectives for Logical English

- Readable to the casual English speaker
- Unambiguous and automatically translatable to executable code (e.g., Porlog)
- Expressive enough for legal applications (e.g., contract translation)

Building blocks: parts of speech

Building blocks: parts of speech

Proper (i.e. capitalised) nouns correspond to constants: **John**, **Jane**, **a Hat**, **the EiffelTower**, etc...

Common (i.e. lower-case, preceded by determiner) nouns correspond to variables: **a person**, **the unicorn**, **some water**, **a redDoor**, etc...

Building blocks: parts of speech

Proper (i.e. capitalised) nouns correspond to constants: **John**, **Jane**, **a Hat**, **the EiffelTower**, etc...

Common (i.e. lower-case, preceded by determiner) nouns correspond to variables: **a person**, **the unicorn**, **some water**, **a redDoor**, etc...

Verbs (i.e. lower-case, no determiner) correspond to predicate names: **runs**, **is**, **walksQuickly**, etc...

Building blocks: parts of speech

Proper (i.e. capitalised) nouns correspond to constants: **John**, **Jane**, **a Hat**, **the EiffelTower**, etc...

Common (i.e. lower-case, preceded by determiner) nouns correspond to variables: **a person**, **the unicorn**, **some water**, **a redDoor**, etc...

Verbs (i.e. lower-case, no determiner) correspond to predicate names: **runs**, **is**, **walksQuickly**, etc...

Reserved syntax (RS) words and phrases from a small dictionary:

- Logical connectives: **if**, **then**, **", and"**, **either... or...**.
- Determiners: **a**, **an**, **another**, **some**, **the**
- Prepositions: **on**, **from**, **to**, **and**, etc...
- Ordinals: **first**, **second**, **third**, etc...
- Phrases for special predicates/operators: **it cannot be shown that**, **not**, etc...

Building blocks: formulas

The LE basic formula:

```
[Reserved syntax] + Noun + Verb + [RS] + [Nouns]
```

Building blocks: formulas

The LE basic formula:

```
[Reserved syntax] + Noun + Verb + [RS] + [Nouns]
```

Basic LE formulas linked with reserved syntax to form more complex LE formulas:

```
LE formula = [RS] + LE formula + [RS] + [LE formula]...
```

Types of sentences

Types of sentences

- Simple sentences – correspond to ground facts

```
Jane buys a Hat. The Hat is a hat.
```

✗ Not an LE sentence: `Jane buys a hat.`

Types of sentences

- Simple sentences – correspond to ground facts

```
Jane buys a Hat. The Hat is a hat.
```

✗ Not an LE sentence: `Jane buys a hat.`

- Rules – correspond to LP clauses

```
Jane buys a hat if
Jane likes the hat.
```

Types of sentences

- Simple sentences – correspond to ground facts

```
Jane buys a Hat. The Hat is a hat.
```

✗ Not an LE sentence: `Jane buys a hat.`

- Rules – correspond to LP clauses

```
Jane buys a hat if
Jane likes the hat.
```

- Complex sentences – correspond to meta-predicates

```
Jane wants a hat so that
 the hat is red.
```

Entity types and hierarchies

Assigning types and building type hierarchies:

```
A thing is an animal if
  the thing is a bird.
```

```
type_is(animal, X) :-
  type_is(bird, X).
```

Entity types and hierarchies

Assigning types and building type hierarchies:

```
A thing is an animal if
  the thing is a bird.
```

```
type_is(animal, X) :-
  type_is(bird, X).
```

Syntactic sugar:

```
A bird is an animal.
```

Strong and weak negation

```
A debt is an obligation for a person if
the person incurred the debt , and
it cannot be shown that
the debt is not an obligation for the person.
```

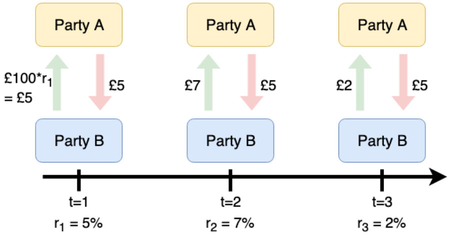
```
relat_is(obligation, Debt, Person) :-
  incurred(Person, Debt),
  \+relat_is(not, obligation, Debt, Person).
```

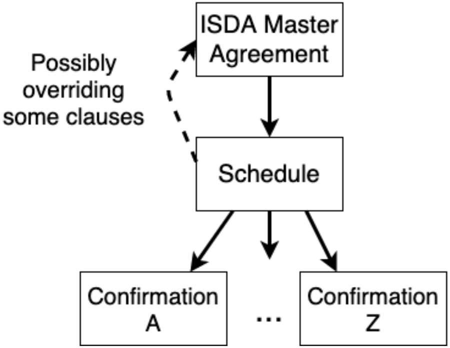
Meta-predicates

```
A taxBreak is approved if
a company asks the RelevantAuthority
  for the taxBreak , and
the RelevantAuthority determines that the
  the taxBreak is appropriate.
```

```
approved(TaxBreak) :-
  asks(Company, relevantAuthority,
    TaxBreak),
  determines(relevantAuthority,
    is_(appropriate, TaxBreak) ),
  type_is(company, Company),
  type_is(taxBreak, TaxBreak).
```

The ISDA Agreement





Defining the key individuals

*[Contract] ...***dated as of March 22, 2011**. **Bank and Corp 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".*

Defining the key individuals

```
Bank is a party. Corp is a party.
A thing is a counterparty if
  the thing is a party.
IsdaAgreement is an agreement between Bank
  and Corp.
IsdaAgreement is an agreement between Corp
  and Bank.
IsdaAgreement commences on 20110322.
```

Defining the key individuals

```
type_is(party, bank). type_is(party, corp).
type_is(counterparty, X) :-
  type_is(party, X).
relat_is(agreement, isdaAgreement, bank, corp).
relat_is(agreement, isdaAgreement, corp, bank).
commences(isdaAgreement, 20110322).
```

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.*

Obligations

```
An action is an obligation for a party if
a transaction specifies the action , and
the transaction is a transaction between
  the party and a counterparty , and
the action is directed from the party to
  the counterparty ,
and either
    the action is a payment
  or
    the action is a delivery ;
and
it cannot be shown that
the action is not an obligation for the party.
```

Obligations

```
relat_is(obligation, Action, Party) :-
 specifies(Transaction, Action),
 relat_is(transaction, Transaction, Party, Counterparty),
 relat_is(directed, Action, Party, Counterparty),
 relat_is(governed, Transaction, isdaAgreement),
 (
  type_is(payment, Action ) ;
  type_is(delivery, Action)
 ),
 \+(type_is(payment, Action ),
     type_is(delivery, Action)), % exclusive or
 \+relat_is(not, obligation, Action, Party).
```

Obligations: satisfying and breaking

```
% keep an obligation
An action is satisfied on a date if
the action is an obligation , and
the action is due on a second date , and
the action happens on the second date , and
the second date is before the first date.

% break an obligation
An obligation is broken on a date if
it cannot be shown that
the obligation is satisfied on the date.
```

Obligations: satisfying and breaking

```
% keep an obligation
relat_is(satisfied, Action, Date) :-
  relat_is(obligation, Action, Party),
  relat_is(due, Action, Date2),
  happens(Action, Date2),
  Date2 #<= Date.

% break an obligation
relat_is(broken, Obligation, Date1) :-
  \+relat_is(satisfied, Obligation, Date1).
```

Netting payments

(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 [...]* **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.*

Netting payments: long rule

```
A netPayment is a netPayment form a party
 to a counterparty in a currency for
 a transaction for a netAmount on a date if
isdaAgreement is an agreement between
 the party and the counterparty , and
```

```
relat_is(netpayment, NetPayment, Party, Counterparty,
          Currency, Transaction, Amount, Date) :-

  relat_is(agreement,isdaAgreement, Party, Counterparty),
```

Netting payments continued...

```
a first listOfPayments is an aggregatePayment
 for the party in the currency for
  the transaction on the date , and

a first aggregateAmount is an aggregAteamount
 for the first listOfPayments on the date, and
```

```
    relat_is(aggregatePayment, ListOfPayments1,
                 Party, Currency, Transaction, Date),
    relat_is(aggregateamount, A1, ListOfPayments1, Date),
```

Netting payments continued...

```
a second listOfPayments is an aggregatePayment
 for the counterparty in the currency for
  the transaction on the date , and

a second aggregateAmount is an aggregateAmount
 for the second listOfPayments on the date, and
```

```
  relat_is(aggregatePayment, ListOfPayments2,
                Counterparty, Currency, Transaction, Date),
  relat_is(aggregateamount, A2, ListOfPayments2, Date),
```

Netting payments continued...

```
the first aggregateAmount is larger than the
 second aggregateAmount , and
the netPayment is the concatenation of
 the first listOfPayments
  and the second listOfPayments , and
the netAmount equals the
 first aggregateAmount minus
  the second aggregateAmount.
```

```
   A1 #>= A2,
   append(ListOfPayments1, ListOfPayments2, NetPayment),
   Amount #= A1 - A2.
```

Demo of simple interpreter

Evaluation and future work

Verdict on LE

Evaluation and future work

Verdict on LE

- Lawyers: readable but not fully self explanatory
- Expressive enough for the ISDA
- But verbose, long rules get very challenging
- Interpreter incomplete

Evaluation and future work

Verdict on LE

- Lawyers: readable but not fully self explanatory
- Expressive enough for the ISDA
- But verbose, long rules get very challenging
- Interpreter incomplete

Future work

Evaluation and future work

Verdict on LE

- Lawyers: readable but not fully self explanatory
- Expressive enough for the ISDA
- But verbose, long rules get very challenging
- Interpreter incomplete

Future work

- Meta-predicates and reactive rules
- Adjectives and adverbs

The End[1]

[1]For the simple interpreter:
https://gitlab.doc.ic.ac.uk/vk2018/logical-english-lps

Sources for images

Example contract: https://www.verypossible.com/blog/ethereum-smart-contracts-learning-solidity-by-example
FilmChain: https://filmchain.co/