™

LEXON WRITER'S TUTORIAL

0.4.3.23-1

Lexon is under development. The facts and examples in this document will be replaced by better iterations. The principles described will hold. For updates, visit **https://www.Lexon.org.**

SEP. 2024 LEXON 0.3

TEXT VERSION 0.4.3.23-1 for LEXON VERSION 0.3
FIRST EDITION

Please send feedback to

tutorial@lexon.org

PLEASE SEND YOUR FEEDBACK,
THOUGHTS AND CRITICISM TO
TUTORIAL@LEXON.ORG

FOR UPDATES CHECK OUT THE
LEXON SITE
WWW.LEXON.ORG

TRY THE
ONLINE TUTORIAL
LEXON.ORG/TUTORIAL

WRITE CONTRACTS USING THE
ONLINE COMPILER
LEXON.ORG/COMPILER

GET IN TOUCH JOINING THE
TELEGRAM CHANNEL
T.ME/LEXONIANS

WWW.LEXON.ORG

ABOUT LEXON

Lexon is a computer language that anyone can read.

With Lexon, the same text is both a contract and a program. Lexon was made for blockchain smart contracts and makes them legally enforceable. It can be used on- and off-chain to express agreements, internal process flow, high-level business logic, regulations, and statute. Lawmakers can write 'Robotic Laws' in Lexon to accurately and transparently direct and constrain machines.

Lexon is based on AI, but not on machine learning. It needs no data pools, no training, minimal energy and hardware power, is perfectly accurate, transparent, and provides full agency.

Lexon is a programming language, based on advanced paradigms, implemented using mainstream compiler building technology.

Find updates about Lexon at https://www.Lexon.org.

CREDITS

Thank you for your contributions to Lexon to:

Carla Reyes, Brian Fox, Thomas Hardjono, T. J. Saw, Constance Choi, Daniel Nemet, Oliver Goodenough, David Bovil, Anja Blaj, Marina Markezic, Dominic Williams, Boris Adloff, Xenya Serova, Tom Montgomery, Yanislav Malahov, Marcelo Alaniz, Nicolas Guzzo, Benedikt Schuppli, Nikolas Guggenberger, Harald Stieber, Florian Glatz, Stan Stalnaker, Ed Hesse and Dan Barnhizer.

COPYRIGHT

TABLE OF CONTENTS

INTRODUCTION

Programs that anyone can read. Legalese that just works.

A new profession is rising, not from the ashes, but from the much-evolved body of the legal profession: the legal engineer.

Lexon will not replace lawyers, much less coders. But it will lift the veil that shrouds their magic. Stark change is around the corner: like retail and banking experienced, the internet keeps shaking up our world in fundamental ways, often hard to imagine. Entrenched players have learned the hard way that bits and bytes can be sound, light, or information. Now, thanks to the blockchain, bytes can be money.

Money more real than before, actually. Not just bank account 'money' that any banker will tell you is but a promise. Money more like central bank money that today only banks own. That's huge in itself. Lexon hypercharges it:

Contracts funnel money. Imagine they could be made to perform automatically, unbreakably. That's a smart contract. Imagine further you could write such a contract in plain English. And 'magically' it took care of itself, the receipts, the billing, the handling of edge cases, just as written. And as far as the payment side is concerned, it could not be broken. As scientists of economics will tell you, this changes the fundamental power equations of contracting. It will change not just legal practice but the meaning of negotiations, risk estimates, financial planning, and commerce. That's around the corner.

Lexon helps navigating this change, joining the old with the new, the cryptic with the obvious, the power of the word with the power of the electron. And by this makes blockchain technology accessible in a completely unexpected way that will touch many walks of life.

This document is for everyone who is curious and has an open mind. You will find use for Lexon that no-one thought of.

LANGUAGE

The following is a step-by-step introduction to writing digital contracts. It is an excerpt from the Lexon BIBLE.

It is assumed that you have a rough idea what Lexon is. This tutorial is not needed to read and understand Lexon but to learn to write it, to create Lexon digital contracts or statute.

If you haven't, **www.lexon.org/about.html** gets you started.

You find the most up-to-date version of this tutorial online at **www.lexon.org/tutorial.**

If you print this document, it's best printed in landscape, two pages per sheet, even page numbers left. Deleting a page in the PDF can be the fastest way to accomplish that.

Some dedicated remarks have been added to the tutorial steps to assist legal professionals and software developers to confirm their suspicions. It's ok to ignore them if you are neither.

▌ Text marked like this addresses legal professionals.

■ Text marked like this is for software developers.

1: DIGITAL CONTRACTS

```
LEX Managed Escrow.
LEXON: 0.2.22
PREAMBLE: 0.1.b - an escrow contract that is controlled
by a third party.

"Payer" is a person.
"Payee" is a person.
"Agent" is a person.
"Payment" is an amount.

The Payer pays the Payment into escrow,
appoints the Payee,
and appoints the Agent.

CLAUSE: Pay Out.
The Agent may pay the Payment from escrow to the
Payee.

CLAUSE: Pay Back.
The Agent may return the Payment from escrow to the
Payer.
```

Managed Escrow
A simple escrow contract.
Example for language version: 0.2.22 or higher
Lexon Example: 0.a.01 escrow/digital contracts

This example shows a simple escrow contract. It is a contract between a Payer and a Payee, in which a third person, the Agent, will decide where the money goes.

The code example shown is both a legally enforceable contract and a blockchain smart contract. With Lexon, a contract and a program can be the same thing. This is called a *digital contract*.

Digital contracts can be read by anyone, without any knowledge about programming. This tutorial is about learning to *write* them.

Digital contracts collapse the legal and the programming world. The consequences of this are not always obvious.

On the one hand, this code is admissible in court as proof for the *meeting of the minds* between two parties. On the other hand, it can be made to automatically, unstoppably and un-tamperably execute on a blockchain. This takes only a few clicks and is a matter of minutes.

For lawyers: Lexon *digital contracts* can be read by a judge, or anyone else, without the help of experts. Importantly, they are readable to the parties entering into the contract. An important motivation for using Lexon to express blockchain smart contracts is to deny the contracting partners a plausible deniability-excuse when trying to sue themselves out of a smart contract.

For programmers: The Lexon 0.3 compiler is built with the standard compiler building tools *Flex* and *Bison*. The grammar is expressed in *LGF*.

2: THE POINT OF LEXON

```
LEX Will.

"Grantor" is a person.
"Heir" is a person.
"Executor" is a person.

The Grantor pays an Amount into escrow,
appoints the Heir,
and also appoints the Executor.

CLAUSE: Execute.
The Executor may
pay the escrow to the Heir.
```

Will
A minimal will.
Example for language version: 0.2.22 or higher
Lexon Example: 0.a.02 escrow/tutorial

This example shows a minimal will. As a lawyer, you might find issue with it. *That's the very point of Lexon*: that you *can*, within seconds.

The novelty with Lexon is that computer code can be read by anyone. A great example is the right reaction of lawyers to the text in this example: it is probably insufficient in many ways.

But the point of Lexon is that lawyers *can* chime in immediately to discuss how a text could be made right. Mind, this *is* a program being discussed as to its correctness. By lawyers.

Likewise, businesspeople, potential customers or business partners will benefit from being able to actually read the digital contract they might be asked to enter into.

Lawyers: this smart contract, as a program that can manage money, will work. The interesting question is, can it be made safer and compliant by writing it the right way?

Lexon helps to put the challenges of smart contracts in sharp relief. It helps to get the contracts right – legally as well as logically – because it broadens the scope of who can take part in their verification.

3: THE MAIN PARTS

```
LEX Managed Escrow.
LEXON: 0.2.22
PREAMBLE: 0.1.b - an escrow contract that is controlled
by a third party.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
"Payer" is a person.
"Payee" is a person.
"Agent" is a person.
"Payment" is an amount.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
The Payer pays the Payment into escrow,
appoints the Payee,
and appoints the Agent.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
CLAUSE: Pay Out.
The Agent may pay the Payment from escrow to the
Payee.

CLAUSE: Pay Back.
The Agent may return the Payment from escrow to the
Payer.
```

Managed Escrow
A typical, brief smart contract
Example for language version: 0.2.22 and higher
Lexon Example: 0.a.04 escrow/parts escrow

Lexon Digital Contracts have a document structure that resembles traditional paper contracts.

The Lexon code shown here is the same escrow contract as before on the previous page.

Lexon texts always starts with a

- **Head** that begins with the keyword **LEX**, a name, and can have more meta information.

Three main parts follow:

- **Definitions** that describe the meaning of *names* as used in the contract.
- **A Recital** that describes what happens when the contract is first brought into existence.
- **Clauses** that list the *events* that the contract covers.

There can be more parts in more complex digital contracts.

Code of Lexon digital contracts can be *inserted* into the prose of traditional contracts. But it can also stand alone. In this tutorial we will mainly look at stand-alone examples.

Lawyers: In general, blockchain smart contracts support two distinct situations: they can be a small, automated part of a larger contract, or they can be the complete contract for a simple agreement. There are quite complex smart contract programs out there meanwhile and it turns out that a surprisingly large share of traditional contract prose can be 'blockchainyfied,' i.e., automated. Lexon increases the scope where blockchain smart contracts can help.

```
LEX Will.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
"Grantor" is a person.
"Heir" is a person.
"Executor" is a person.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
The Grantor pays an Amount into escrow,
appoints the Heir,
and also appoints the Executor.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
CLAUSE: Execute.
The Executor may
pay the escrow to the Heir.
```

Will
A simple smart contract will
Example for language version: 0.2.22 or higher
Lexon Example: 0.a.05 escrow/parts will

This contract has the same parts as in the previous example:

- **Head** beginning on **LEX**, followed by a name.
- **Definitions** that define names.
- **Recitals** describing the first things to happen.
- **Clause** - in this case just one.

The Lexon code shown here is the same (flawed) estate contract as shown before.

Programmers: Obviously, **definitions** are *type declarations*, the **recitals** are the *constructor* code and **clauses** are *functions*. There are similarities like this throughout the design of the Lexon language. It is at heart, but a computer language built based on the same premises and with the same tools as other programming languages.

Let's take a closer look at the individual parts of a digital contract:

4: HEAD

```
LEX Managed Escrow.
LEXON: 0.2.22
PREAMBLE: An escrow contract that is controlled by a
third party.

"Payer" is a person.
"Payee" is a person.
"Agent" is a person.
"Payment" is an amount.

The Payer pays the Payment into escrow,
appoints the Payee,
and appoints the Agent.

CLAUSE: Pay Out.
The Agent may pay the Payment from escrow to the
Payee.

CLAUSE: Pay Back.
The Agent may return the Payment from escrow to the
Payer.
```

Managed Escrow
A simple escrow contract
Example for language version: 0.2.22 or hihger
Lexon Example: 0.a.06 escrow/head

Lexon texts always starts with the **head**. It consists of:

1. The **LEX** keyword, followed by the *name* of the digital contract. The **LEX** keyword tells both the computer and a judge that this is the start of the automated parts of a contract.

The name can be any sensible name that helps remembering the contents, managing or filing this code.

The head may consist of only the **LEX** keyword and the name that follows it.

2. The **LEXON** tag is optional. If it is present, it is followed by a *version number* that indicates with which version of Lexon the code will work. This is a concession to the fact that Lexon is software and evolving. This version number establishes the link between the code and the revision of the Lexon language it was made for. The current Lexon version is 0.3 but the examples work for 0.2, too.

3. The **PREAMBLE** is also optional. This keyword is followed by a *description* of the contract text that follows. The preamble text is neither legally binding (as preambles in normal contracts never are!) nor part of the automation. It should be followed by an empty line for clarity.

> **Lawyers:** To have a *keyword* like **LEX** is useful for the legal perspective of a digital contract. It provides clear separation between the automated parts of a digital contract and legal prose (if any) that might precede it. Because of this keyword requirement, Lexon digital contracts are *not* completely seamlessly embedded in a larger document prose. But a judge would, at any rate, never be completely ignorant of the fact that there is automation in play with a digital contract. Therefore, it will only help to have a clear indication of where the automation text starts: with **LEX**.

The **definitions** follow the **head**:

5: DEFINITIONS

```
LEX Managed Escrow.
LEXON: 0.2.22
PREAMBLE: 0.1.b - an escrow contract that is controlled
by a third party.

"Payer" is a person.
"Payee" is a person.
"Agent" is a person.
"Payment" is an amount.

The Payer pays the Payment into escrow,
appoints the Payee,
and appoints the Agent.

CLAUSE: Pay Out.
The Agent may pay the Payment from escrow to the
Payee.

CLAUSE: Pay Back.
The Agent may return the Payment from escrow to the
Payer.
```

Managed Escrow
A simple escrow contract
Example for language version: 0.2.22 or higher
Lexon Example: 0.a.07 escrow/definitions

The first thing after the **head** are the **definitions**.

Definitions start with a *name* followed by **'is a person'**, '**is an amount**' or whatever else can be said about what the respective name stands for.

Because the contract examples we are looking at here are really *templates*, we do not yet fix a concrete amount in the definitions, nor a concrete street address or blockchain wallet address. Rather, keywords like **person**, **amount**, or **data** are used to give a first indication of what a name will mean. We're just clarifying the *category* a name falls into. The concrete information will be put into place once the contract is digitally signed and moved towards the blockchain. In blockchain parlance, *deployed.*

Names can be defined as:

- **person**
- **amount**
- **data**

Lawyers: Definitions are something lawyers are familiar with. With Lexon, definitions are less concrete than usual because Lexon code at this stage is a template for multiple contracts, rather than one concrete contract.

Programmers: The definitions evidently use *data types*:

- **person** is a blockchain address.
- **amount** is an integer.
- **data** is a hash.

Lexon has *composite types*. They are limited in scope, so that they can be understood intuitively by non-programmers. But they make for powerful constructs. More on that later.

6: RECITAL

```
LEX Managed Escrow.
LEXON: 0.2.22
PREAMBLE: An escrow contract that is controlled by a
third party.

"Payer" is a person.
"Payee" is a person.
"Agent" is a person.
"Payment" is an amount.

The Payer pays the Payment into escrow,
appoints the Payee,
and appoints the Agent.

CLAUSE: Pay Out.
The Agent may pay the Payment from escrow to the
Payee.

CLAUSE: Pay Back.
The Agent may return the Payment from escrow to the
Payer.
```

Managed Escrow
A simple escrow contract
Example for language version: 0.2.22 or higher
Lexon Example: 0.a.08 escrow/recitals

The **recital** sets up the context for the performance of the contract. It works like in a paper contract: it is text that is performed *once* before any clauses are executed. It cannot be skipped and clauses cannot be triggered before it.

As opposed to the **PREAMBLE**, the recital is a binding part of the contract, its first part, *not* a paraphrasing comment.

There is no keyword that marks the beginning of the recital. I.e., the recital does *not* start on the word '**RECITAL**.' It is simply the text that follows right after the **definitions.**

The recital is articulated in present tense. In Lexon 0.3, it cannot contain the word **may**. Everything stated in the recital must happen.

> **Lawyers:** Just as in a traditionally paper contract, where parties commonly use recitals to list the actions taken that led them to enter into the agreement, a Lexon recital provides the prerequisite foundation for the clauses that follow.

> **Programmers:** Recitals are *constructor* code.

The main part of a digital contract follows: the **clauses**.

7: CLAUSES

```
LEX Managed Escrow.
LEXON: 0.2.22
PREAMBLE: An escrow contract that is controlled by a
third party.

"Payer" is a person.
"Payee" is a person.
"Agent" is a person.
"Payment" is an amount.

The Payer pays the Payment into escrow,
appoints the Payee,
and appoints the Agent.

CLAUSE: Pay Out.
The Agent may pay the Payment from escrow to the
Payee.

CLAUSE: Pay Back.
The Agent may return the Payment from escrow to the
Payer.
```

Managed Escrow
A simple escrow contract
Example for language version: 0.2.22 or higher
Lexon Example: 0.a.09 escrow/clauses

The **clauses** describe *events* that can occur during the lifetime of the digital contract. Such clauses, as in the example on the opposite page, consist of sentences called **statements**.

(Clauses can also be used to provide the explanation of a term, which is an alternate way of defining the meaning of a word.)

Note that every clause in a digital contract is optional. Digital contracts cannot oblige anyone to do anything.

This is because blockchain smart contracts cannot 'reach out' of the blockchain. They can only *incentivize* parties to the contract by requiring stakes and rewarding actions.

Recitals and **clauses** must have a person as an acting subject. They must describe someone doing something. They cannot be phrased as abstractly as sentences in a paper contract often are. For a blockchain smart contract, for anything to happen at all, someone has to 'trigger' it. Therefore, digital contracts do not, e.g., express requirements in the passive tense.

Lawyers: The fact that blockchain smart contracts require an *actor* 'for anything to happen' and cannot *oblige* parties to the contract to anything, forces a shift in the way that lawyers usually write contracts. Playing to the strengths of blockchain smart contracts requires a focus on providing *incentives* for performance rather than emphasizing legal remedies in the event of a breach. This does not make digital contracts not-contracts: but they are based on a *subset* of mechanisms that can be employed in contracts. See the *Holy Grail* paper at **https://lexon.org/papers** on how obligations will be realized, and *Efficient Breach* in the Lexon book and BIBLE.

Programmers: Clauses are basically *functions* or *procedures.*

8: SENTENCES

```
LEX Managed Escrow.
LEXON: 0.2.22
PREAMBLE: An escrow contract that is controlled by a
third party.

"Payer" is a person.
"Payee" is a person.
"Agent" is a person.
"Payment" is an amount.

The Payer pays the Payment into escrow,
appoints the Payee,
and appoints the Agent.

CLAUSE: Pay Out.
The Agent may pay the Payment from escrow to the
Payee.

CLAUSE: Pay Back.
The Agent may return the Payment from escrow to the
Payer.
```

Managed Escrow
A simple escrow contract
Example for language version: 0.2.22 or higher
Lexon Example: 0.a.10 escrow/sentences

Sentences in a digital contract have a structure like in natural language: they consist of a *subject* and a *predicate*.

The predicate, in turn, consists of *verb* and *object*. This results in the standard English sentence structure of *subject-verb-object*.

9: PREDICATES

```
LEX Managed Escrow.
LEXON: 0.2.22
PREAMBLE: An escrow contract that is controlled by a
third party.

"Payer" is a person.
"Payee" is a person.
"Agent" is a person.
"Payment" is an amount.

The Payer pays the Payment into escrow,
appoints the Payee,
and appoints the Agent.

CLAUSE: Pay Out.
The Agent may pay the Payment from escrow to the
Payee.

CLAUSE: Pay Back.
The Agent may return the Payment from escrow to the
Payer.
```

Managed Escrow
A simple escrow contract
Example for language version: 0.2.22 or higher
Lexon Example: 0.a.11 escrow/predicates

Predicates consist of one or multiple *verbs* and *objects*, like in natural language.

Programmers: Note that Lexon allows for the concatenation of verbs that belong to the same subject.

10: THE WORD 'MAY'

```
LEX Managed Escrow.
LEXON: 0.2.22
PREAMBLE: An escrow contract that is controlled by a
third party.

"Payer" is a person.
"Payee" is a person.
"Agent" is a person.
"Payment" is an amount.

The Payer pays the Payment into escrow,
appoints the Payee,
and appoints the Agent.

CLAUSE: Pay Out.
The Agent may pay the Payment from escrow to the
Payee.

CLAUSE: Pay Back.
The Agent may return the Payment from escrow to the
Payer.
```

Managed Escrow
A simple escrow contract
Example for language version: 0.2.22 or higher
Lexon Example: 0.a.12 escrow/permission

May has a pronounced role in digital contracts:

The keyword **may** controls who can do what with a digital contract. It works like a gatekeeper on the **clauses**.

Note that the logic applied in digital contracts is *otherwise* strictly Boolean (i.e., **binary**: *yes/no*) and not Deontic (obligations and permissions: *must/must not*). **May** is the exception to this rule and used to specify *permissions* regarding the performance of individual **clauses** of a digital contract.

Lawyers: The term **may**, in digital contracts, gives a party to the contract *agency*. Only those allowed by a **may** clause to perform an action, can. In the example above, only the Agent can perform the **Pay Out** or **Pay Back**. As seen from the blockchain, if the wrong party tries to perform these clauses, the smart contract will simply cancel the attempt.

Programmers: The keyword **may** acts like a *guard*, a permission operator to functions. It works like an *assertion*.

Let's examine a small example.

11: A SIMPLE EXAMPLE

```
LEX Payment.

"Payer" is a person.
"Payee" is a person.
"Payment" is an amount.

The Payer pays a Payment to the Payee.
```

Payment
Simple transfer of funds
Example for language version: 0.2.22 or higher
Lexon Example: 0.b.1 payment

This is a payment, nothing more. The payer literally just transfers an amount of tokens to the payee.

This simple 'contract' has only three parts: the **head**, consisting only of the line starting on **LEX**, after that the **definitions** are given, and last, a **recital**. It does not have **clauses**, like a normal digital contract will always have.

Therefore, this is maybe not 'really' a smart contract, because it is so simple. But let's use the simplicity to highlight a couple of things.

LEX is followed by the *name* you want to give the script, in this example, the word 'Payment'. This name is used to organize code and contracts but does *not* have any special meaning in the contract text that follows.

The next line is a **definition**, it defines what a 'Payer' is supposed to be. It does not get concrete yet but rather just defines a category. In this case, a **person**. This points to the fact that Lexon contracts are really templates, because a concrete contract would have a concrete name in this place. When a Lexon contract is deployed to the blockchain, what it gets instead is a unique numeric ID that serves the role of a passport number in the blockchain world: it unambiguously defines which person the Payer is.

After the Payer, we also define what the Payee and the Payment are: a **person** and an **amount** respectively. When the contract is deployed to the blockchain, those two parameters will also be made concrete.

Finally, there is the heart of the contract, the 'code': in this case it is not conditional - as contracts usually are - but simply money made changing hands. After that happens, the contract terminates. That there is no conditionality is because we just have a **recital,** and the performance of the recital is mandatory, not optional like **clauses** are. For this example, that was the intention.

12: LEXON TEXTS ARE TEMPLATES

Aoll Lexon 0.3 texts are really templates. This will change in later version.

But for now, only once a contract is signed and deployed to a blockchain, will persons named be identified by concrete blockchain addresses. It is in that sense that the Simple Example shown above really is a *template* and the names defined in it are placeholders.

> **Lawyers:** A contract's *meaning*, of course, does not depend on the names of definitions that you use. Neither does the automated performance 'care', i.e., the computer when performing it.
>
> Note that we are deliberately blurry with the use of the words contract and smart contract at this point. We will be very precise later.
>
> You might know that there was a passionate discussion in the blockchain and the overlapping legal space, whether smart contracts are contracts or smart at all.
>
> The question was always misleading. As we have seen, a more complex Lexon smart contract will often define multiple legal contracts.

We will now look at two things that 'do not matter' for the *meaning* of Lexon code: automatic *coloring* of the online editor and your choice of *names*.

13: COLOR CODE

In the vocabulary examples at www.lexon.org/vocabulary, one can switch on color coding, and there will be a color code for the Lexon 0.3 online compiler.

These color schemes exist to help the reader. They also support writing. They do not affect automation or legal weight.

Green – or bold in black and white display and in this document's word reference – marks *names* that an author of a smart contract invented in **definitions** or **clauses.**

Blue (bold or not) marks the Lexon keywords that the Lexon compiler immediately understands (those in the language reference).

Violet marks numbers.

Lawyers: The colors are only a matter of support while reading or writing code. They do not affect the legal meaning nor the machine's interpretation of the code. The colors are not 'part' of the document. They are added on the fly by the online editor when displaying code.

Programmers:

Green marks *variable* and *function* names

Blue marks Lexon *keywords*.

Violet marks *literals*.

14: EXCHANGING NAMES

```
LEX Transfer.

"Sender" is a person.
"Receiver" is a person.
"Sum" is an amount.

The Sender pays a Sum to the Receiver.
```

Payment
Simple transfer of funds
Example for language version: 0.2.22 or higher
Lexon Example: 0.b.3 payment/renamed

This code has the same functionality and meaning as the one shown before. It just uses different names.

The point is, if you change the definition names, or the clause names, or the name of the contract, it does not change the logic of the contract. Its meaning is not dependent on how you call the parties.

Programmers: this is the same as with any program. Variable names eventually don't matter.

15: SYNONYMOUS VERBS

```
LEX Transfer.

"Sender" is a person.
"Receiver" is a person.
"Sum" is an amount.

The Sender transfers a Sum to the Receiver.
```

Payment
Simple transfer of funds
Example for language version: 0.2.22 or higher
Lexon Example: 0.b.4 payment/synonyms

This code has the same functionality and meaning as the one shown before, it just uses **transfer** instead of **pay**.

If you use a synonymous verb, it also does not change the logic of the contract.

We will later see how 'names' can actually amount to entire phrases, which contributes massively to the readability of Lexon code.

> **Lawyers:** a legal contract's meaning can change materially if just a single operative word is changed. A paper is forthcoming that explains why Lexon's vocabulary specifically cannot be misconstrued and can sustain synonyms. Though strictly speaking unnecessary, the Lexon synonyms - and the fact that they are synonyms - will also be explained in an auto-generated glossary of terms that Lexon will be able to create that describes the entire vocabulary used in Lexon.

16: NEUTRAL NAMES

```
LEX Transfer.

"A" is a person.
"B" is a person.
"C" is an amount.

A transfers C to B.
```

Payment
Simple transfer of funds
Example for language version: 0.2.22 or higher
Lexon Example: 0.b.5 payment/one letter names

This code has the same functionality and meaning as the one shown before.

As you see, the names really don't matter. You cannot do this with a verb though. Instead of 'transfers' you could write 'pays' or 'returns' if that makes the contract better readable for human readers.

> This is in keeping with how definitions can be used in legal texts.

There is another change versus the previous example: in the recital 'A transfers C to B.' all articles have been left out:

17: ARTICLES

Articles (a, an, the) can be freely used or left out.

```
LEX Payment.

"Payer" is person.
"Payee" is person.
"Payment" is amount.

Payer pays Payment to Payee.
```

Payment
Simple transfer of funds
Example for language version: 0.2.22
Lexon Example: 0.b.6 payment/articles

The meaning of the text is identical to the one before.

Articles and some other words in Lexon are called 'fillers'. They have a big role in making a text easy to read for a human but are irrelevant to the automation of the contract on the blockchain.

> **Lawyers**: Obviously, articles can fundamentally change the meaning of a contract. The writer of a Lexon contract must take care to not abuse them. Reining in the possibilities for abuse of fill words is a high priority for future Lexon tools. But language is not the right level to prevent fraud as it must provide uninhibited expression. Lexon is not the promise - at all - that smart contracts could not be misleading. Lexon is the promise that smart contracts can be readable.

18: MISLEADING NAMES

Language can be used to deceive.

```
LEX Payment.

"Payee" is person.
"Payer" is person.
"Payment" is amount.

Payee pays Payment to Payer.
```

Payment
Simple transfer of funds
Example for language version: 0.2.22 or higher
Lexon Example: 0.b.7 payment/misleading names

The meaning of this code is identical to the one shown before. It is just the labels that are misleading.

There is nothing in the language itself that keeps you from using misleading definitions. It will confuse readers but the automation of the contract will still work. At this point, the interpretation of a human and the machine may fork for a moment but on close inspection a human should find it just non-sensical. This would be the case with *any* such attempts to deceive. It's not possible to write the perfect deception.

Lawyers: A judge may throw this contract out because it is going to be hard to argue that switching the words **Payer** and **Payee** served a purpose that both sides agreed upon.

19: META INFORMATION

```
LEX Timestamp.
LEXON: 0.2.16
COMMENT: 1.a.1 - a timestamp of a data hash

The "Data" is a data.
The "Time" is a time.

Data be certified,
with Time fixed as the current time.
```

Payment
Simple Timestamping
Example for language version: 0.2.22
Lexon Example: 1.a.1 timestamp/data 1

Two keywords, **LEXON** and **COMMENT** can help working with Lexon code. The information associated with them is not part of the code proper but helps filing, processing and understanding it.

This contract attaches a timestamp to arbitrary information that will serve to prove that a certain information existed at a certain point in time.

The keyword **LEXON** is followed by a colon and then the compiler and language version number for which the code below it is intended for. In programming, this is a crucial information to avoid errors due to a change in rules. After the **COMMENT** keyword, any text may be noted that names or gives context to the understanding of the code below it. Both **LEXON** and **COMMENT** are optional.

Congrats for working through this, you should have a very good idea now about how Lexon text is structured.

Get interactive at **lexon.org/compiler**, which has more examples that you can tinker with and check out what works.

A manual and tutorial is online at **lexon.org/manual**.

Possibly the best way to get a feel for Lexon is to play around with the online vocabulary at **lexon.org/vocabulary**.

That page is connected with some excellent examples, which are also explained in this document but online have every single word wired to the vocabulary, for a very fast way to get an impression of how Lexon grammar looks in practice.

We will now turn to three interesting, larger examples.

EXAMPLES

The following examples show what Lexon can do and how digital contracts look in practice.

If you are reading the electronic version of this document, every keyword in the examples is linked to the online vocabulary. You should give that webpage a try and randomly browse it, clicking from word to word and example to the example.

The first example is the familiar escrow example that is used throughout the Lexon material, listed here for completeness.

The second example (pg. 37) is a license agreement that was written for a paper that compares Lexon with other languages.

Finally, we look at what US law would look like written in Lexon (pg. 42). This is an actual proposal to the reform committee of the Universal Commercial Code (UCC).

There are more examples — a trade contract, a DAO LLC, estate law, a variant of the Moloch DAO, and for future Lexon code (version 0.4 and beyond) in the Lexon book and BIBLE.

Note how the examples in this section are digital contracts that attain full identity of program and legal prose. They are *digitally expressed* rather than merely *digitally enhanced.*

The examples are compatible with Lexon 0.2 and 0.3.

MINIMAL: ESCROW EXAMPLE

This is the example that frequently features in the Lexon material, from the 2019 books to the 2024 whitepapers.[1]

Note that all bold keywords are clickable if you are looking at an electronic version of this document. You find this and the other examples online at **https://lexon.org/examples**.

```
LEX Escrow.

"Payer" is a person.
"Payee" is a person.
"Arbiter" is a person.
"Fee" is an amount.

The Payer pays an Amount into escrow, appoints the Payee,
appoints the Arbiter, and fixes the Fee.

CLAUSE: Pay Out.
The Arbiter may pay from escrow the Fee to themselves, and
afterwards pay the remainder of the escrow to the Payee.

CLAUSE: Pay Back.
The Arbiter may pay from escrow the Fee to themselves, and
afterwards return the remainder of the escrow to the Payer.
```

Fig 1 – Escrow Example (clickable)

This example shows a simple escrow contract. It is a contract between a Payer and a Payee, in which a third person, the Agent, will decide where the money goes.

Like all digital contracts, this code is both 1) admissible in court and 2) can be performed on a blockchain as smart contract.

[1] See **https://www.lexon.org/books** and **https://www.lexon.org/papers**.

SUBCONTRACTS: AN EVALUATION LICENSE

This digital contract was created by Florian Idelberger, PhD candidate at the European University Institute in Florence. It appears in MERGING TRADITIONAL CONTRACTS (OR LAW) AND (SMART) E-CONTRACTS – A NOVEL APPROACH,[2] comparing this text to smart contracts written in other languages.

```
LEX: Evaluation License System.

LEXON: 0.2.1
AUTHORS: FLORIAN IDELBERGER, HENNING
DIEDRICH

PREAMBLE: This is a licensing contract for a software
evaluation.

TERMS:

"Licensor" is a person.
"Arbiter" is a person.
"Licensing Fee" is an amount.
"Breach Fee" is an amount.

The Licensor appoints the Arbiter,
fixes the Licensing Fee,
and fixes the Breach Fee.

TERMS PER License:
```

[2] © 2020 Florian Idelberger – **https://lawgorithm.com.br/wp-content/uploads/2020/09/MLR2020-Florian-Idelberger.pdf**

```
"Description of Goods" is a text.
"Licensee" is a person.
"Paid" is a binary.
"Commissioned" is a binary.
"Comment Text" is a text.
"Published" is a binary.
"Permission to Comment" is a binary.
"Notice Time" is a time.
"License" is this contract.

The Licensor appoints the Licensee, and fixes the
Description of Goods.

CLAUSE: Pay.
The Licensee pays the Licensing Fee to the Licensor,
and pays the Breach Fee into escrow.
This License is therefore Paid.

CLAUSE: Commission.
The Licensor may certify this License as Commissioned.

CLAUSE: Comment.
The Licensee may register a Comment Text.

CLAUSE: Publication.
The Licensee may certify this License as Published.

CLAUSE: Grant Permission to Comment.
The Licensee may grant the Permission to Comment.

CLAUSE: Declare Breach.
The Arbiter may, if this License is Factually Breached:
pay the Breach Fee to the Licensor,
and afterwards terminate this License.

CLAUSE: Factually Breached.
```

"Factually Breached" **is defined as**:
this License **is** Commissioned **and the** Comment Text **is not fixed**,
or this License **is** Published **and there is no** Permission to Comment **and the** Notice Time **lies at least** 24 **hours in the past**.

CLAUSE: Notice.
The Licensor **or the** Arbiter **may fix the** Notice Time **as the respective current time**.

CLAUSE: Noticed.
"Noticed" **is defined as a** Notice Time **being fixed**.

Fig 2 – License Agreement (Idelberger)

Idelberger describes the license (pg. 3, ibid.):

"The test case is a license contract to license a copy of a software or other specified work for use and evaluation, in exchange for a licensing fee. Furthermore, sublicensees can be specified. These grants and license are defined in article 1. The sublicense part was inspired by Surden's description of a licensing system where universities can automatically manage the licenses of their libraries and conclude more tailored licensing agreements. In article 2, it is defined that optionally, the licensee or sublicensee is commissioned to publish comments about the use of the product. This approves publication, but also requires it. In article 3, publishing of comments about the use and evaluation of the asset without approval by the licensor beforehand is prohibited. In case of unauthorized publication, the licensee has 24 hours to remove the published material. This improves the test case, as it requires use of external agents or data sources depending on the system, as otherwise there is no basis on which to automate or act. Additionally, the passing of time is tested."

The original description of this logic is given as (ibid.):

This license is an example evaluation license.

LICENSEE - The University
SUBLICENSEE - A student as set force in the appendix. (array of persons)
ARBITER - An arbiter or oracle that decides in case of disputes. Can be a natural or legal person or a machine. Art. 2, 3 and 4 especially are evaluated by the arbiter in case of disputes.
ASSET X - An asset to be licensed.

Article 1. *The Licensor grants the Licensee a license to use and evaluate asset X and grant sublicenses among group Y, for use and evaluation. This grant is in exchange for a licensing fee.*

Article 2. *(optional) The (Sub)Licensee is commissioned to publish comments about the use of the product. This allows publication of comments but also requires them.*

Article 3. *The (Sub)Licensee must not publish comments of the use and evaluation of the Product without the approval of the Licensor; the approval must be obtained before the publication. If the Licensee publishes results of the evaluation of the Product without approval from the Licensor, the Licensee has 24 h to remove the material.*

Article 4. *This license terminates automatically if the (Sub)Licensee breaches this Agreement. Breach obliges the licensee to pay a fee to Licensor for Breach of the Licensing Terms.*

Fig 3 – algorithm of the license agreement

SUBCONTRACTS (COVENANTS)

The license illustrates the concept of *covenants*, or subcontracts. This is a case of the *Contract Factory*: the digital contract is a description of multiple, separate 1:1 agreements.

Everything after **TERMS** (before **TERMS PER**) exists and happens only once. There is only one *Licensor*, the *Licensor* appoints the *Arbiter* only once, etc.:

```
TERMS:

"Licensor" is a person.
"Arbiter" is a person.
"Licensing Fee" is an amount.
"Breach Fee" is an amount.

The Licensor appoints the Arbiter,
fixes the Licensing Fee,
and fixes the Breach Fee.
```

Everything after **TERMS PER License** describes potentially multiple agreements with different individuals. There can be multiple *Licenses,* and each have their own *Description of Goods*, etc.:

```
TERMS PER License:

"Description of Goods" is a text.
"Licensee" is a person.
"Paid" is a binary.
...
The Licensor appoints the Licensee, and fixes the
Description of Goods.
```

This is what the tag **TERMS PER** effects. Also, all clauses listed below it are part of (potentially multiple) individual agreement.

DIGITAL LAW: U.C.C. FINANCIAL STATEMENT

This Lexon text is model trade statute developed by asst. prof. Carla L. Reyes. For an in-depth, abstract and concrete legal discussion SEE REYES, CARLA, CREATING CRYPTOLAW FOR THE UNIFORM COMMERCIAL CODE (2021).[3] For additional technical details, including a terminal run-through, see **https://www.lexon.org/reyes**.

LEXON DIGITAL LAW

This Lexon text example is part of a proposal to reform the notice filing system included in *Article 9* of the *U.S. Uniform Commercial Code* (UCC).[4, 5] Specifically, the example is of a smart contract-based UCC-1 form – a financing statement that secured lenders use to notify other prospective lenders that a loan has been made that takes specific assets as collateral. The main function of the UCC-1 financing statement is not in its cryptocurrency aspect. Rather, the key aspect is foremost about record keeping.

This *digital law* allows to keep track of the status of the UCC Financing Statement and related collateral in a way that is more powerful than the current implementation in US law. It would better serve the notice function of the Article 9 filing system.

This example proposes the implementation and performance of digital law with the understanding that the states' filing offices *could implement and enforce law directly on the blockchain.*

[3] © 2021 Carla L. Reyes. Washington and Lee Law Review 1521 (2021), SMU Dedman School of Law Legal Studies Research Paper No. 502. – **https://ssrn.com/abstract=3809901**

[4] For further discussion of the underlying concepts, see Carla L. Reyes, *Conceptualizing Cryptolaw*, 96 NEB. L. REV. 384 (2017).

[5] UCC § 9-502. CONTENTS OF FINANCING STATEMENT – **https://www.law.cornell.edu/ucc/9/9-502**

This code has **definitions**, a lot of **clauses** but *no* **recitals**. It is structurally simpler than the short escrow contract listed above.

```
LEX UCC Financing Statement.

LEXON: 0.2.12

"Financing Statement" is this contract.
"File Number" is data.
"Initial Statement Date" is a time.
"Filer" is a person.
"Debtor" is a person.
"Secured Party" is a person.
"Filing Office" is a person.
"Collateral" is data.
"Digital Asset Collateral" is an amount.
"Reminder Fee" is an amount.
"Continuation Window Start" is a time.
"Continuation Statement Date" is a time.
"Continuation Statement Filing Number" is data.
"Lapse Date" is a time.
"Default" is a binary.
"Continuation Statement" is a binary.
"Termination Statement" is a binary.
"Termination Statement Time" is a time.
"Notification Statement" is a text.

The Filer fixes the Filing Office, fixes the Debtor, fixes the
Secured Party, and fixes the Collateral.

Clause: Certify.
The Filing Office may certify the File Number.

Clause: Set File Date.
```

The Filing Office **may fix the** Initial Statement Date **as the current time**.

Clause: Set Lapse.
The Filing Office **may fix the** Lapse Date.

Clause: Set Continuation Start.
The Filing Office **may fix the** Continuation Window Start.

Clause: Pay Fee.
The Secured Party **may pay a** Reminder Fee **into escrow**.

Clause: Notice.
The Filing Office **may fix the** Notification Statement.

Clause: Notify.
The Filing Office **may**, **if the** Continuation Window Start **has passed**, **send the** Notification Statement **to the** Secured Party.

Clause: Pay Escrow In.
The Debtor **may pay the** Digital Asset Collateral **into escrow**.

Clause: Fail to Pay.
The Secured Party **may declare** Default.

Clause: Take Possession.
The Filing Office **may**, **if** Default **is declared**, **pay the** Digital Asset Collateral **to the** Secured Party.

Clause: File Continuation.
The Secured Party **may file the** Continuation Statement.

Clause: Set Continuation Lapse.
The Filing Office **may**, **if the** Continuation Statement **is filed**, **fix the** Continuation Statement Date.

Clause: File Termination.
The Secured Party **may file a** Termination Statement, **and certify the** Termination Statement Time **as the then current time**.

Clause: Release Escrow.
The Filing Office **may**, **if the** Termination Statement **is filed**, **return the** Digital Asset Collateral **to the** Debtor.

Clause: Release Reminder Fee.
The Filing Office **may**, **if the** Termination Statement **is filed**, **return the** Reminder Fee **to the** Secured Party.

Clause: Termination Period.
"Termination Period" **is defined as** 365 **days after the** Termination Statement Time.

Clause: Terminate and Clear.
The Filing Office may, if the Termination Period has passed, terminate this contract.

Fig 4 – U.C.C. Financing Statement (Reyes)

The *Filer* will usually be a bank employee or outside counsel for the bank, the *Debtor* is the person taking out a loan, the *Secured Party* is the bank. The *Collateral* is the real-world object the debtor is putting up as security. It can also be cryptocurrency and similar, i.e., *Digital Asset Collateral.*

The *Reminder Fee* is a fee that the bank can pay to the filing office but is not required to pay. If the bank pays it, the filing office *may* send a *Notification* to remind the bank to put in a *Continuation Statement* every 5 years – i.e., during the *Continuation Window*. Else, the statement will lapse.

Note that while many features of this example merely effect existing rules related to the UCC filing system, this feature of the example represents a new proposal.

That there is no *obligation* described here is in keeping with blockchain powers. A blockchain smart contract cannot coerce anyone to do anything. It can only incentivize.[6]

If the debtor *Defaults*, all that is needed is that the bank says so. This is the intended way the law works, not a weakness introduced by the blockchainification. It is clearly an oracle-moment,[7] and a weird one because the bank as *Secured Party* can simply say that the money should now be theirs. This power, however, comes from the underlying contract provisions that are part of the secured loan documentation. The idea of ensuring that the *Filing Office* retains a role in relation to Digital Asset Collateral – in that without its action to *Give Possession* the collateral is not actually going to go to the bank – is an attempt to address the unique issues around custody and priority in the context of the *Digital Asset Collateral*. If the *Collateral* is a real-world item and not a *Digital Asset Collateral*, the normal rules related to self-help repossession apply. By law though, the moment the bank says so, they collateral is theirs. If the bank cheats, it's fraud.

A further feature of this smart contract is that it records exactly who said what when: including that and when the bank claimed that there was a default, which is the precondition to the seizing of the assets. This trail of information matters.

[6] See the *Holy Grail* paper at **https://lexon.org/papers** on how the concept of *obligation* can be added to digital contracts on a blockchain, and *Efficient Breach*, pg. 123

[7] Oracles, in blockchain-speak, are the gates through which facts from the outside world are made known within the confines of the digital blockchain data world.

Proposing law to be written in Lexon is pretty rad. Carla blew my mind with that. It's testament to Lexon's elegance, because it wasn't an intended use case for Lexon.

The core statute of the original U.C.C. model law that would be implemented by the above Lexon text, is:[8]

§ 9-502. CONTENTS OF FINANCING STATEMENT; RECORD OF MORTGAGE AS FINANCING STATEMENT; TIME OF FILING FINANCING STATEMENT.

(a) [Sufficiency of financing statement.]

Subject to subsection (b), a financing statement is sufficient only if it:

(1) provides the name of the debtor;

(2) provides the name of the secured party or a representative of the secured party; and

(3) indicates the collateral covered by the financing statement.

(b) [Real-property-related financing statements.]

Except as otherwise provided in Section 9-501(b), to be sufficient, a financing statement that covers as-extracted collateral or timber to be cut, or which is filed as a fixture filing and covers goods that are or are to become fixtures, must satisfy subsection (a) and also:

(1) indicate that it covers this type of collateral;

(2) indicate that it is to be filed [for record] in the real property records;

(3) provide a description of the real property to which the collateral is related [sufficient to give constructive notice of a

[8] **https://www.law.cornell.edu/ucc/9/9-502**

mortgage under the law of this State if the description were contained in a record of the mortgage of the real property]; and

(4) if the debtor does not have an interest of record in the real property, provide the name of a record owner.

(c) [Record of mortgage as financing statement.]

A record of a mortgage is effective, from the date of recording, as a financing statement filed as a fixture filing or as a financing statement covering as-extracted collateral or timber to be cut only if:

(1) the record indicates the goods or accounts that it covers;

(2) the goods are or are to become fixtures related to the real property described in the record or the collateral is related to the real property described in the record and is as-extracted collateral or timber to be cut;

(3) the record satisfies the requirements for a financing statement in this section, but

(A) the record need not indicate that it is to be filed in the real property records; and

(B) the record sufficiently provides the name of a debtor who is an individual if it provides the individual name of the debtor or the surname and first personal name of the debtor, even if the debtor is an individual to whom Section 9-503(a)(4) applies; and

(4) the record is [duly] recorded.

(d) [Filing before security agreement or attachment.]

A financing statement may be filed before a security agreement is made or a security interest otherwise attaches.

Fig 5 – original financing statement model law

CODE & TREES

Below, three ASTs are presented, each followed by the respective source code that they were built[9] from.

First, for a Lexon escrow contract.

Second, for the Solidity source that is generated as output when the Lexon compiler processes the first.

Third, an independent approach at the same problem, programmed natively in Solidity without any Lexon involved. The last example may serve as contrast to see how much less readably structured Solidity looks when it is not generated from Lexon.

The point throughout is to demonstrate how the ASTs express roughly the same functionality but *on different levels of abstraction.* And how the Solidity ASTs are concerned with details that make them lose the 'meaning' that is visible in the higher-level Lexon AST.

[9] AST graphs in this document were not created in an automated process but manually. The Lexon compiler can create AST graphs for any input though using the *--tree* option.

LEXON CODE: WALK-THROUGH

Lexon AST

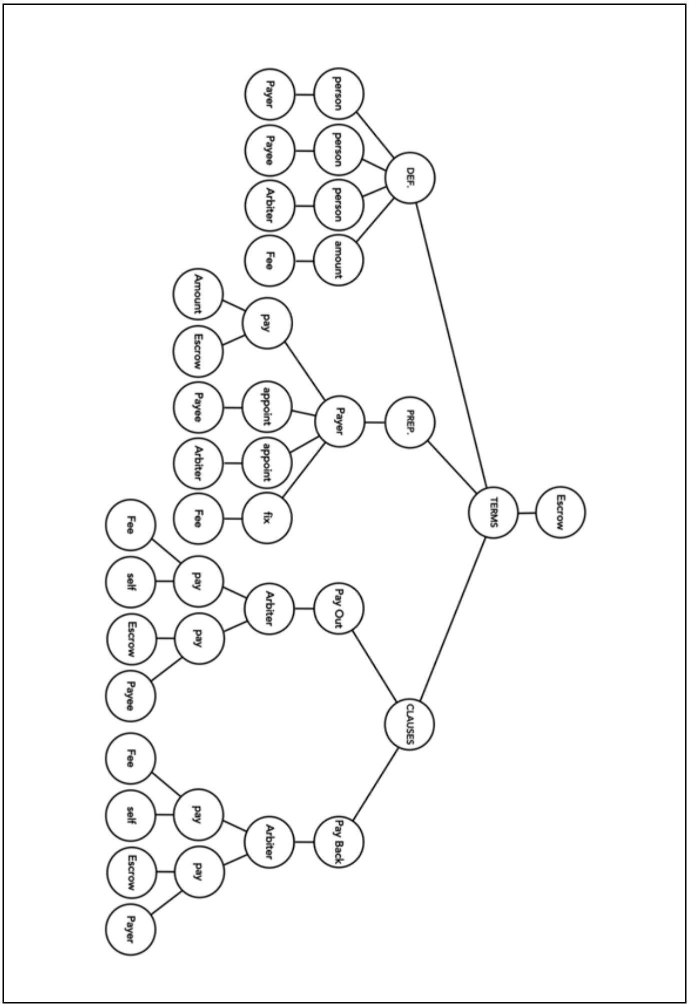

Fig 6 – Lexon AST for Escrow example

This AST is best understood by comparing it to its Lexon code (pg. 52). The major parts (**DEF**[INITIONS], **RECIT**[ALS], **CLAUSES**) are positioned left to right in this AST, while in the source code they are appearing top to bottom, without explicit labels. A blank line is separating them.

The name of the program – or smart contract – is the label of the top-most node: **Escrow**.

The second AST level has only one node, the **TERMS**. The keyword **TERMS** is optional in some cases and therefore also missing in the source code for this example.

The next layer of the AST consists of the three parts of the **TERMS**: **DEFINITIONS**, **RECITALS** and **CLAUSES**. The **DEFINITIONS** start at **"Payer"**, the **PREPARATION** at **The Payer pays**, and the **CLAUSES** with the first **CLAUSE**. In the AST, the respective parts are simply everything under the node with respective label (**DEF.**, **RECIT.**, **CLAUSES**). The reason that labels of that name do not exist in the source is that they would be mostly redundant. The statements in the **DEFINITIONS** section looks a certain way, everything between them and the **CLAUSES** then are **RECITALS**. And each *individual* clause of course is marked by the keyword **CLAUSE**.

The source code is the best explanation for what the nodes and edges of the AST mean. For example, the left-most vertical in the AST, **DEF. -person-Payer** is the definition of the name **Payer** as a **person**. It is derived from the parsing of the line **"Payer" is a person**. After this, the meaning of the remaining nodes under **DEF.** is obvious.

In the AST, left of the center there is the vertical **RECIT.-Payer-pay-Amount**; with **Escrow** added under **pay**, too. This represents the source code **The Payer pays an Amount into escrow.** The remaining nodes under **RECIT.** follow the same logic. These lines are what is performed to set the contract up.

The AST verticals under **Pay Out** and **Pay Back** match **CLAUSE: Pay Out** and **CLAUSE: Pay Back** and are best explained by the source code below. **The Arbiter may,** signifies that *only* the **Arbiter** can invoke this clause. If anyone else tries, they will only get an error. The Arbiter is going to authenticate themselves with the private key that belongs to the blockchain address that is the basis for the identity of **Arbiter**. In Pay Out the **Arbiter** first **pays the Fee to themselves** that the **Payer** has set in the preparational phase of the contract performance. This **Fee** is paid from the escrow, i.e., deducted from the initial payment the **Payer made into escrow** during the Preparation. After the **Fee** is taken care of, the **remainder of the escrow** is sent to the **Payee**. **Pay Back** works the same with the one exception that the money goes back to the **Payer**.

```
LEX Paid Escrow.

“Payer” is a person.
“Payee” is a person.
“Arbiter” is a person.
“Fee” is an amount.

The Payer pays an Amount into escrow, appoints the Payee,
appoints the Arbiter, and also fixes the Fee.

CLAUSE: Pay Out.
The Arbiter may pay from escrow the Fee to themselves,
and afterwards pay the remainder of the escrow to the Payee.

CLAUSE: Pay Back.
The Arbiter may pay from escrow the Fee to themselves,
and afterwards return the remainder of the escrow to the
Payer.
```

Fig 7 – Lexon Source Code for Escrow Example

GENERATED SOLIDITY: WALK THROUGH

For completeness, this is the AST and source code of the Solidity code that is generated by the Lexon compiler from the Lexon Escrow example given above.

Generated Solidity: AST

The Solidity source that Lexon produces, of course loses the vocabulary and grammar of the Lexon code that it was created from. But it keeps the *document structure* and *names*. One can easily identify the four main branches again, now called **elements**, **construct**, **Pay Out** and **Pay Back**. But immediately below those, this AST shifts to a more granular, data-leaning concern and loses the clarity and all semblance to the Lexon AST.

As opposed to Lexon, there is no discernable subject-verb-object relationship. The code, and the tree, deal instead in permission details (**public**, **payable**, **private**) that are far from intuitive. It also takes care of requirements that arise from *variable scopes* (most of the **assigns**) and it is heavy on micro granular **calls** of functions and dereferencing of object elements (**lookup**). All these are purely programming-specific concerns that are not related to natural language. They dominate the shape of this AST.

To be clear, on the way from the Lexon code to the resulting smart contract running on the Ethereum mainnet, there are *two* compilation steps: from Lexon to Solidity, and from Solidity to *op codes* for the Ethereum virtual machine (EVM). Accordingly, an AST is created on two occasions: first the Lexon one shown above and then the Solidity one immediately below.

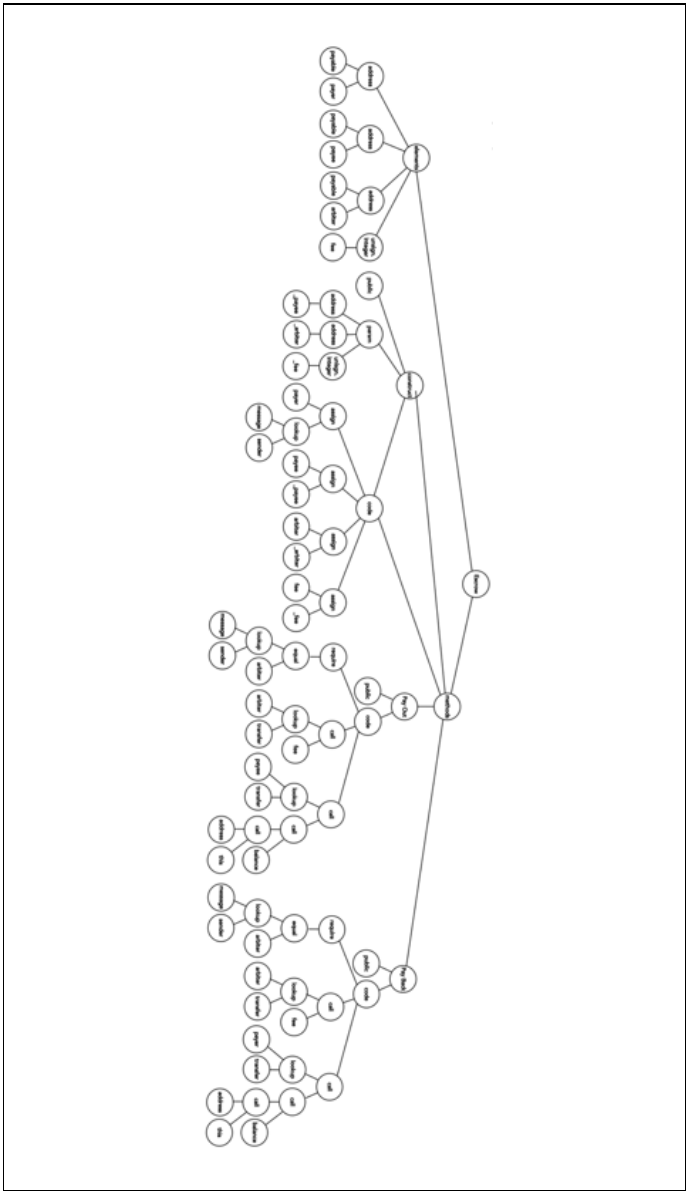

Fig 8 – Solidity AST for Escrow, Generated by Lexon

Generated Solidity: Source Code

```
pragma solidity ^0.5.0;

contract Escrow {
    address payable payer;
    address payable payee;
    address payable arbiter;
    uint fee;

    constructor(address payable _payee, address
payable _arbiter, uint _fee) public {
        payer=msg.sender;
        payee=_payee;
        arbiter=_arbiter;
        fee=_fee;
    }

    function PayOut() public {
        require(msg.sender == arbiter);
        arbiter.transfer(fee);
        payee.transfer(address(this).balance);
    }

    function PayBack() public {
        require(msg.sender == arbiter);
        arbiter.transfer(fee);
        payer.transfer(address(this).balance);
    }
 }
```

Fig 9 – Solidity Source Code for Escrow, Generated by Lexon

It is relative straight forward to match Solidity AST nodes and Solidity source code. And also, to understand their respective meaning from the Lexon source code shown before.

NATIVE SOLIDITY: WALK THROUGH

For comparison, the following is an abbreviated version of a third party example for an Solidity escrow contract.[10] It was chosen because its maker could not have any knowledge of Lexon and it illustrates well how different a 'hand-made' Solidity smart contract is *structured* that tackles pretty much the same task, i.e., implementing a simple escrow on the blockchain.

Expectably, as can be seen from the source code given below (Fig 11), this smart contract source has a different roster of functions: beyond the **constructor**, they are **deposit**, **accept**, **cancel** and **kill**. These names, of course, appear as main nodes in the AST below.

The function **deposit** is used to pay into the escrow. **accept** is a combined entry point to give consent to the pay out and to actually facilitate it, **cancel** is a similarly structured function to allow for the unanimous termination of the agreement, paying the money back. **kill** allows the controller of the escrow to terminate the agreement.

This code is yet heavier than the Lexon-generated Solidity (cf. above Fig 9, pg. 55) regarding the use of conditional branching (**if**), de-referentiation (**lookup**) and function **calls**.

This source does of course not share the document structure of the Lexon code example above. It has its own sequence and logic, and this highlights the lower level of abstraction that is applied when writing Solidity code.

[10] Pranav K. – https://medium.com/@pranav.89/smart-contracting-simplified-escrow-in-solidity-ethereum-b19761e8fe74

Native Solidity: AST

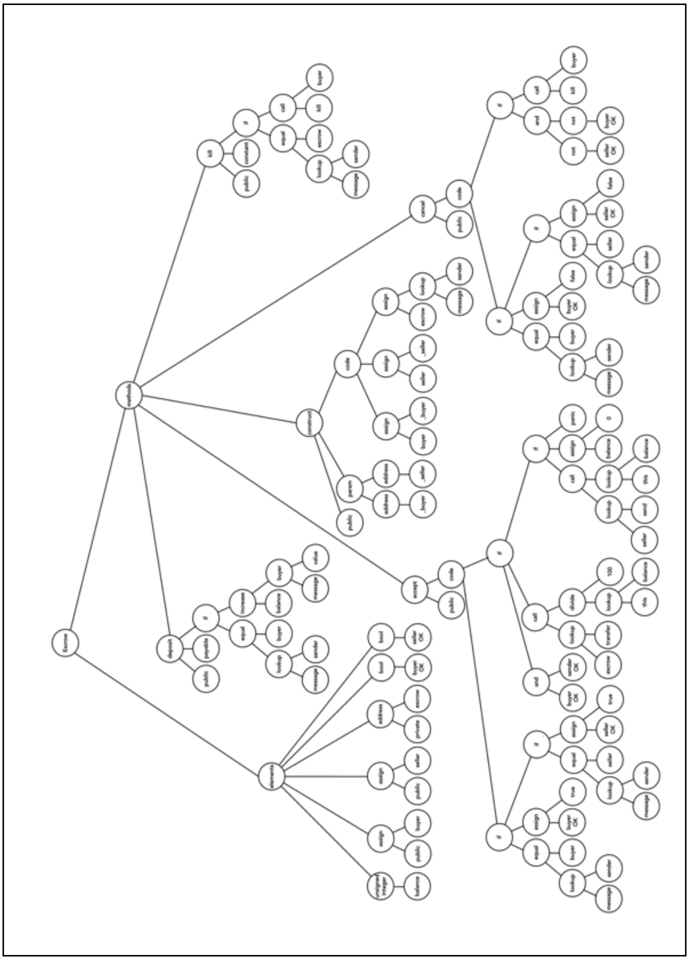

Fig 10 – Solidity AST for Similar Escrow Example (alternate layout)

Native Solidity: Source Code

```
contract Escrow {
    uint balance;
    address public buyer;
    address public seller;
    address private escrow;
    bool buyerOk;
    bool sellerOk;
    constructor(address _buyer,
      address _seller) public {
        buyer = _buyer;
        seller = _seller;
        escrow = msg.sender;
    }
    function accept() public {
        if (msg.sender == buyer){
            buyerOk = true;
        } else if (msg.sender == seller){
            sellerOk = true;
        }
        if (buyerOk && sellerOk){
            // we are sending ourselves
            // (contract creator) a fee
            escrow.transfer(this.balance /100);
            if (seller.send(this.balance)) {
                balance = 0;
            } else {
                throw;
            }
        }
    }
    function deposit() public payable {
        if (msg.sender == buyer) {
            balance += msg.value;
        }
    }
    // if both buyer & seller would like
    // to cancel, money is returned to buyer
    function cancel() public {
        if (msg.sender == buyer){
            buyerOk = false;
        } else if (msg.sender == seller){
            sellerOk = false;
        }
        if (!buyerOk && !sellerOk){
            selfdestruct(buyer);
        }
    }
    function kill() public constant {
        if (msg.sender == escrow) {
            selfdestruct(buyer);
        }
    }
}
```

Fig 11 – Solidity Source Code for Similar Escrow Example

FIGURES